線段樹 (Segment Tree)

a data structure that stores information
about **array intervals** as a tree

http://pisces.ck.tp.edu.tw/~peng/index.php?action=showfile&file=f220f2d6b33ae091978ebf59d2af5908bc8190b51

http://pisces.ck.tp.edu.tw/~peng/index.php?action=showfile&file=fb07452d4b6e4e4100d6a665d4313e753ee4ccebd

Pei-yih Ting

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E
- HDU 1199, 1754, 3397, 4027, 5249, 5493, 1255, 1166, 1698, 1394, 3911, 3074, 4007, 4046, 2492, 3308, 2781, 3642, 3255, 3874, 4107, 4614, 1086

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E
- HDU 1199, 1754, 3397, 4027, 5249, 5493, 1255, 1166, 1698, 1394, 3911, 3074, 4007, 4046, 2492, 3308, 2781, 3642, 3255, 3874, 4107, 4614, 1086
- POJ 21528, 3368, 3264, 1151, 1177, 3277, 2777, 2528, 3468, 2828, 2886, 2352, 3667, 2892, 2180, 2482, 3225, 2991

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E
- HDU 1199, 1754, 3397, 4027, 5249, 5493, 1255, 1166, 1698, 1394, 3911, 3074, 4007, 4046, 2492, 3308, 2781, 3642, 3255, 3874, 4107, 4614, 1086
- POJ 21528, 3368, 3264, 1151, 1177, 3277, 2777, 2528, 3468, 2828, 2886, 2352, 3667, 2892, 2180, 2482, 3225, 2991
- ZOJ 1610, 2301, 3279, 2706, 2859

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E
- HDU 1199, 1754, 3397, 4027, 5249, 5493, 1255, 1166, 1698, 1394, 3911, 3074, 4007, 4046, 2492, 3308, 2781, 3642, 3255, 3874, 4107, 4614, 1086
- POJ 21528, 3368, 3264, 1151, 1177, 3277, 2777, 2528, 3468, 2828, 2886, 2352, 3667, 2892, 2180, 2482, 3225, 2991
- ZOJ 1610, 2301, 3279, 2706, 2859
- ZeroJudge b492, b493, e409, e437, e457, f033, f315, f986, f994, f995, i963, f999, h074, h873, h936, h990

應用題目

- UVa 11532, 11345, 11235, 12086, 12798, 11402, 11350, 1232, 10909, 11990, 12436, 12769, 12669, 21342, 6436, 11311, 11983, 12912
- Codeforces 339D, 295A, 459D, 482B, 772C, 220B, 380C, 495B, 272C, 61E, 383C, 311A, 52C, 242E, 629D, 847B, 474F, 914D, 627B, 920F, 438D, 343D, 703D, 375D, 877E, 501D, 474E, 514D, 43D, 755D, 558E, 292E, 830B, 301D, 597C, 369E, 540E, 863E, 446C, 145E, 580E, 515E, 444C, 500E, 396C, 121E, 19D, 610D, 220E, 813E, 484E, 276E, 56E, 338E, 115E, 501E, 594D, 240F, 516D, 610E, 895E, 351D, 173E, 455E, 117E
- HDU 1199, 1754, 3397, 4027, 5249, 5493, 1255, 1166, 1698, 1394, 3911, 3074, 4007, 4046, 2492, 3308, 2781, 3642, 3255, 3874, 4107, 4614, 1086
- POJ 21528, 3368, 3264, 1151, 1177, 3277, 2777, 2528, 3468, 2828, 2886, 2352, 3667, 2892, 2180, 2482, 3225, 2991
- ZOJ 1610, 2301, 3279, 2706, 2859
- ZeroJudge b492, b493, e409, e437, e457, f033, f315, f986, f994, f995, i963, f999, h074, h873, h936, h990
- A2OJ list of 162 problems: https://a2oj.com/Category.jsp?ID=25

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作

  1. C i a : 將第 i 個位置的數字更換為 a

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作
  1. C i a : 將第 i 個位置的數字更換為 a
  2. Q i j : 詢問區間 i 到 j 之間數字的**最大值**

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作
  1. C i a : 將第 i 個位置的數字更換為 a
  2. Q i j : 詢問區間 i 到 j 之間數字的**最大值**
- Range minimum Query (RmQ): … **最小值**

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作
  1. C i a : 將第 i 個位置的數字更換為 a
  2. Q i j : 詢問區間 i 到 j 之間數字的**最大值**
- Range minimum Query (RmQ): … **最小值**
- Range Sum Query (RSQ): … **總和**

基本問題

- Range Maximum Query (RMQ): 給定一個序列共 n 個整數，O(lg n) 時間完成下列操作
  1. C i a : 將第 i 個位置的數字更換為 a
  2. Q i j : 詢問區間 i 到 j 之間數字的**最大值**
- Range minimum Query (RmQ): … **最小值**
- Range Sum Query (RSQ): … **總和**

關鍵: 需要在資料不斷更新的情況下
以 O(lg n) 的效率完成更新及查詢

範例: RmQ, RMQ, RSQ

A[i]							
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3

RmQ(3,4)=5

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3

RmQ(3,4)=5

RmQ(0,0)=8

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:

(0,6)

A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

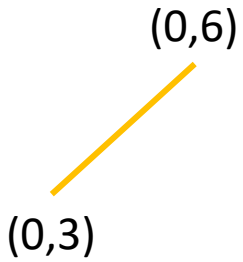
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

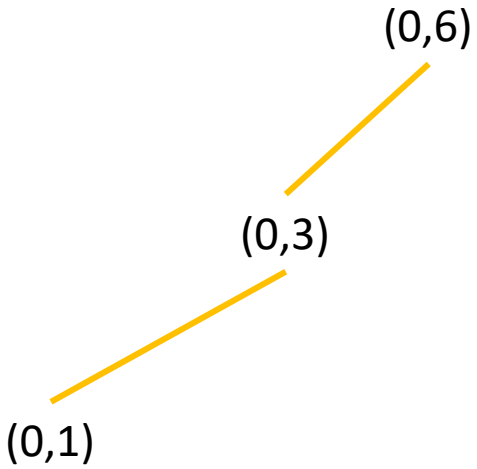
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

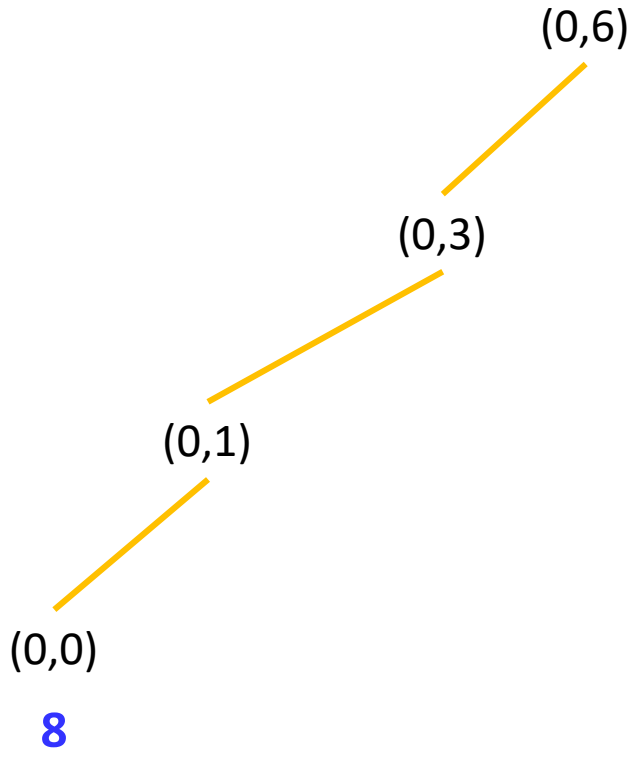
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

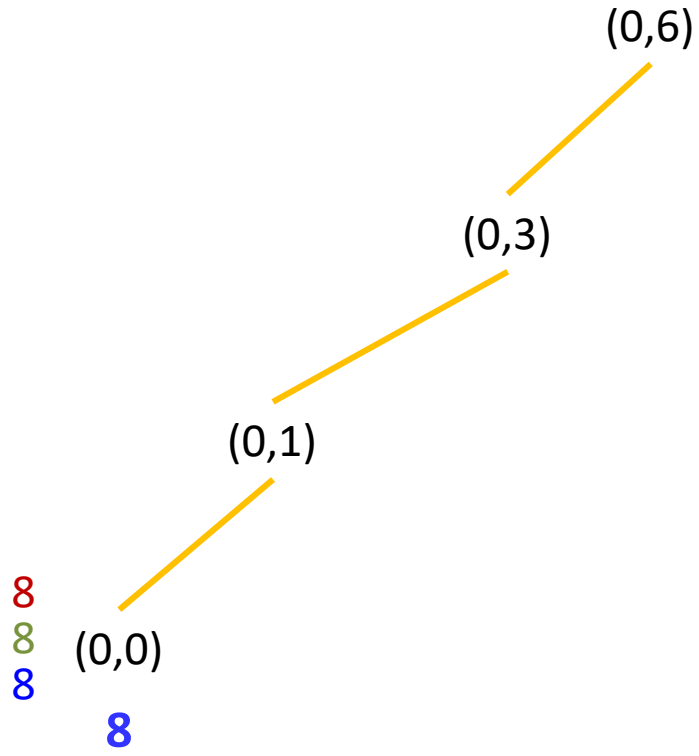
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

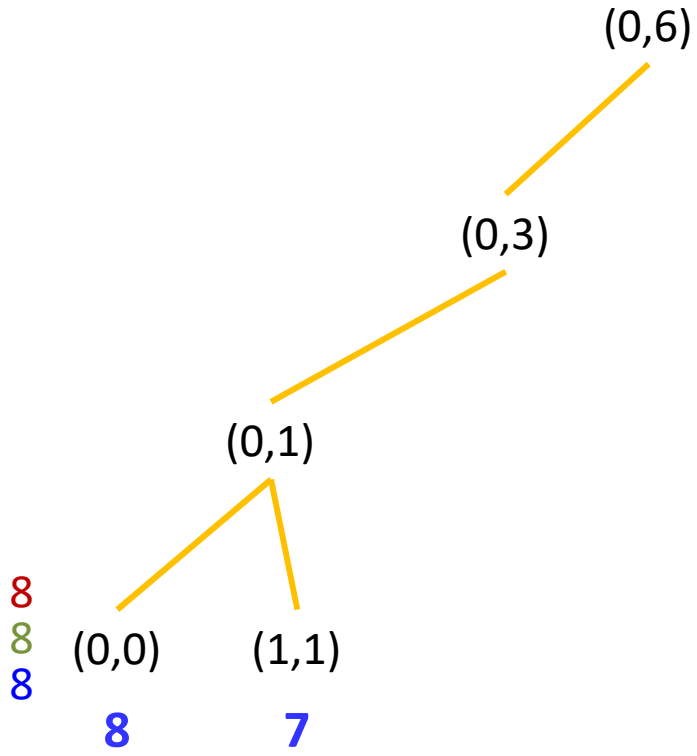
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

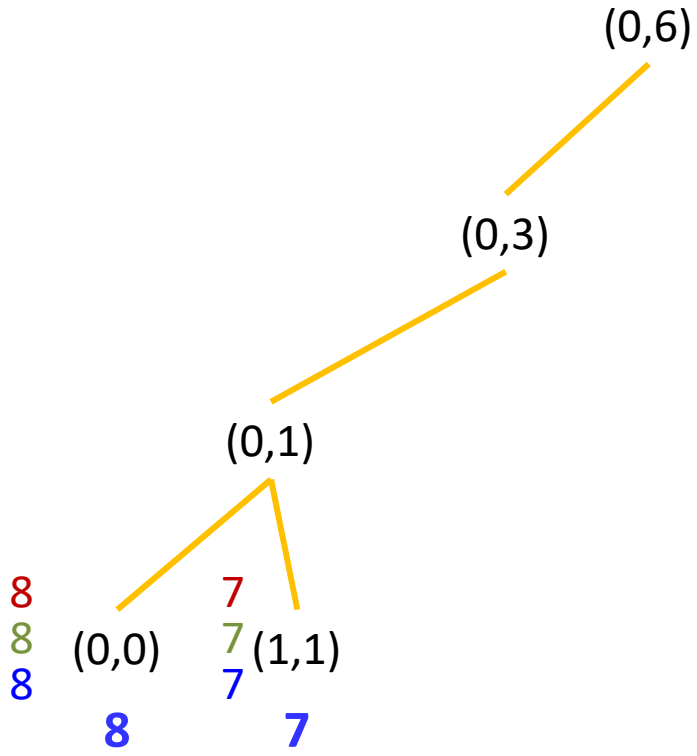
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

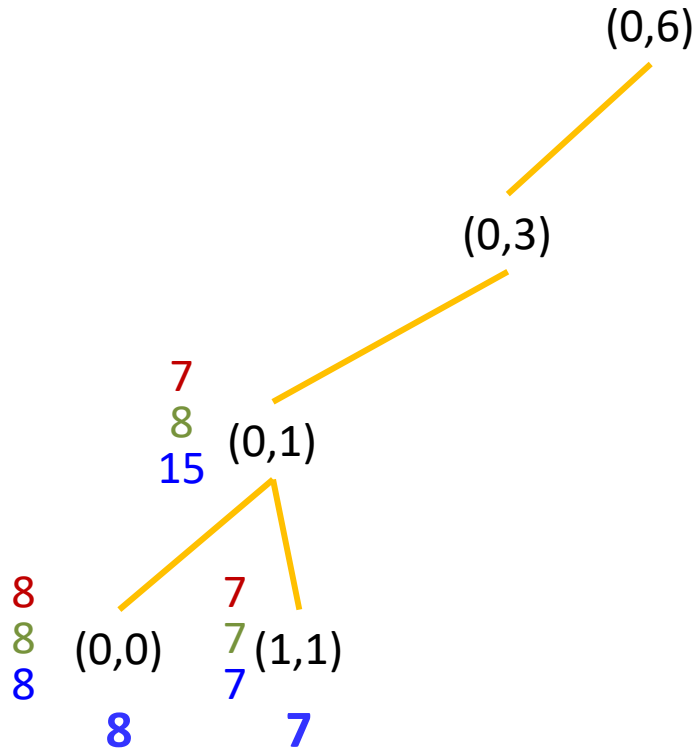
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

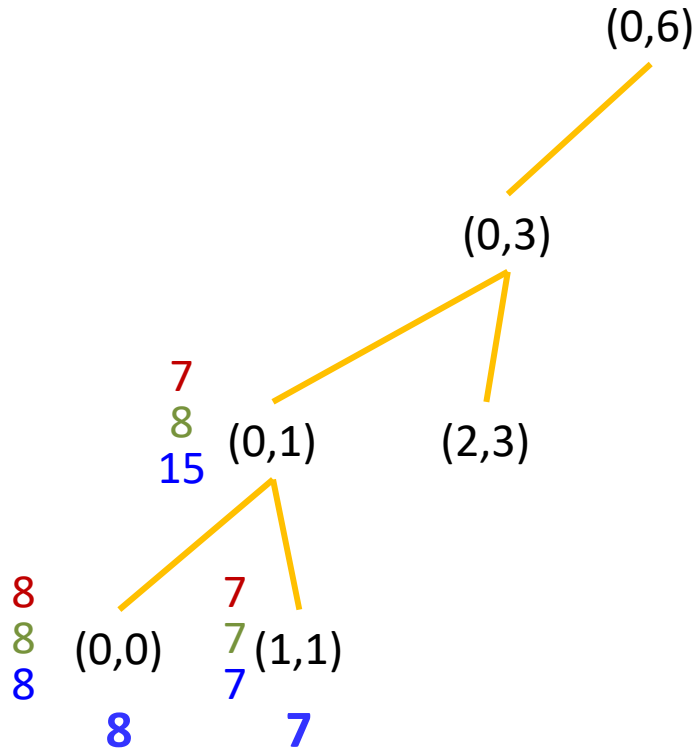
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

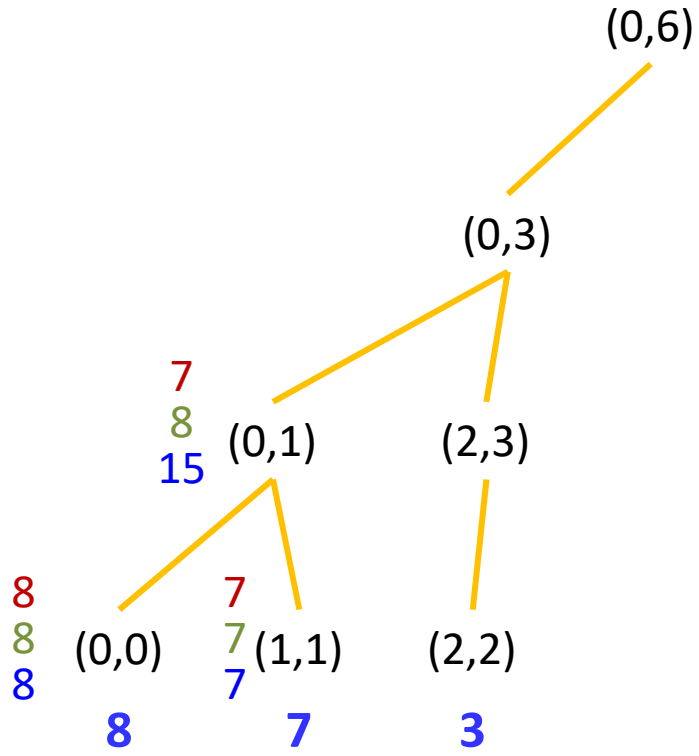
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

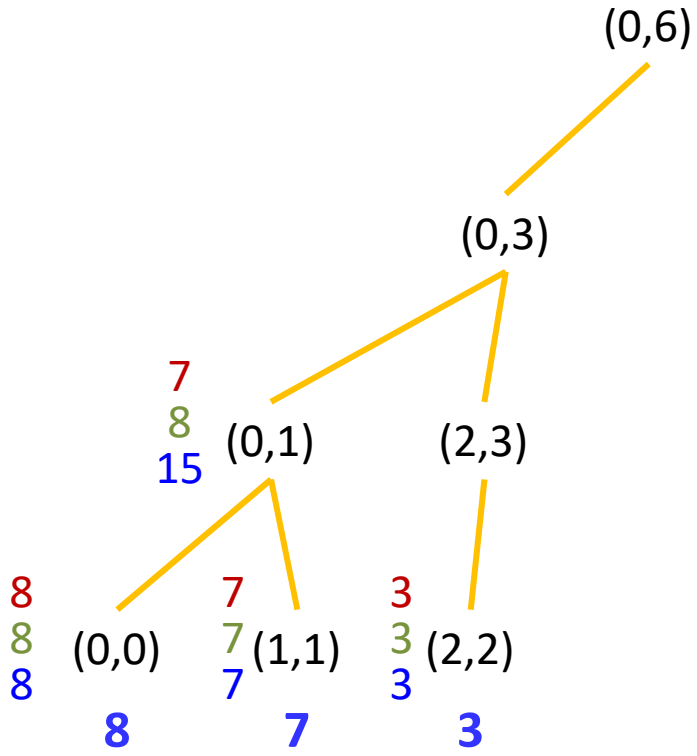
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

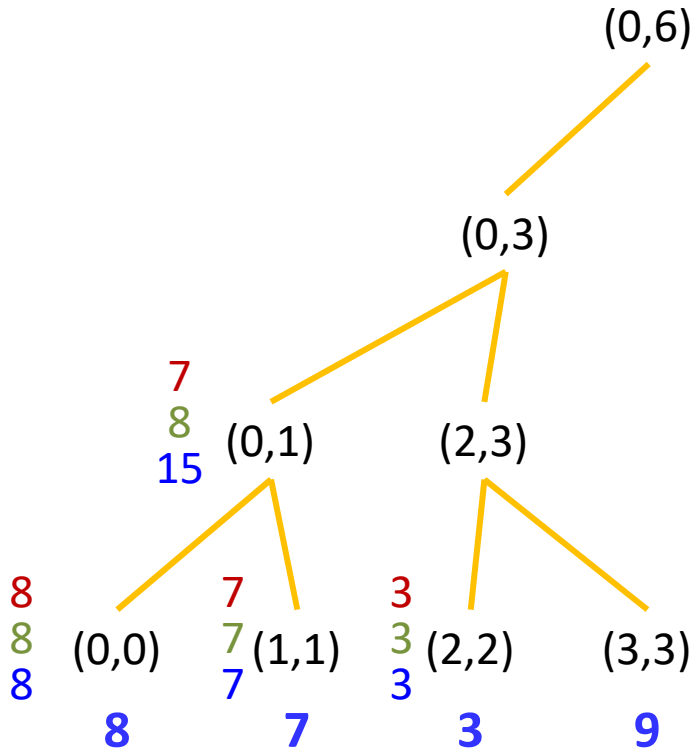
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

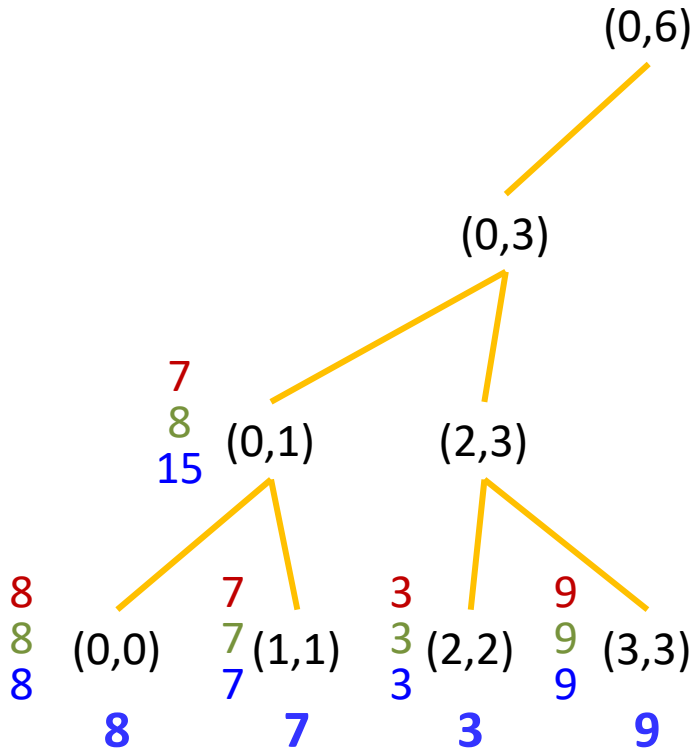
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

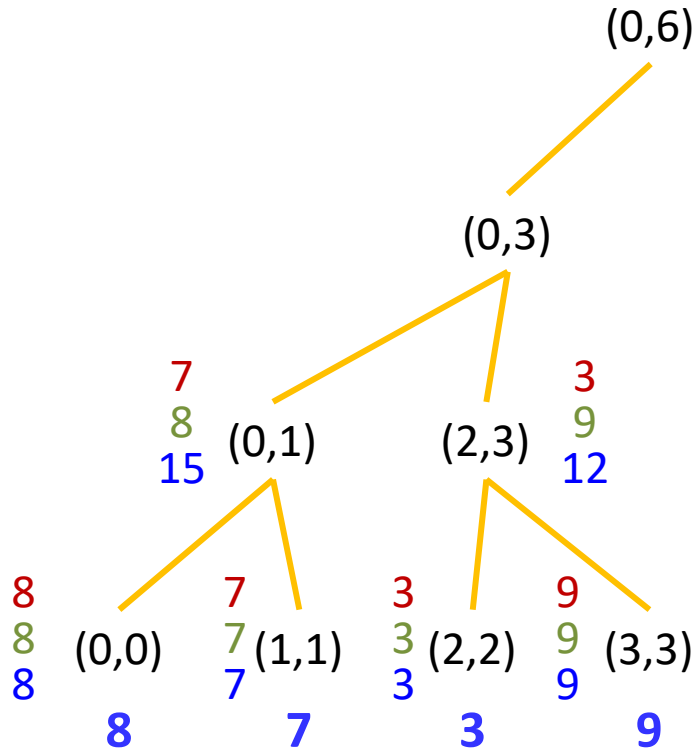
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

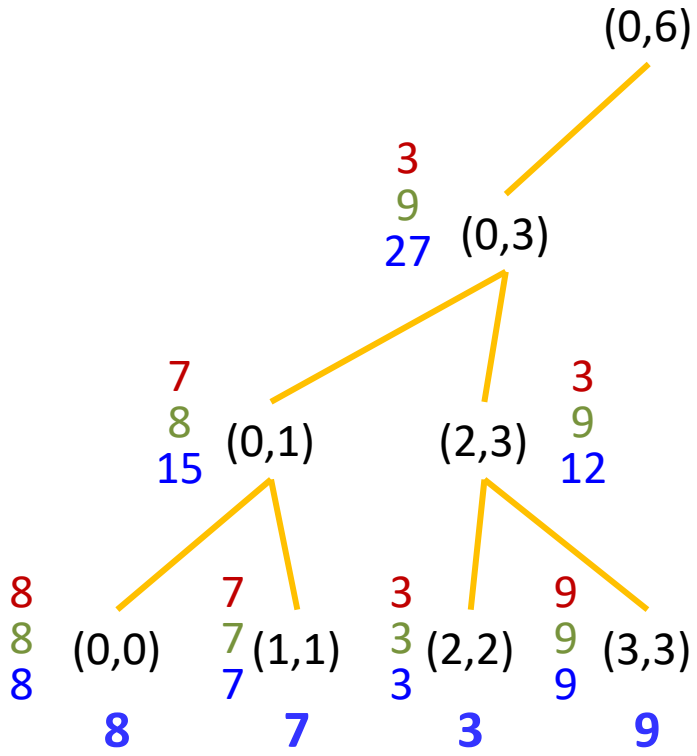
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

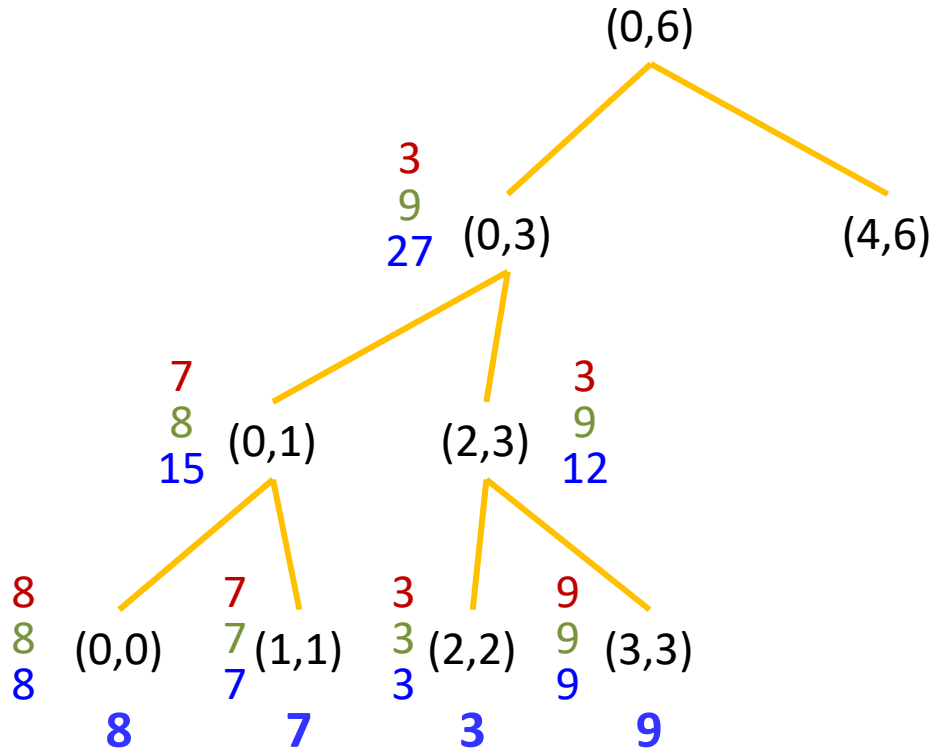
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

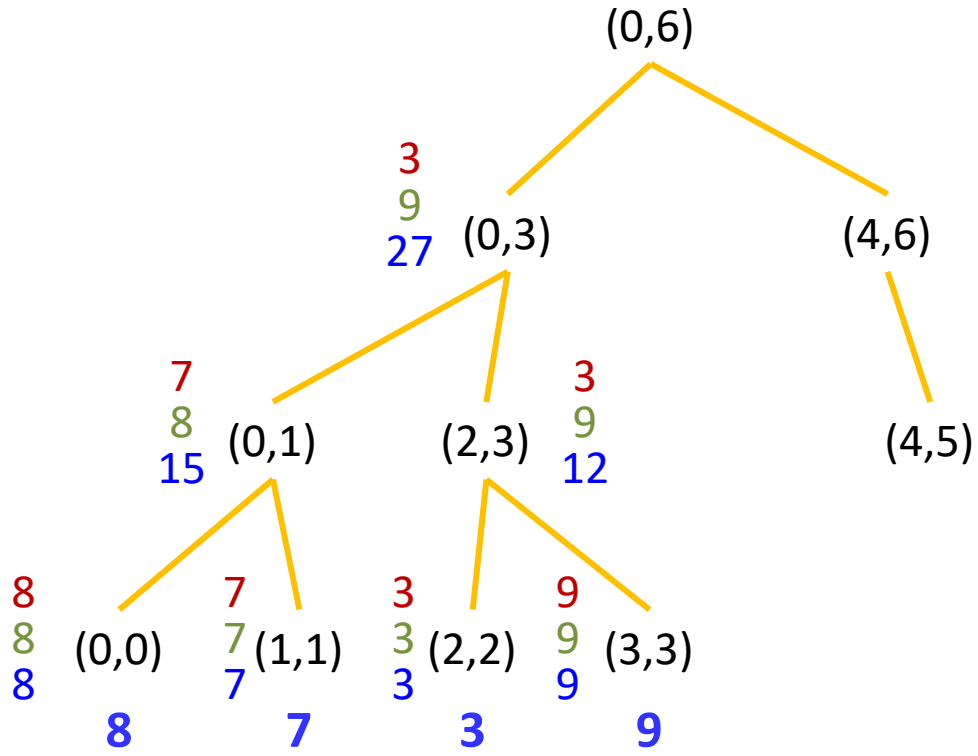
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

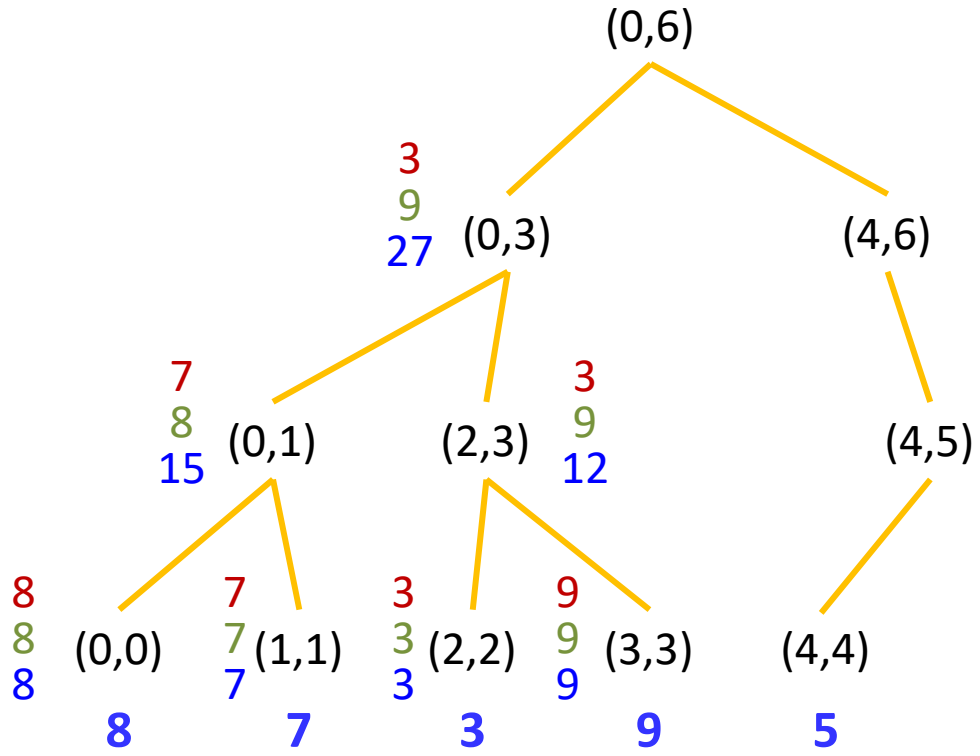
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

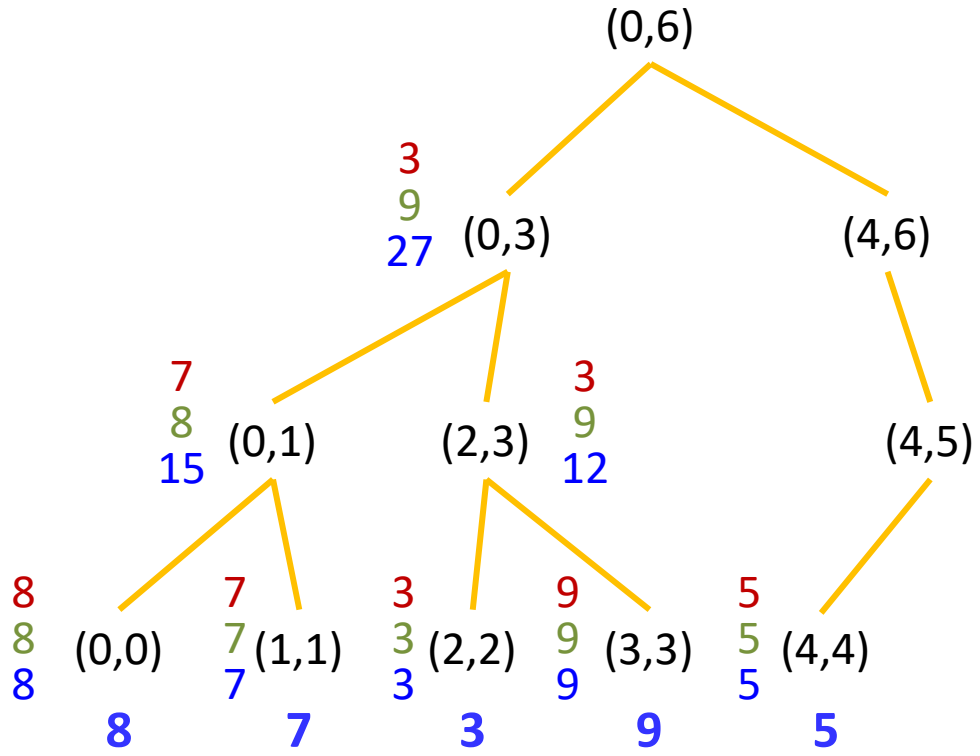
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

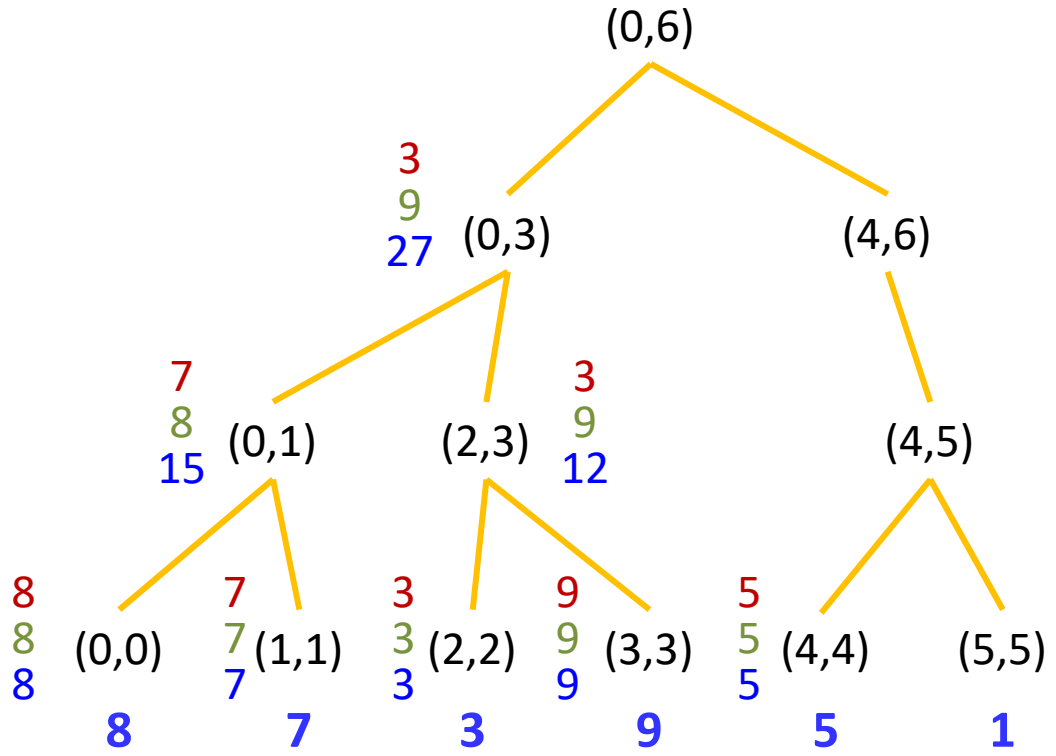
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

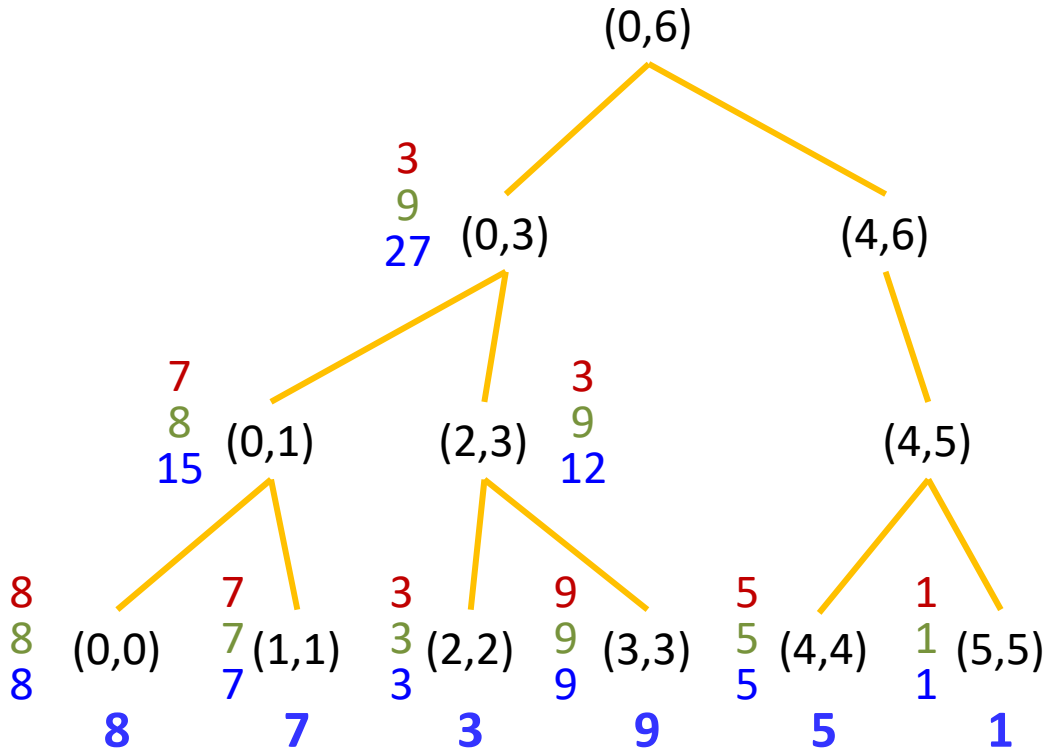
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

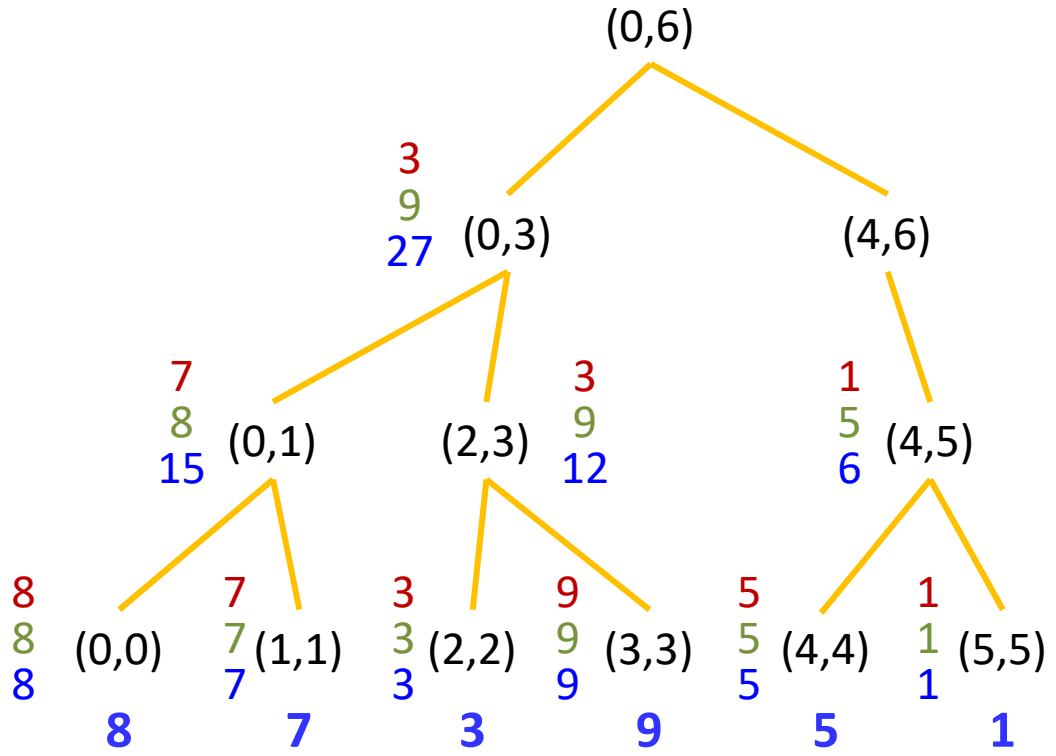
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

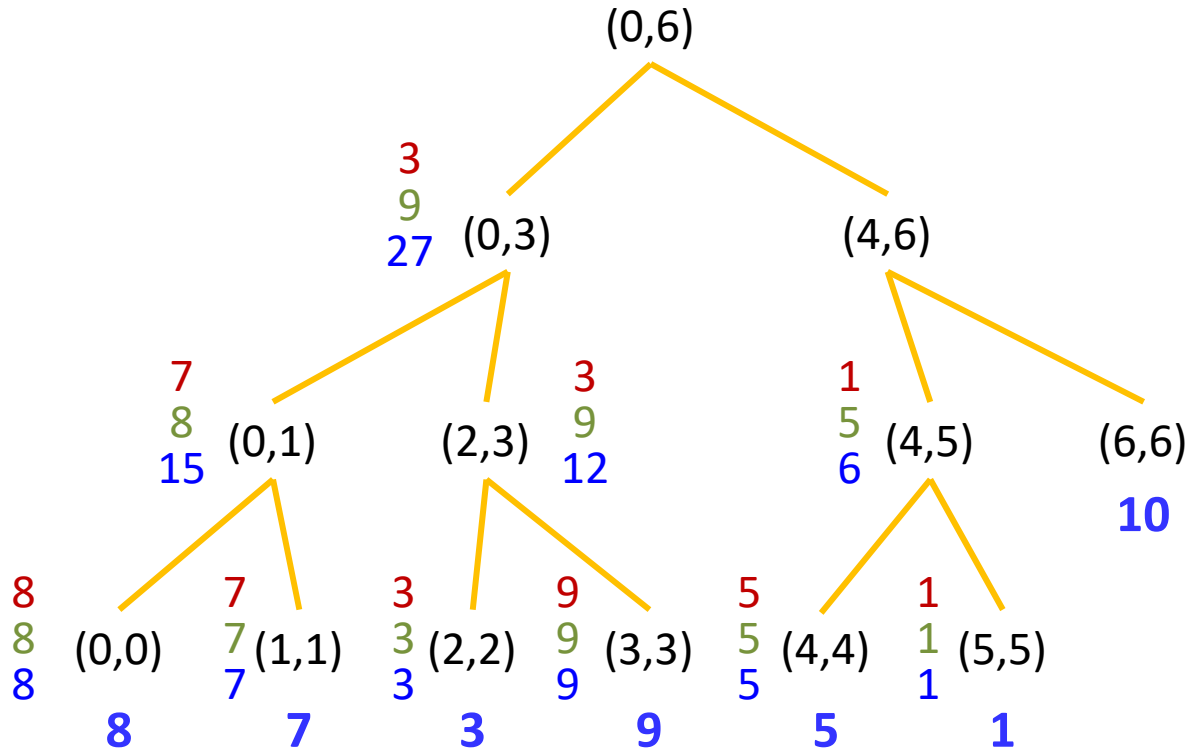
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

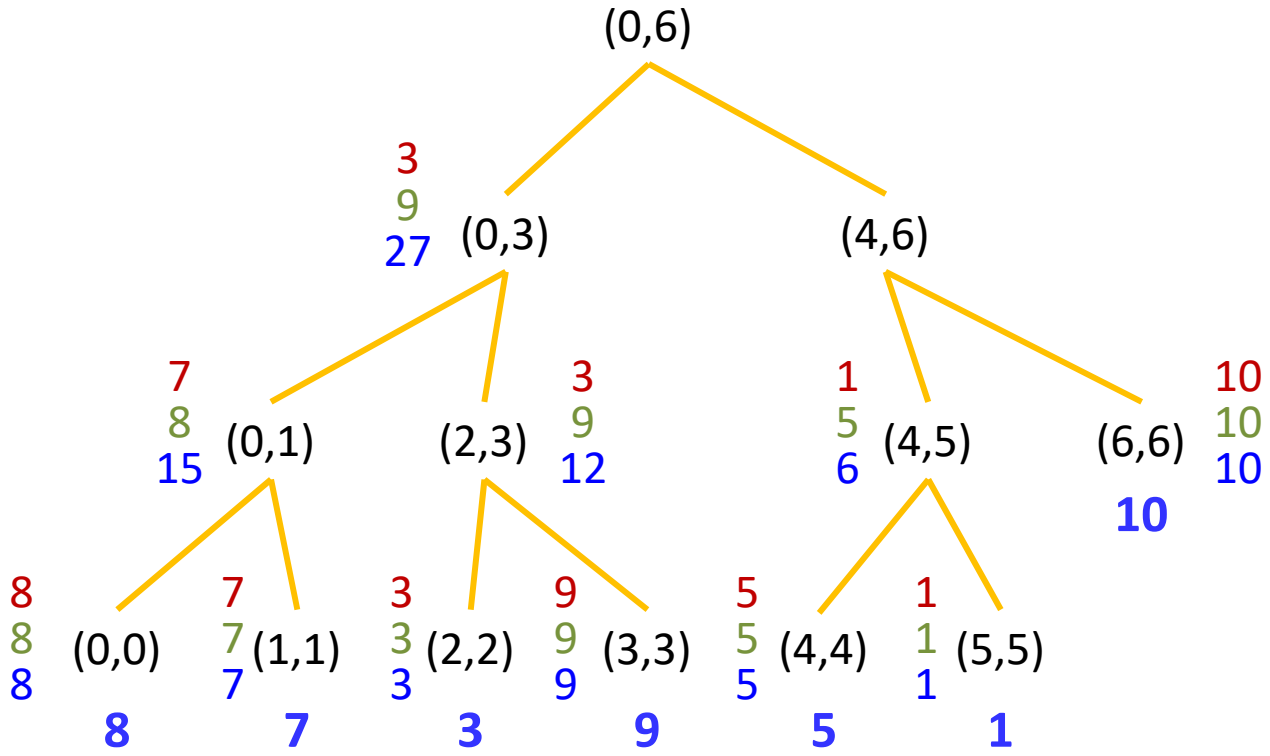
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

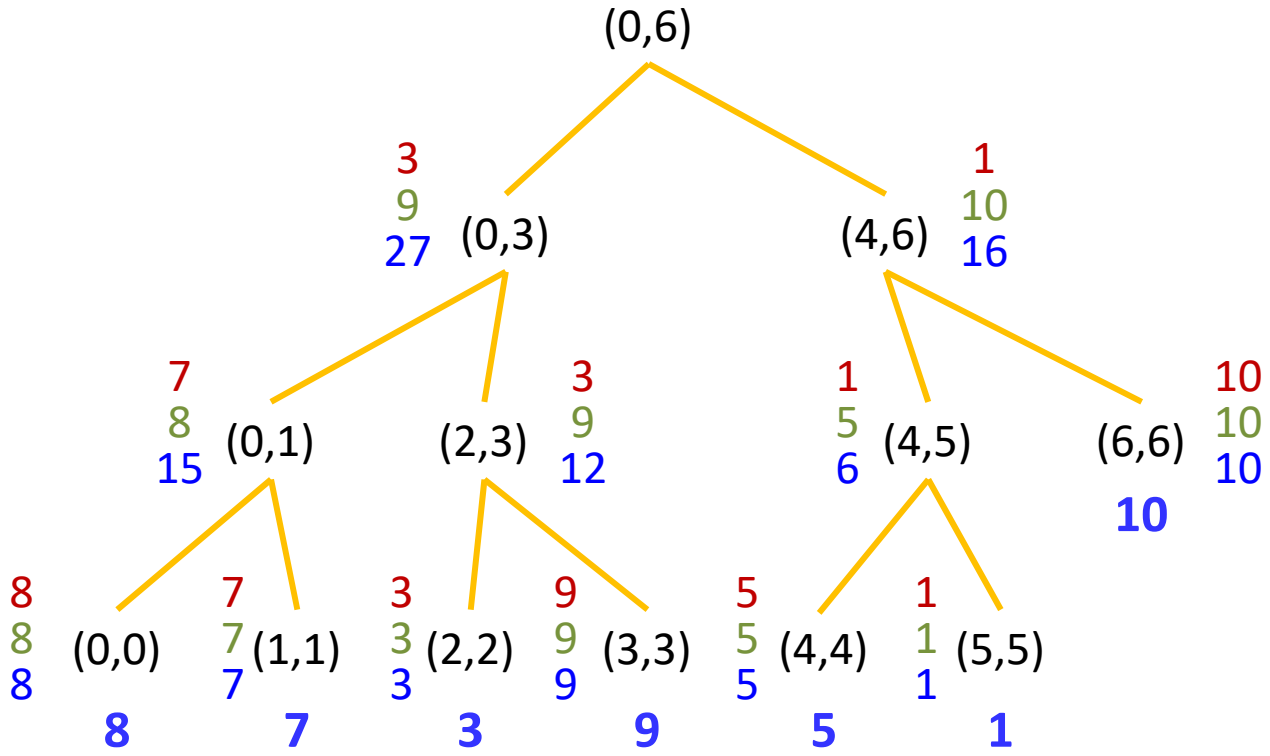
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

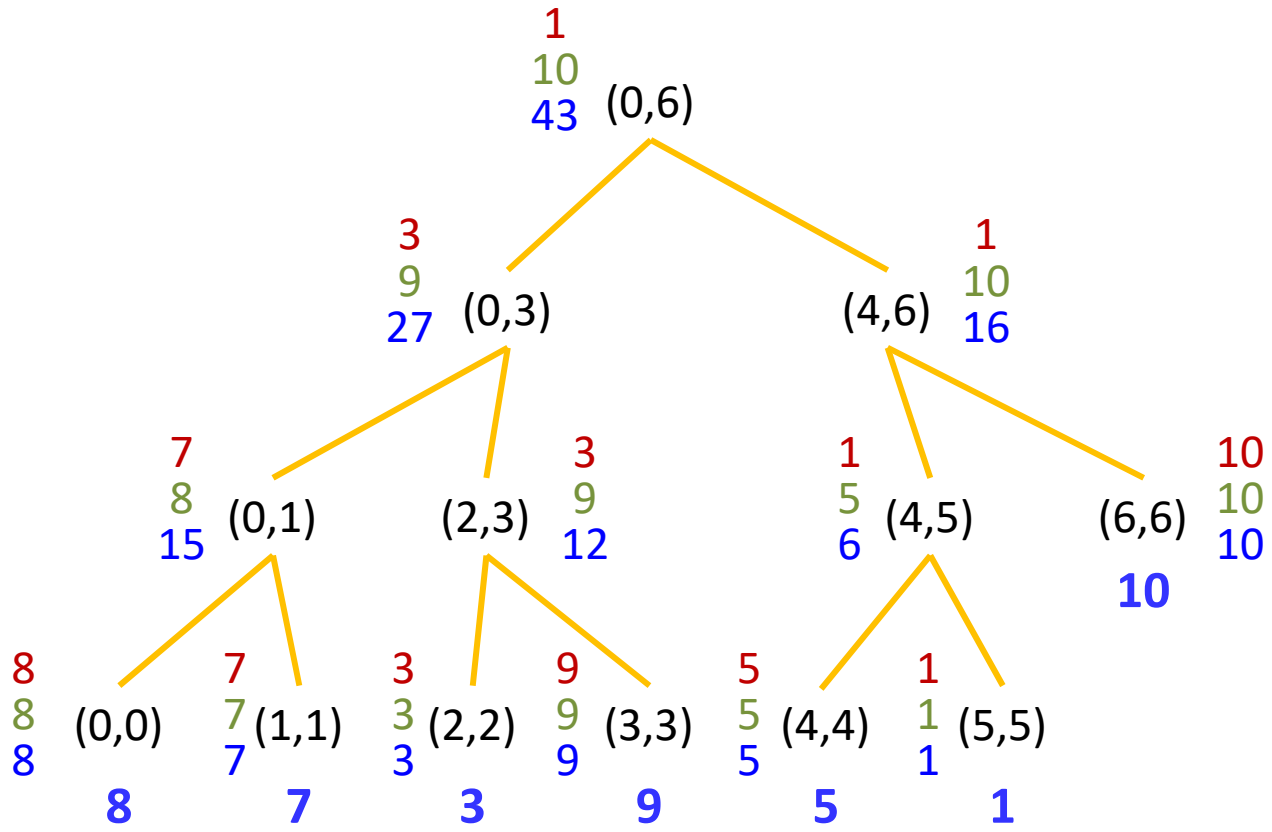
範例: RmQ, RMQ, RSQ

RmQ(1,3)=3
RmQ(3,4)=5
RmQ(0,0)=8
RmQ(0,1)=7
RmQ(0,6)=1

RMQ(1,3)=9
RMQ(3,4)=9
RMQ(0,0)=8
RMQ(0,1)=8
RMQ(0,6)=10

RSQ(1,3)=19
RSQ(3,4)=14
RSQ(0,0)=8
RSQ(0,1)=15
RSQ(0,6)=43

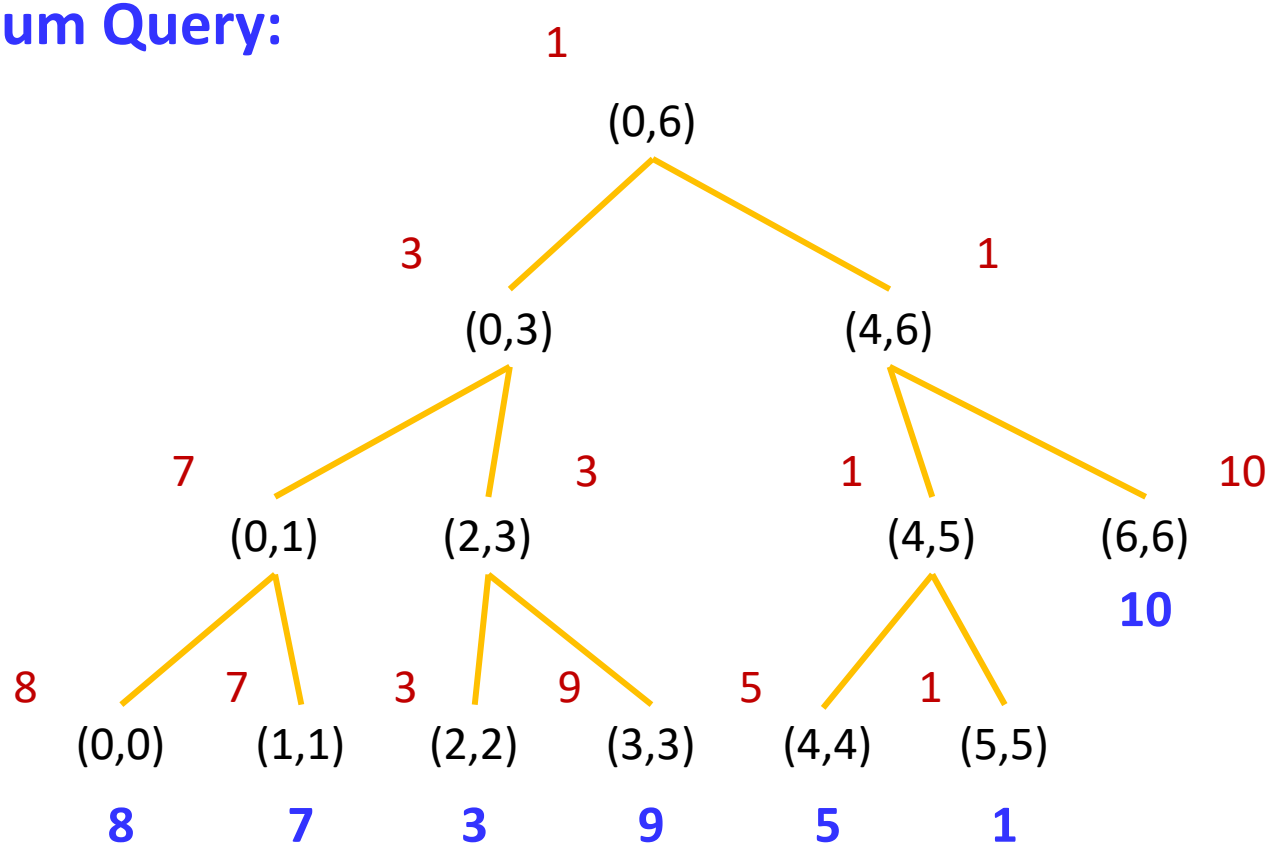
Building Segment Tree:



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

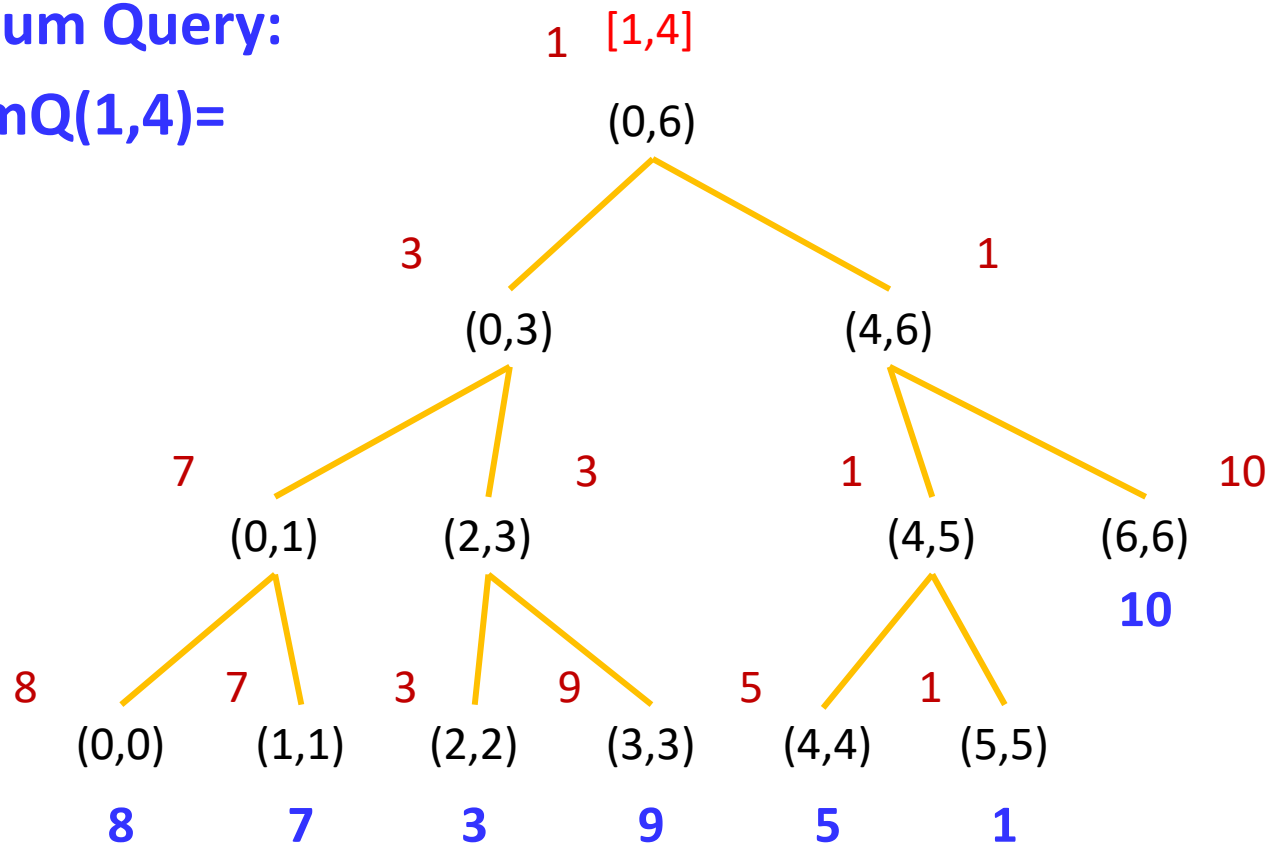


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

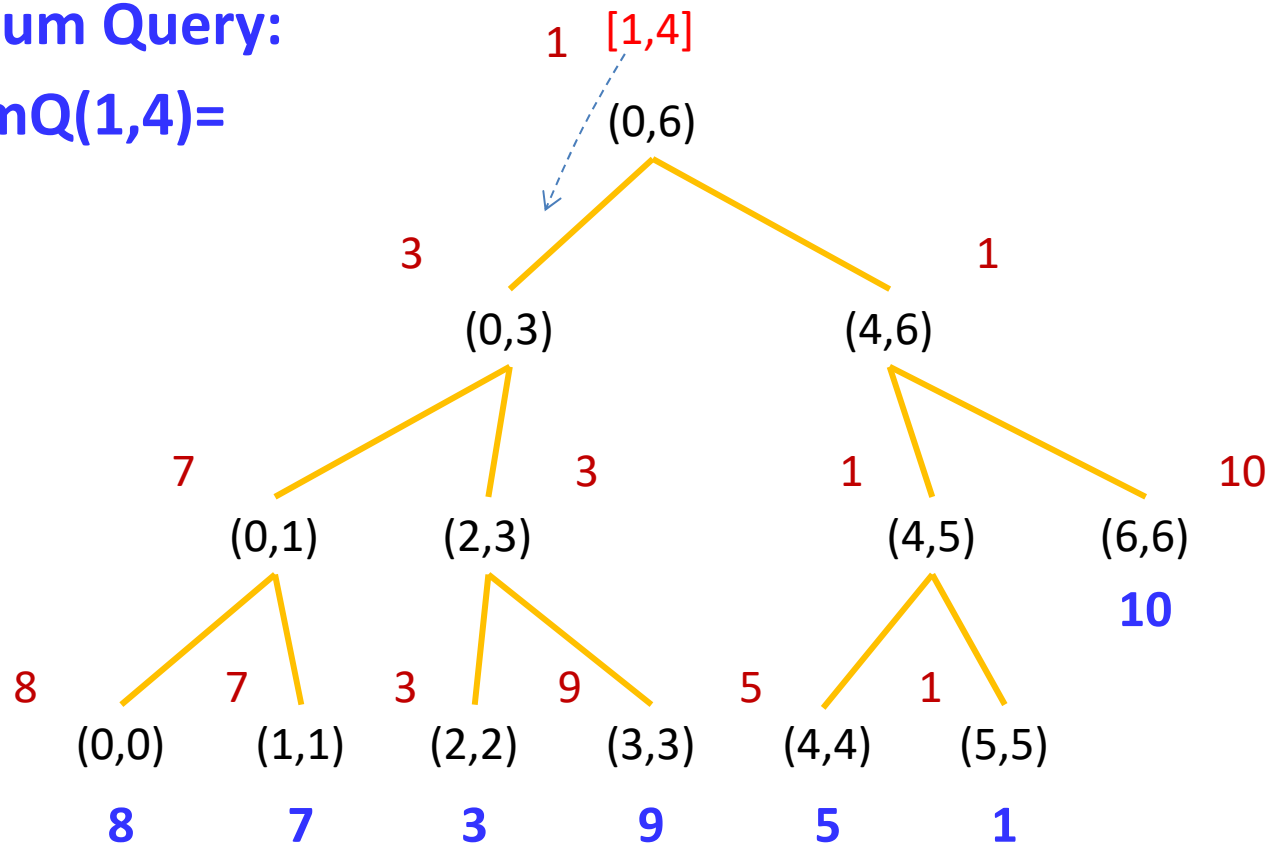


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

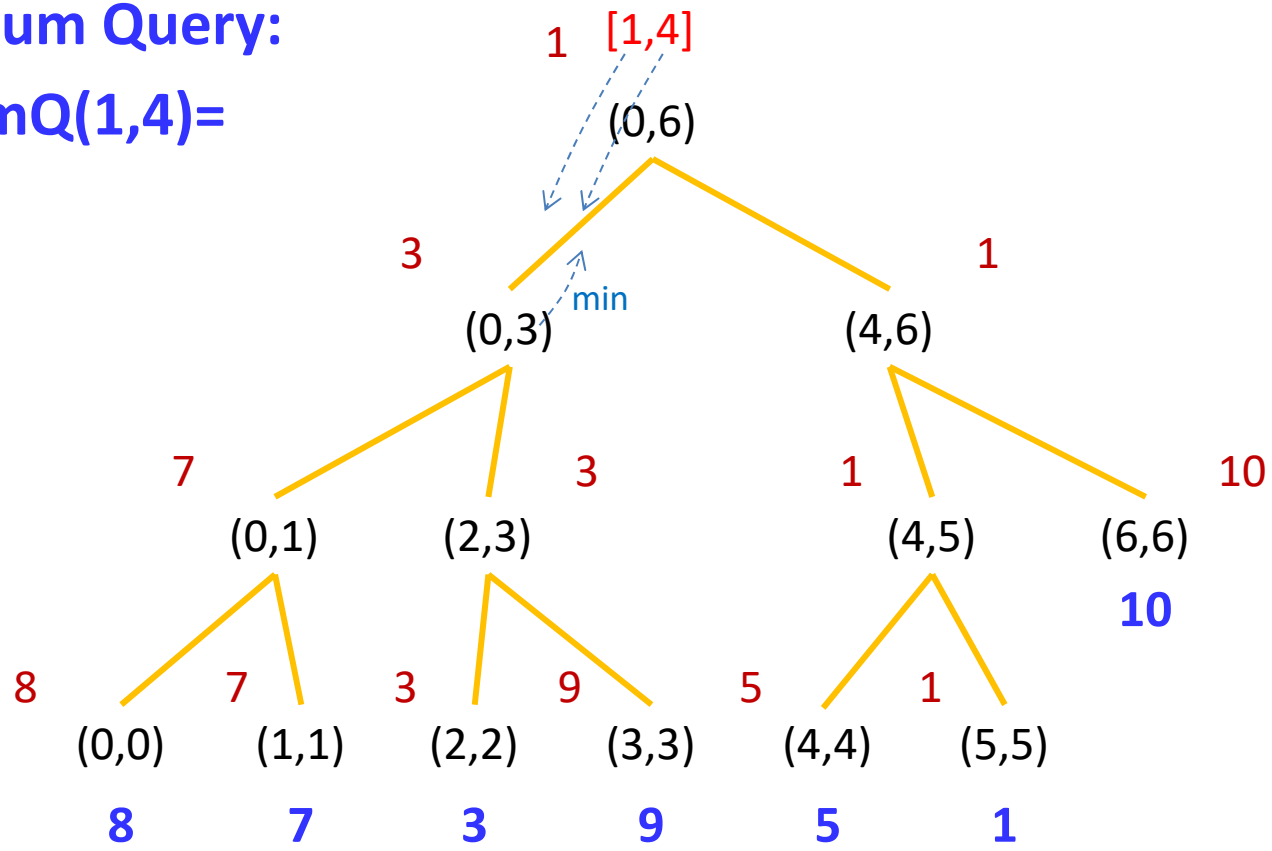


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

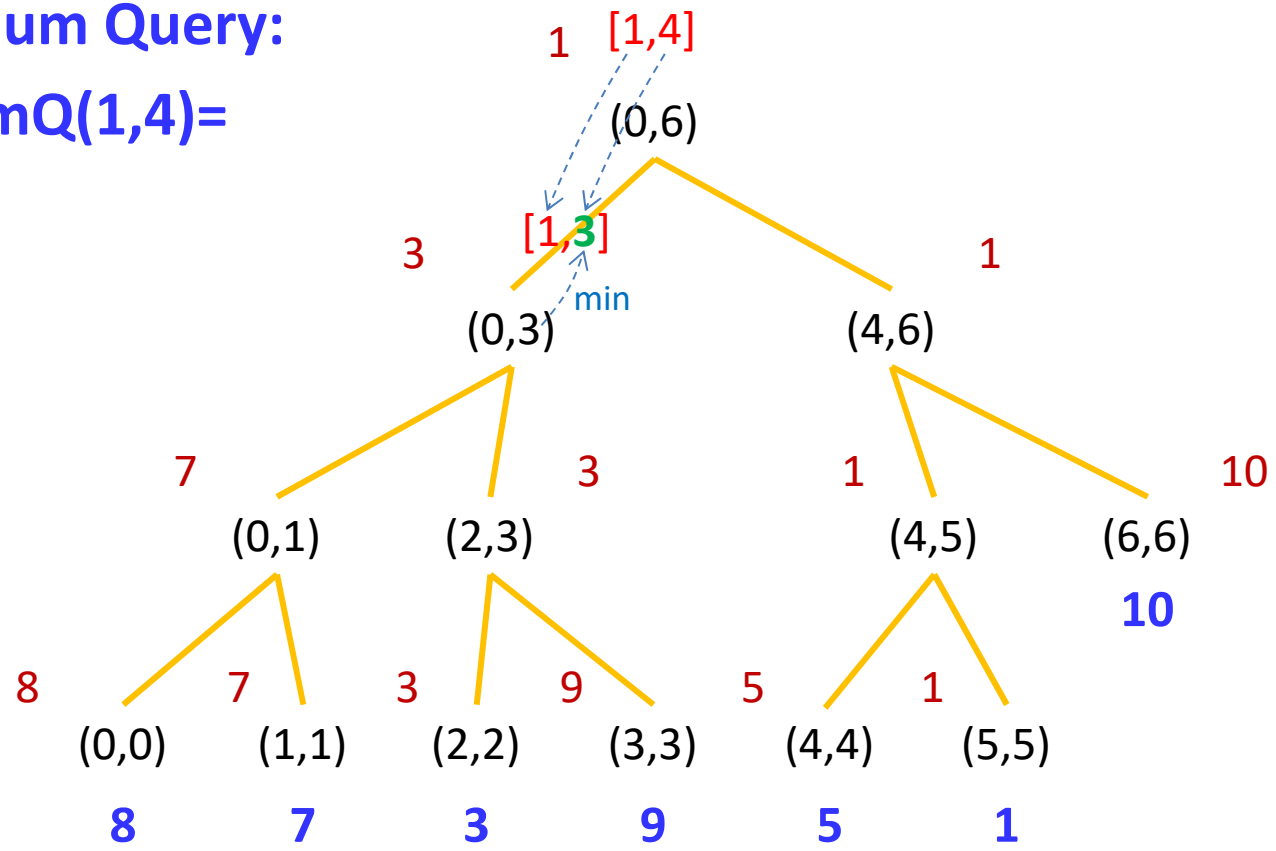


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

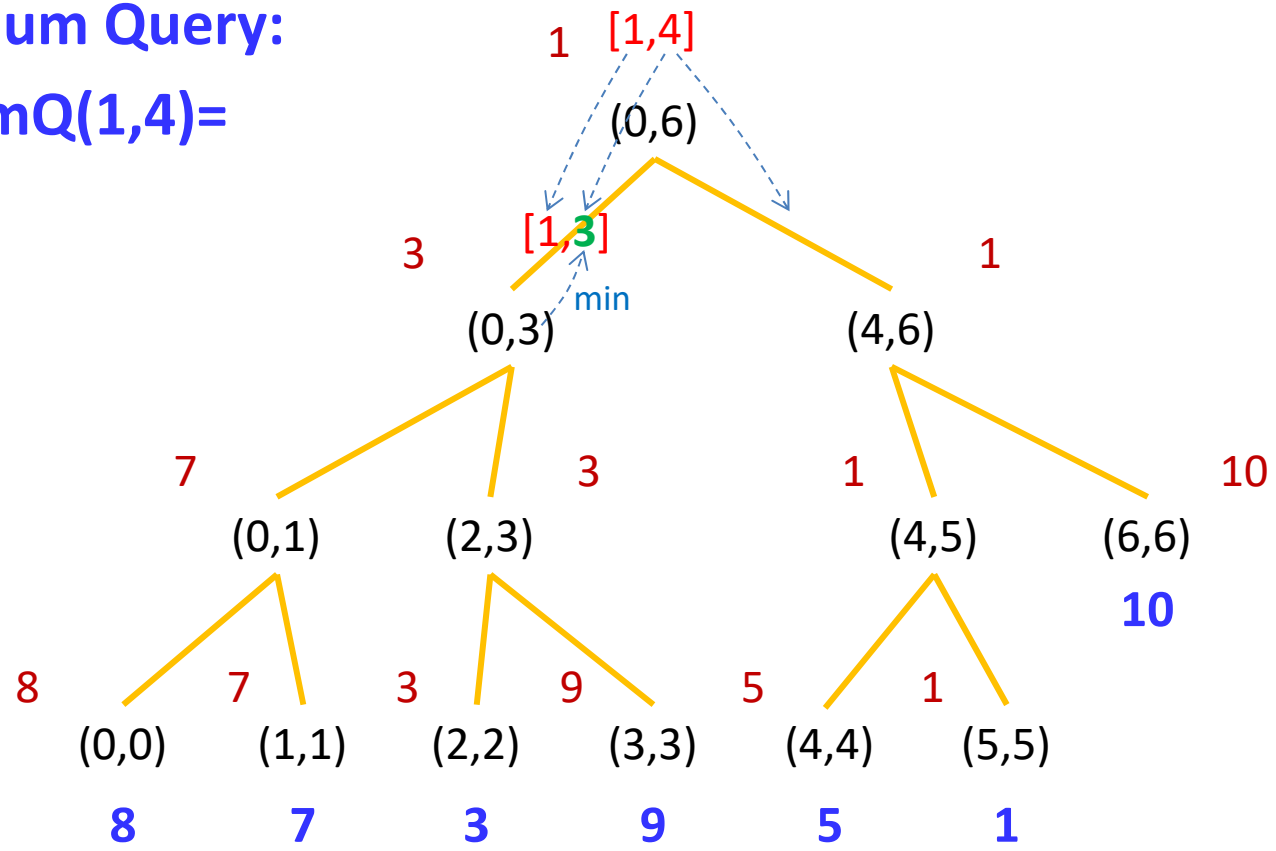


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

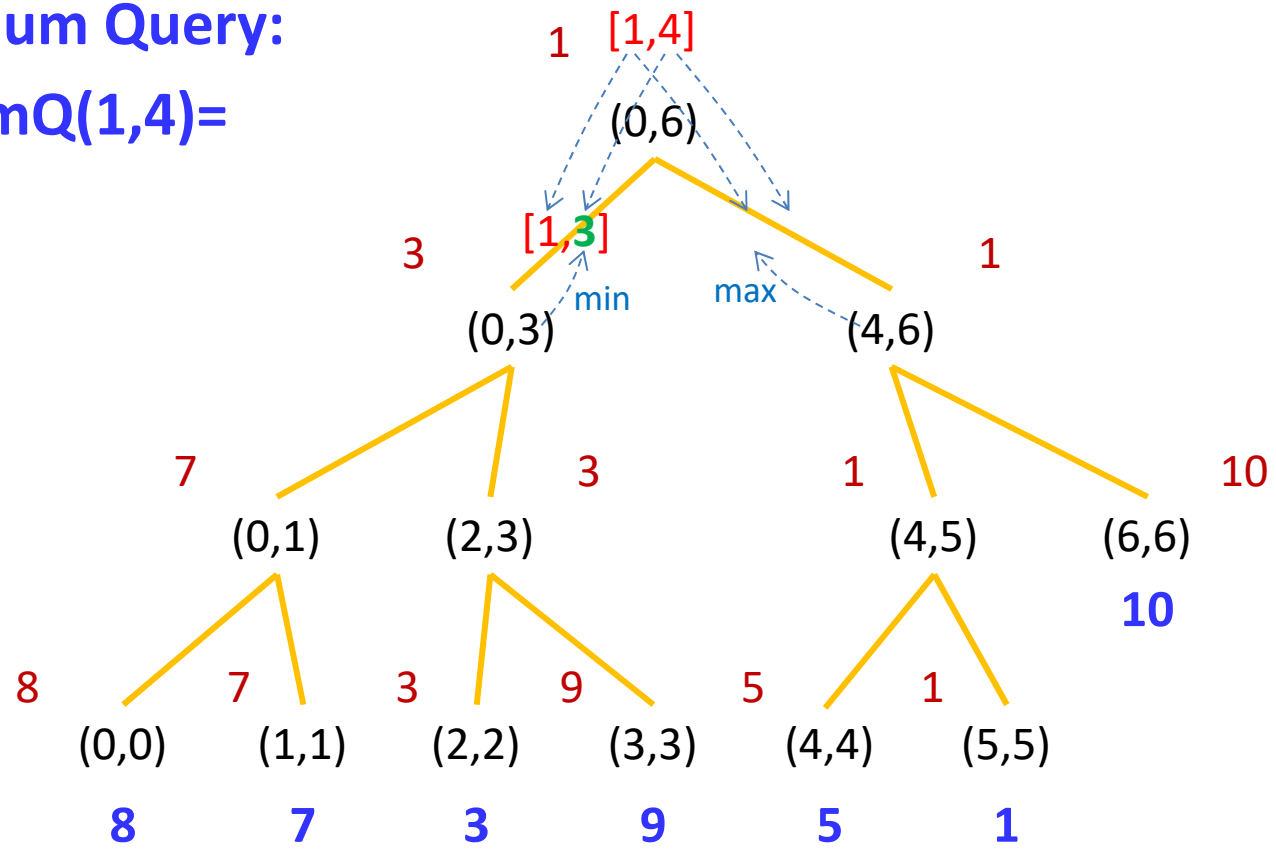


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

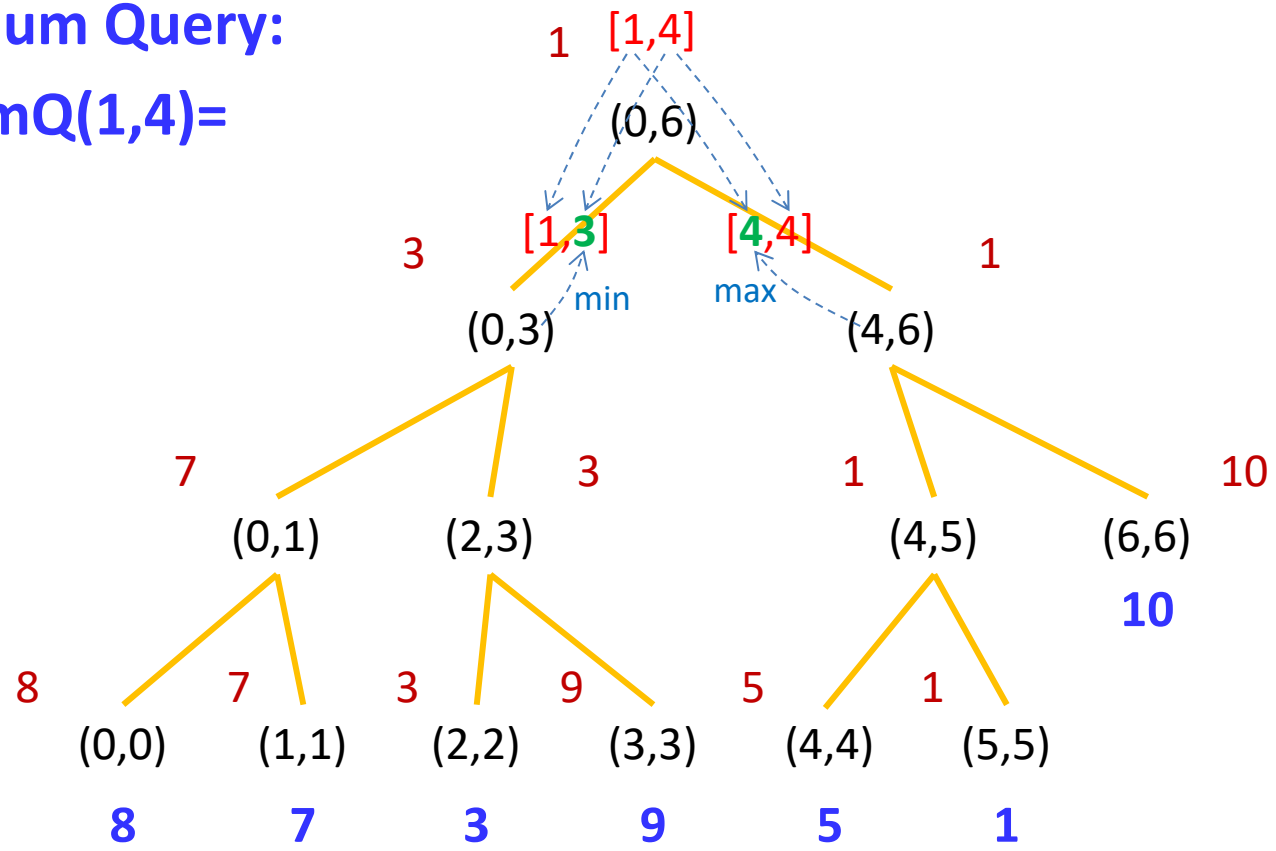


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

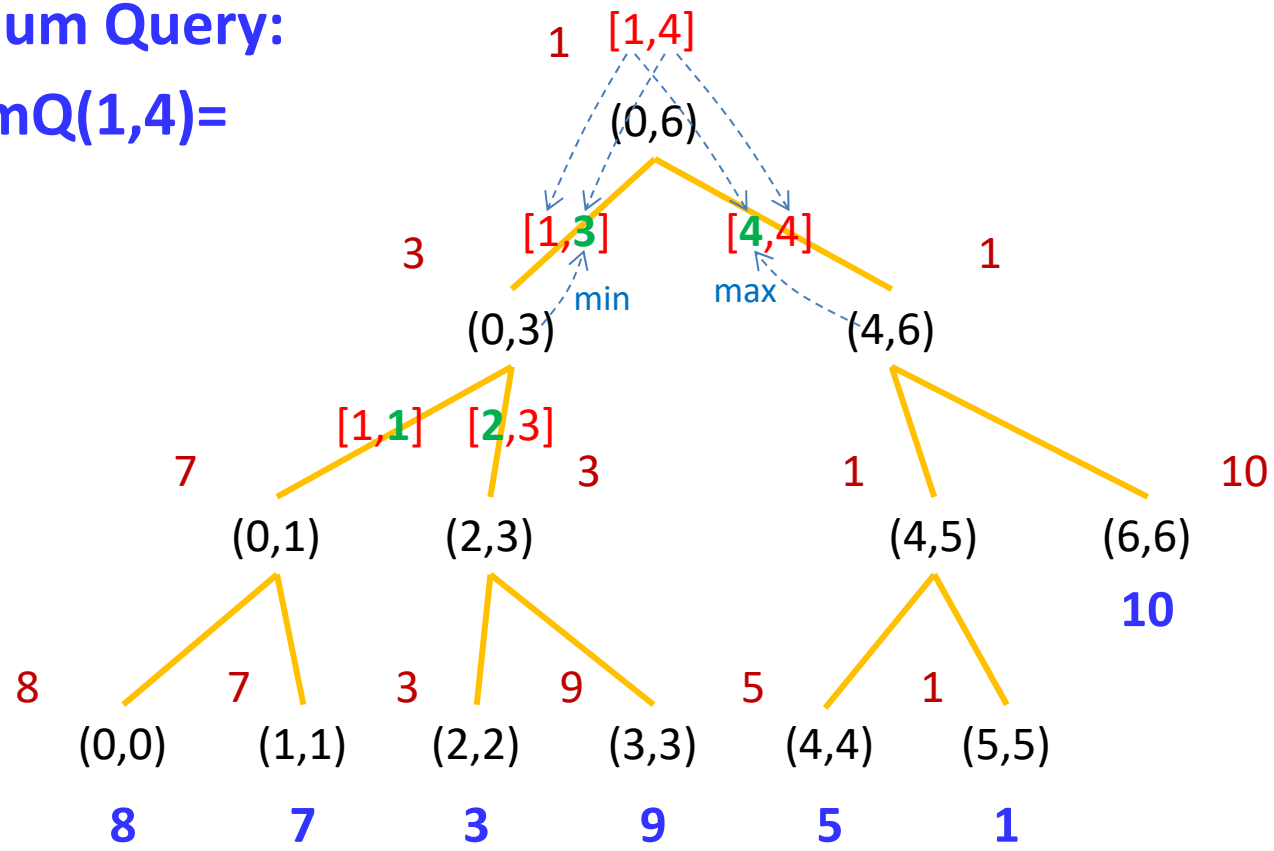


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

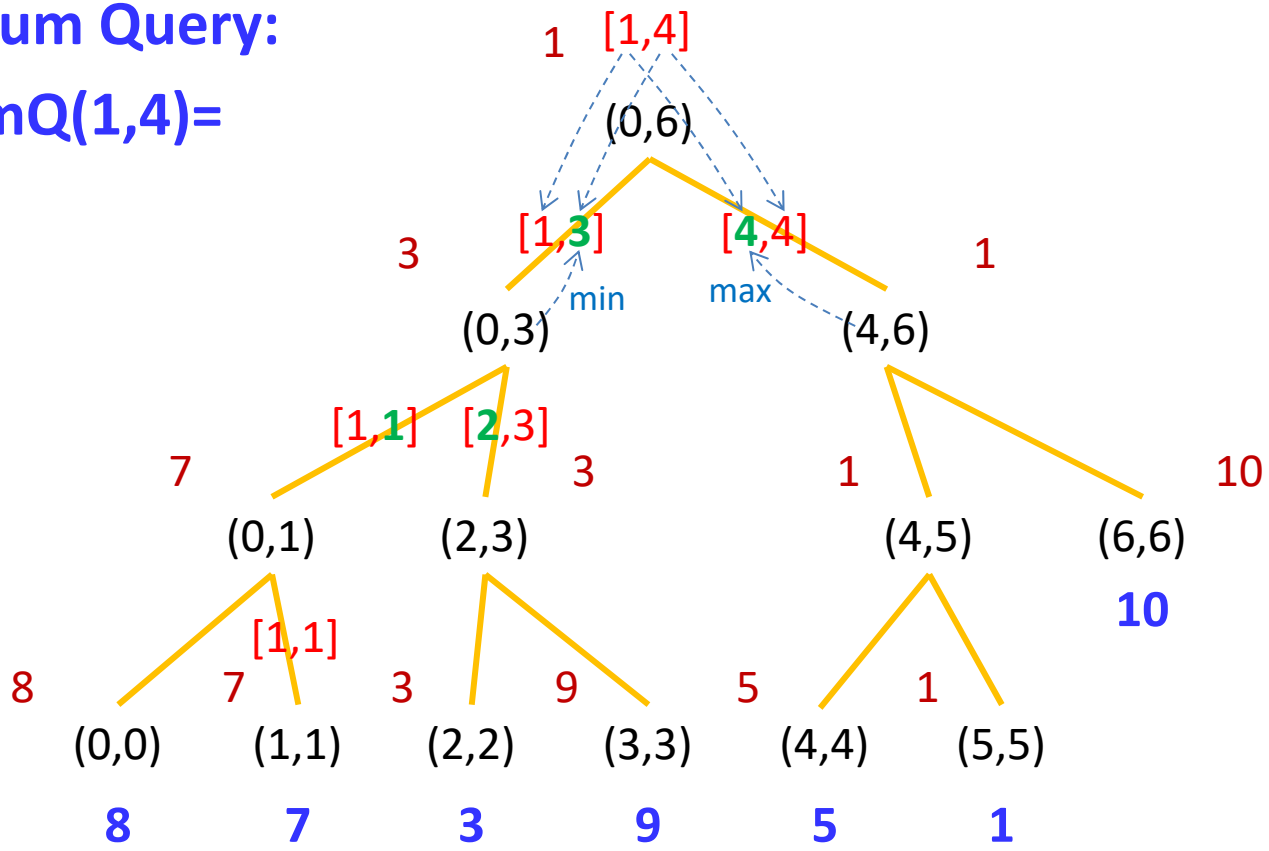


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

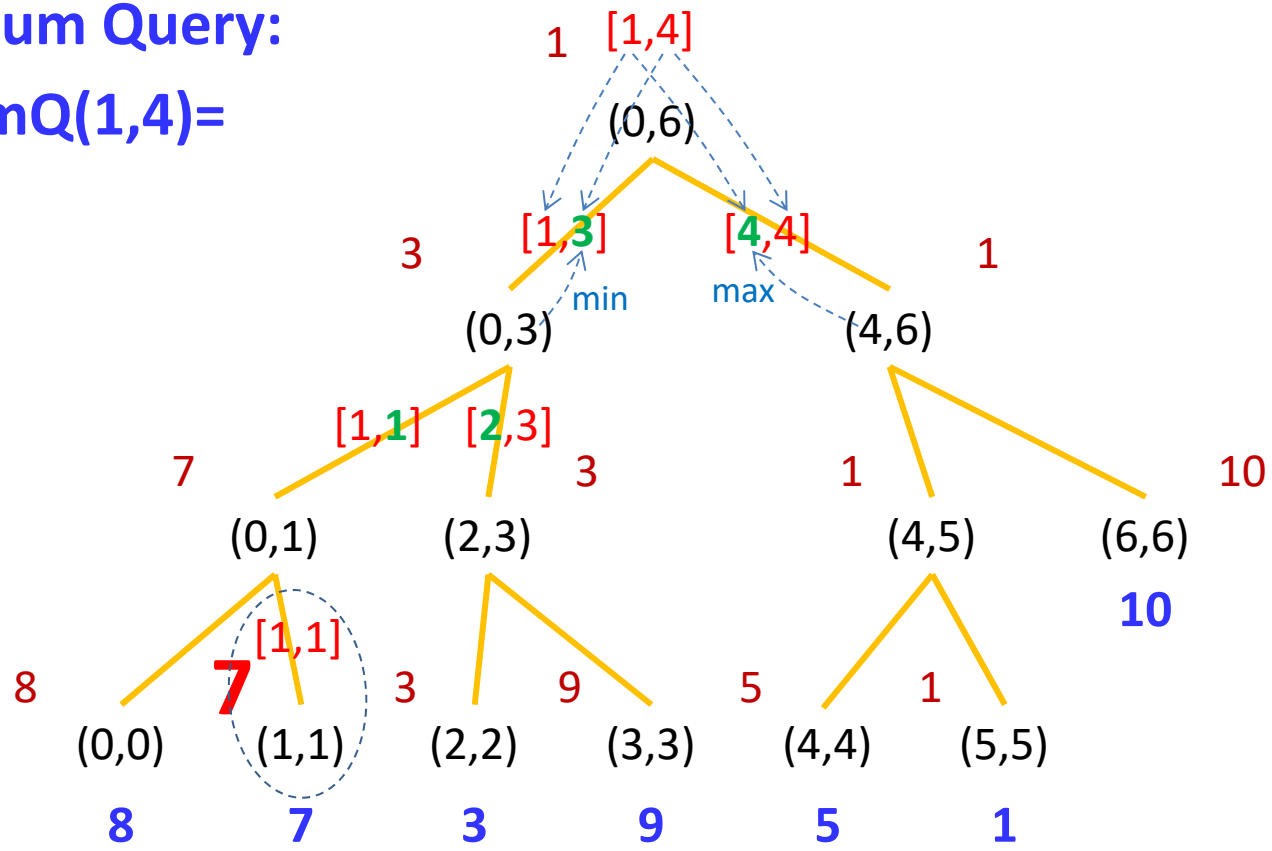


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

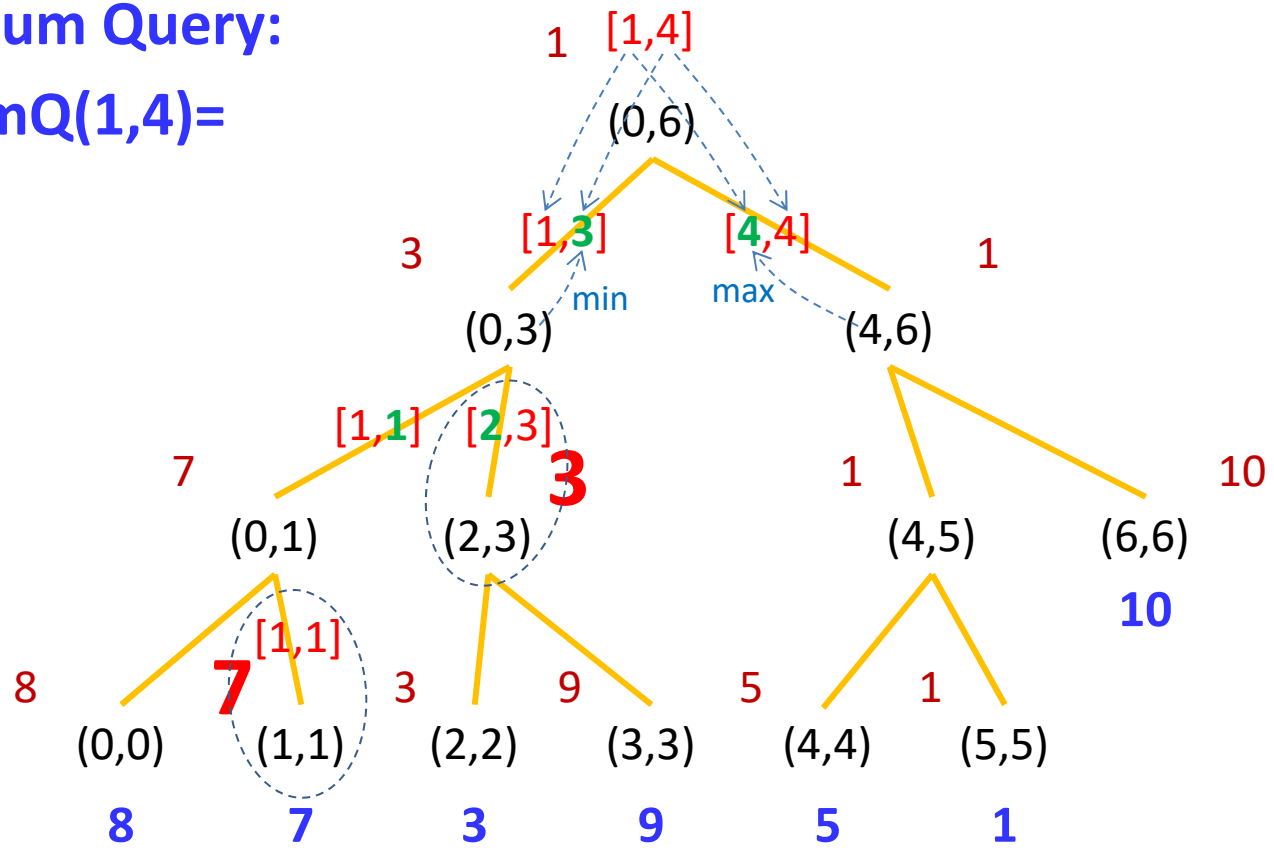


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

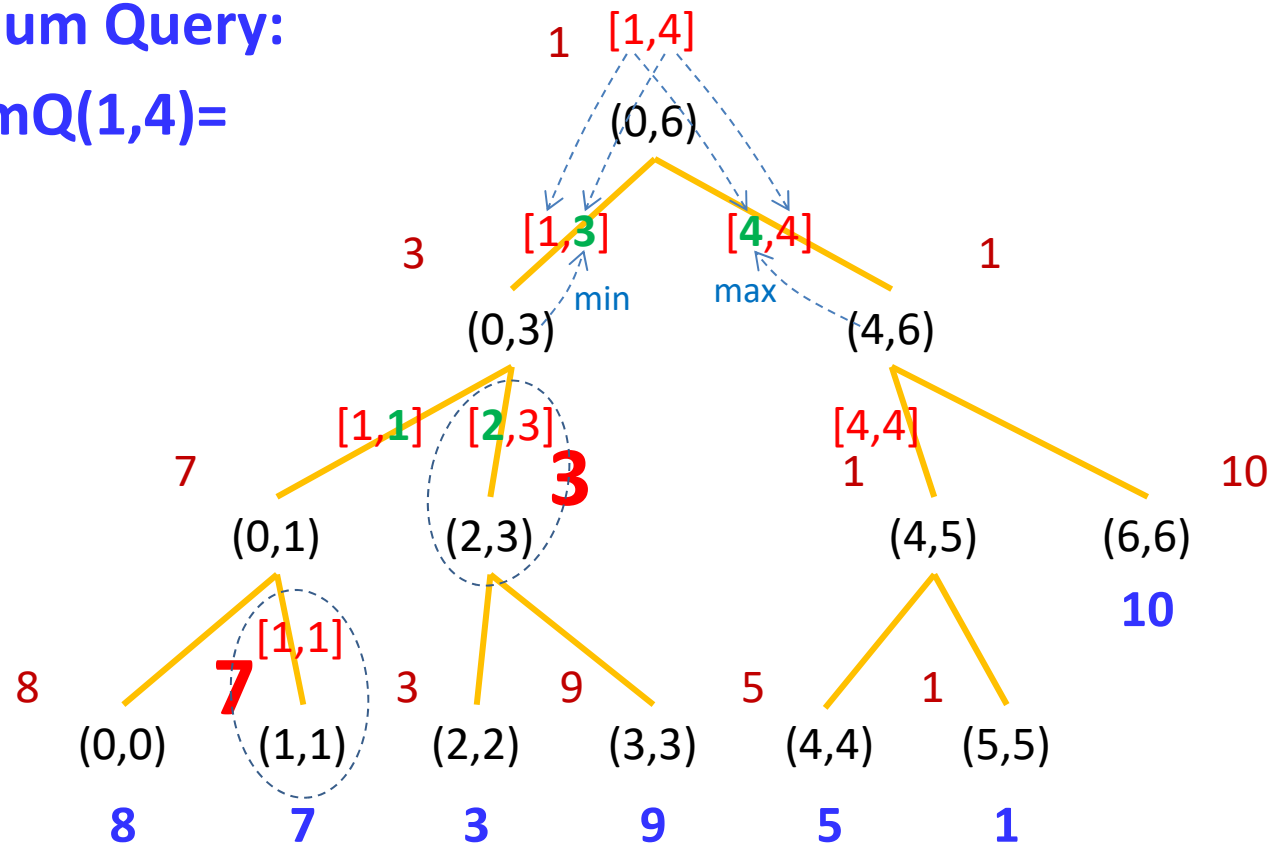


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

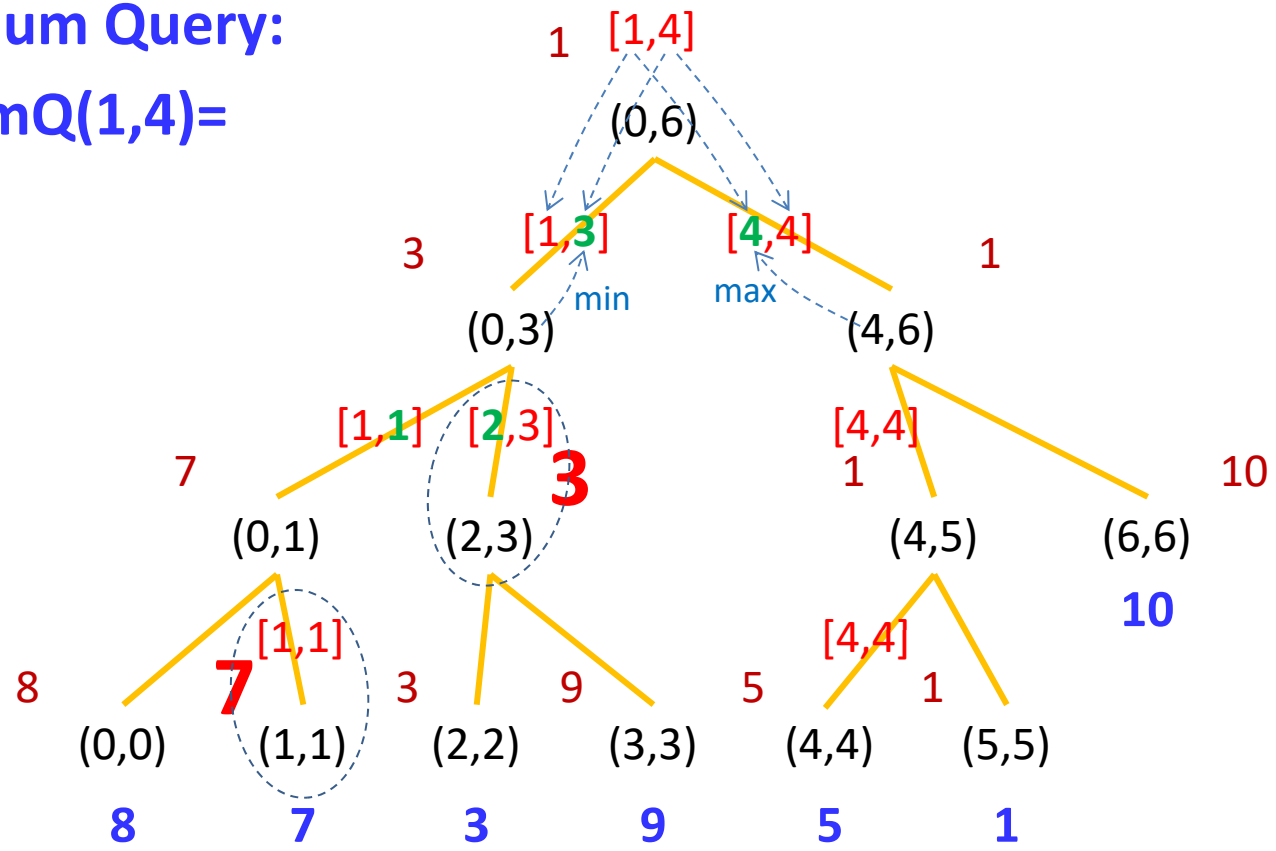


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

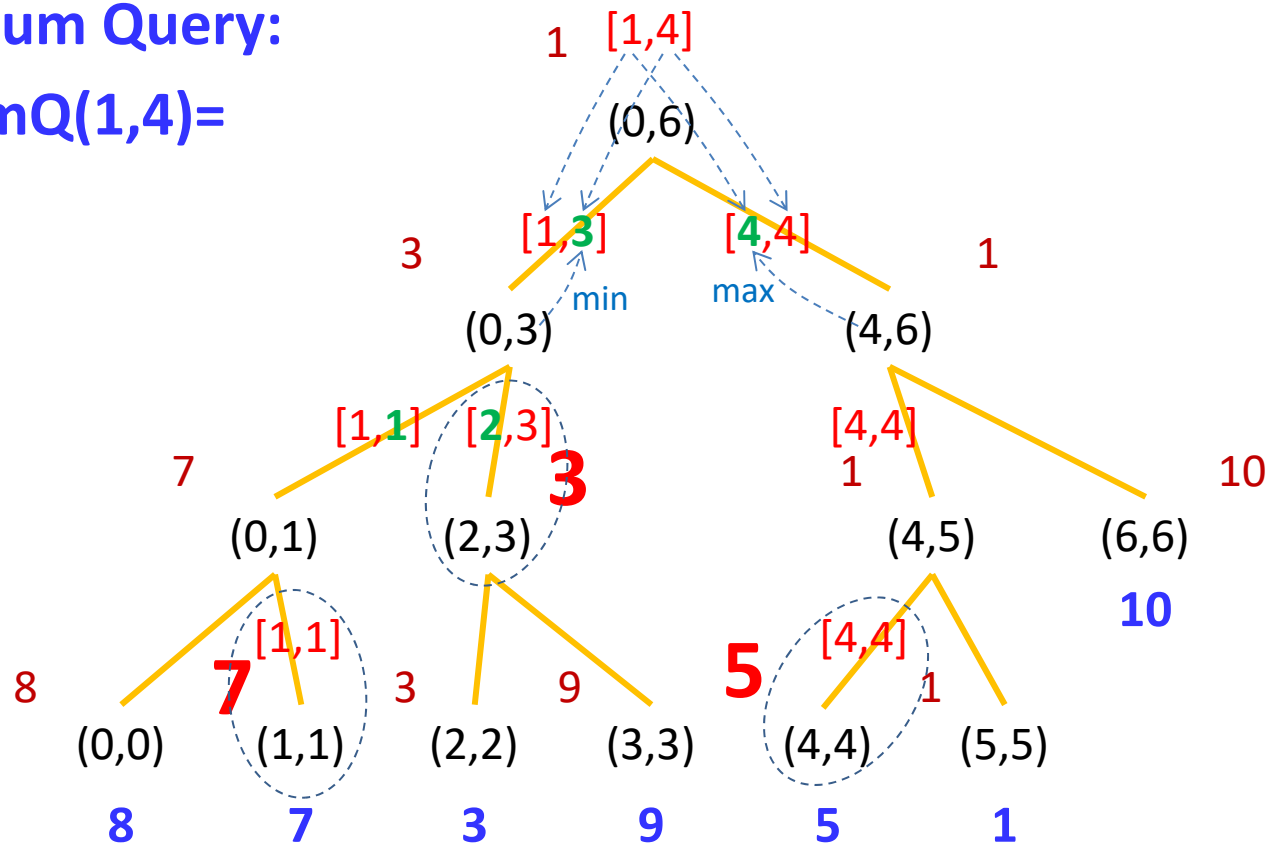


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)=

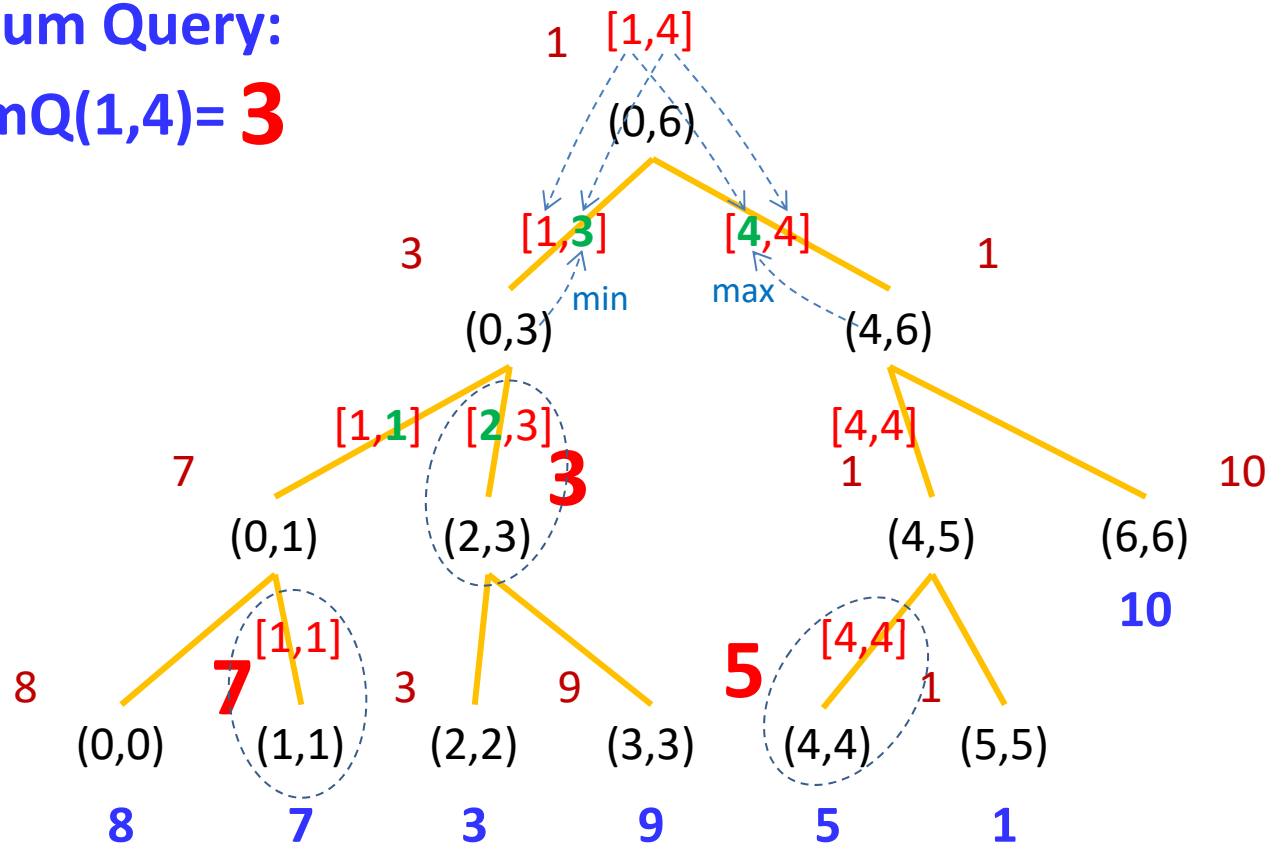


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(1,4)= 3

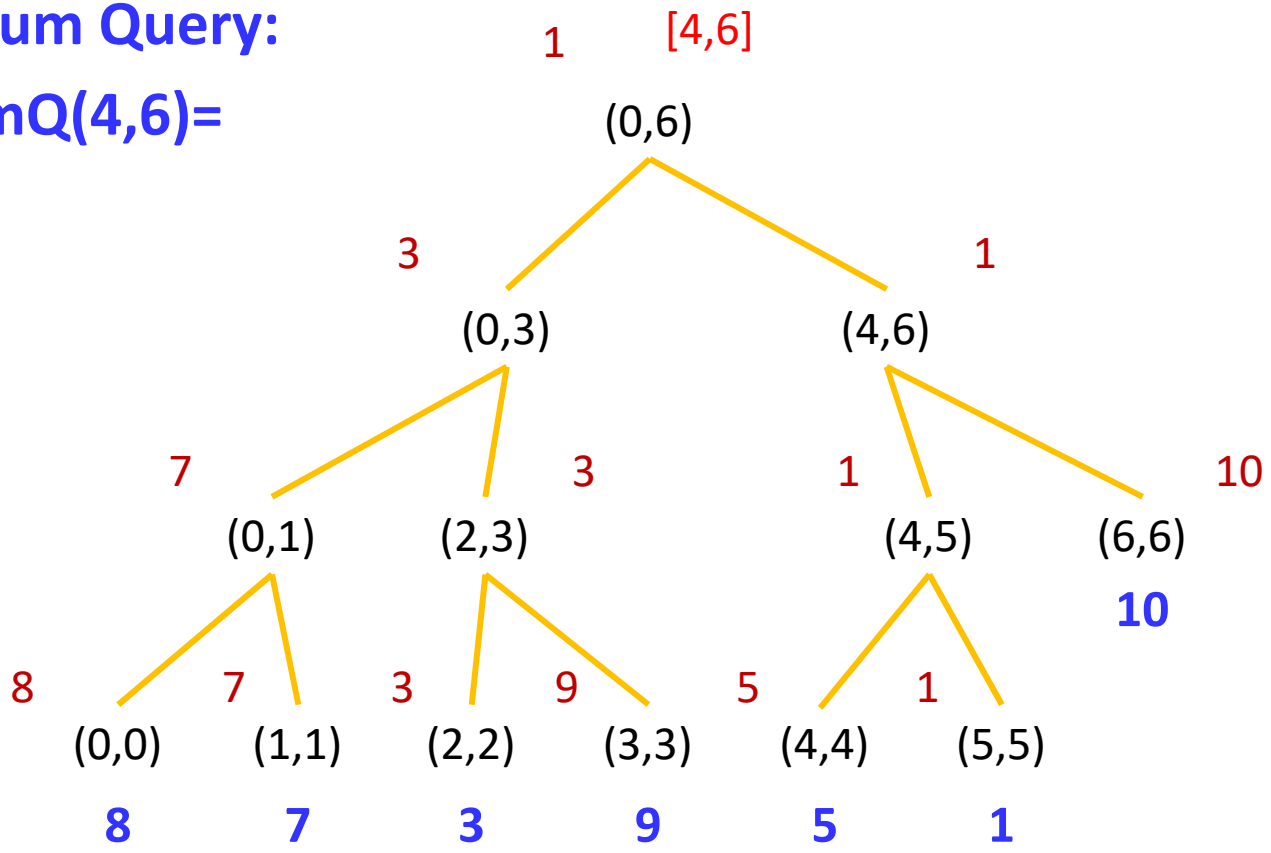


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(4,6)=

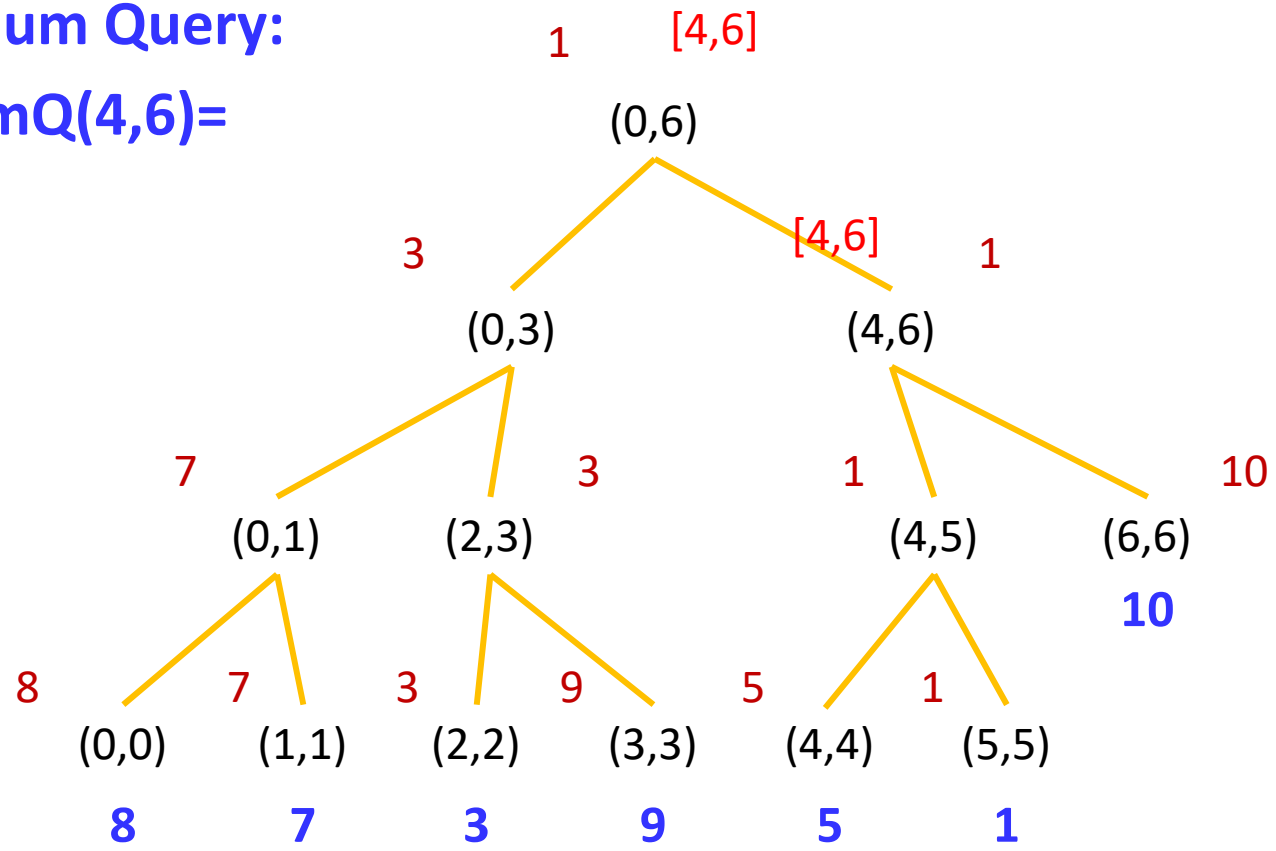


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(4,6)=

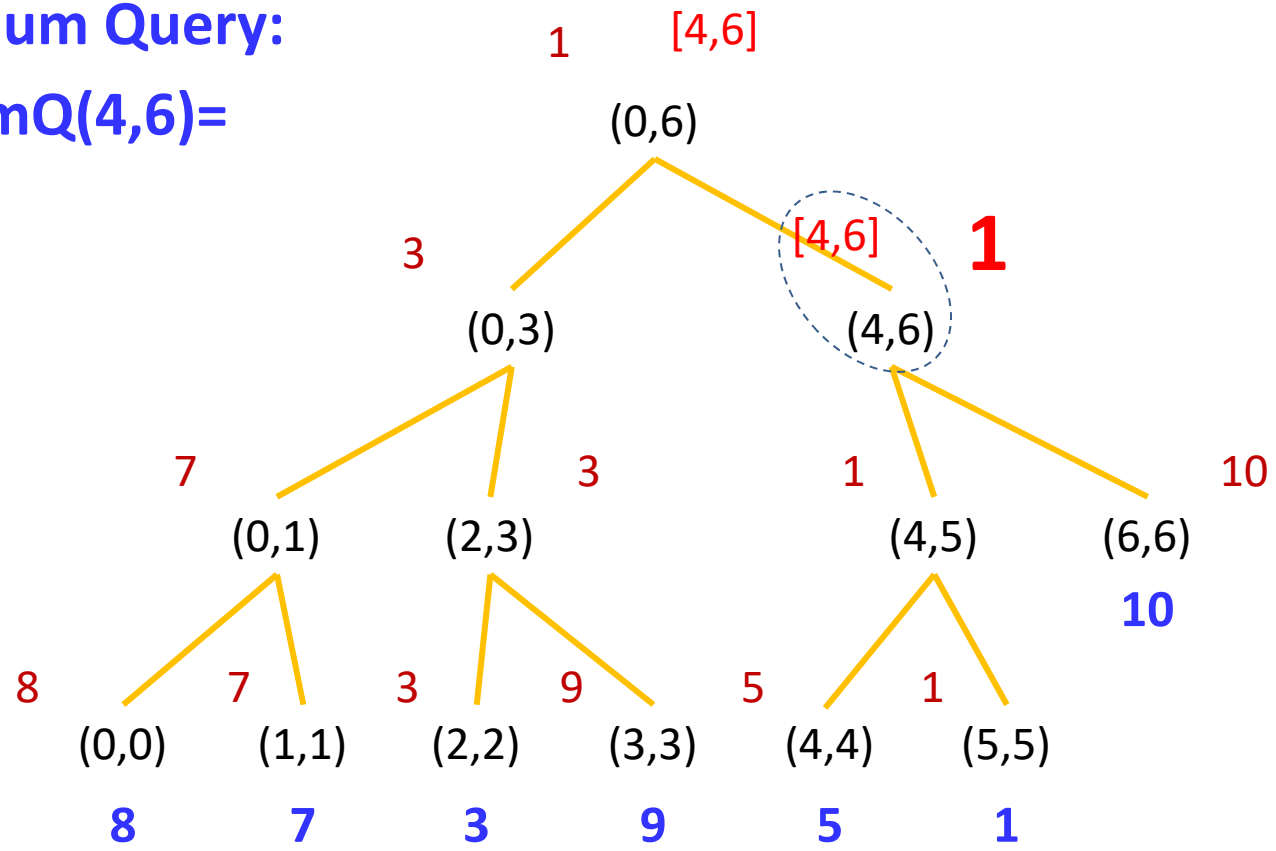


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(4,6)=

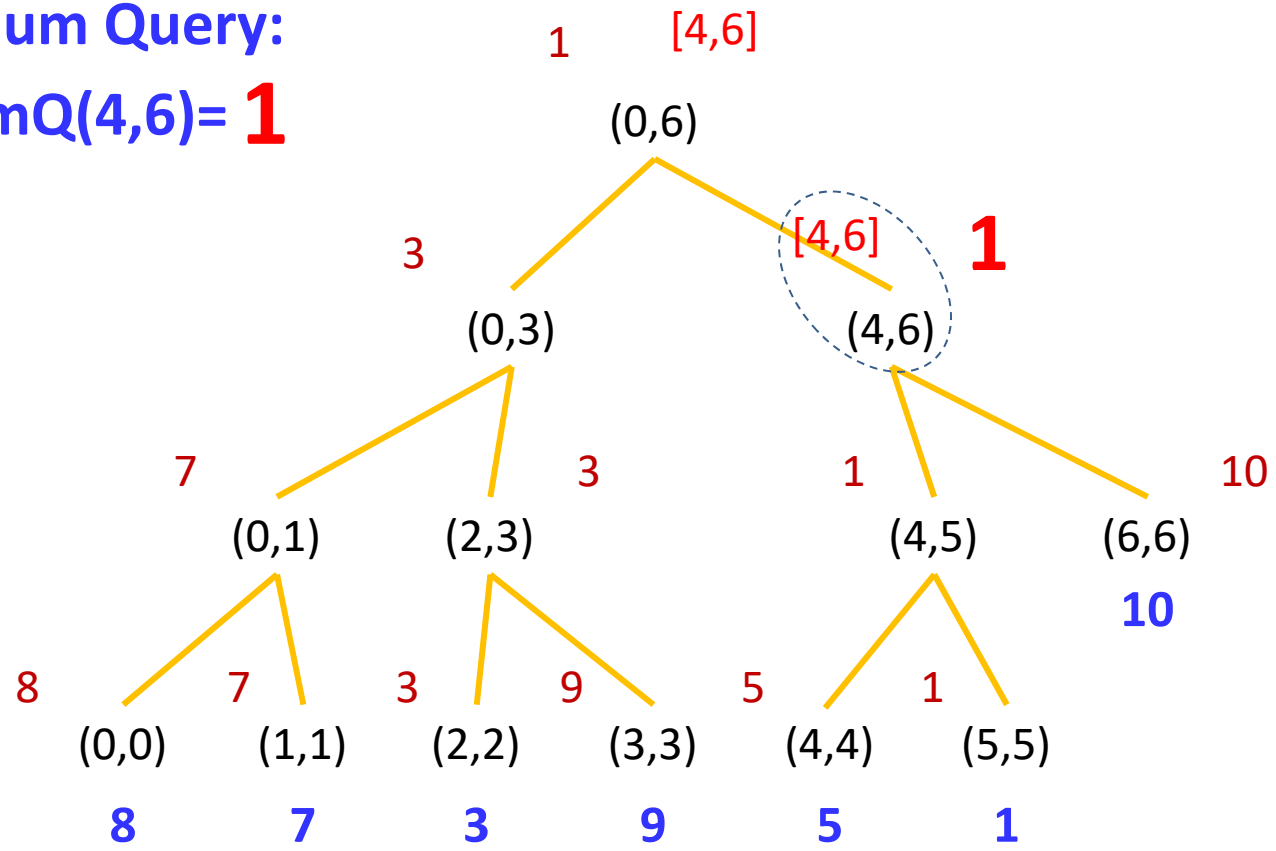


A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

範例: RmQ, RMQ, RSQ

Range minimum Query:

RmQ(4,6)= 1



A[i]	8	7	3	9	5	1	10
i	0	1	2	3	4	5	6

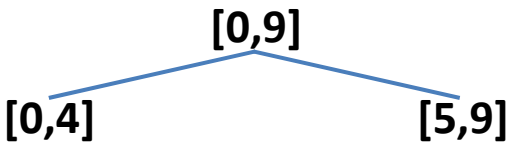
二元樹以陣列存放需要的空間

二元樹以陣列存放需要的空間

[0,9]

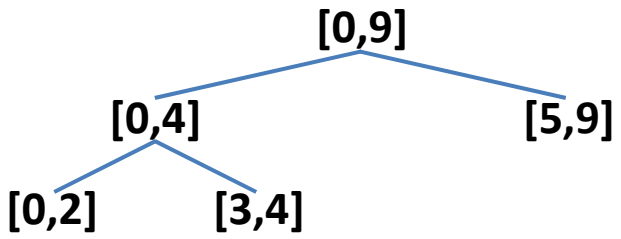
- 二元線段樹

二元樹以陣列存放需要的空間



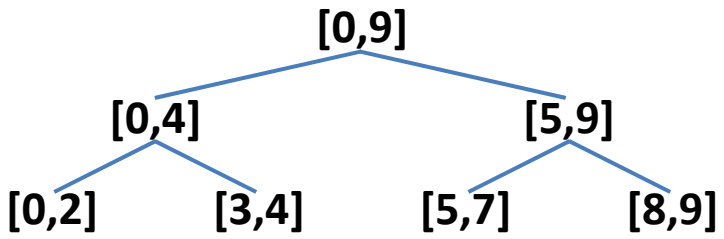
- 二元線段樹

二元樹以陣列存放需要的空間



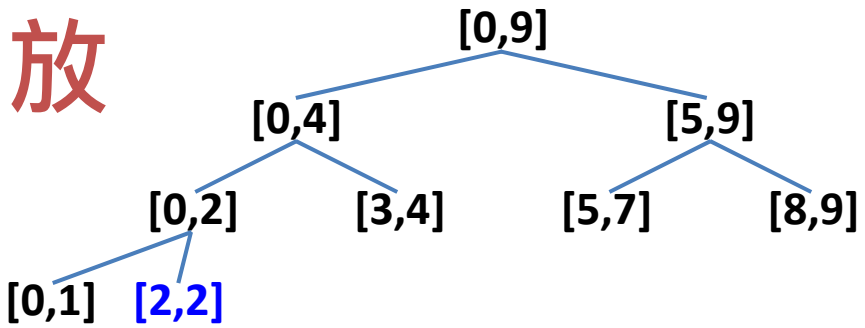
- 二元線段樹

二元樹以陣列存放需要的空間



- 二元線段樹

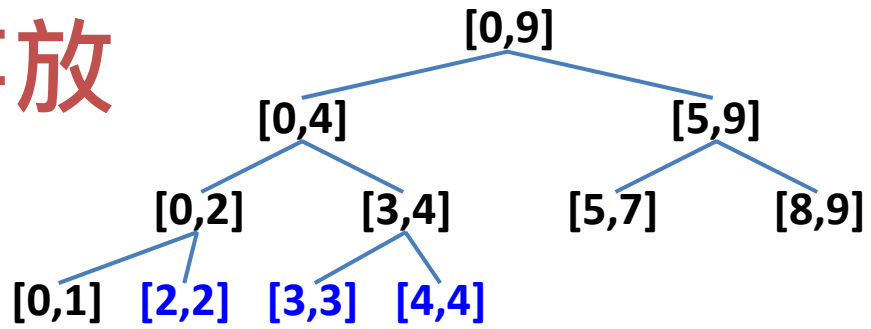
二元樹以陣列存放需要的空間



- 二元線段樹

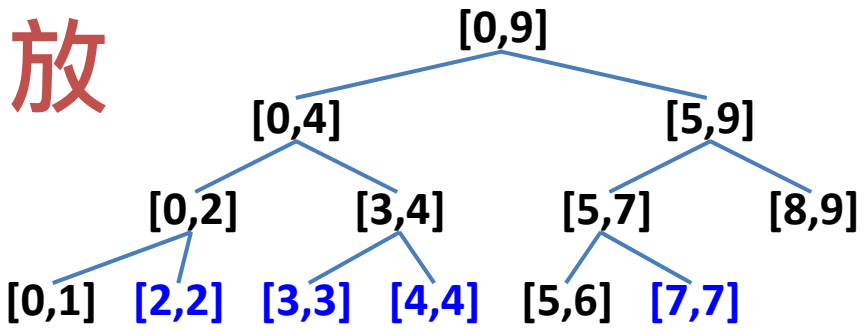
二元樹以陣列存放需要的空間

- 二元線段樹



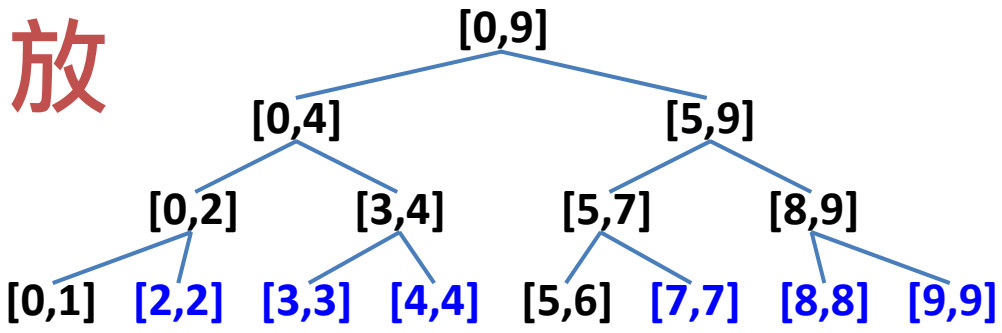
二元樹以陣列存放需要的空間

- 二元線段樹



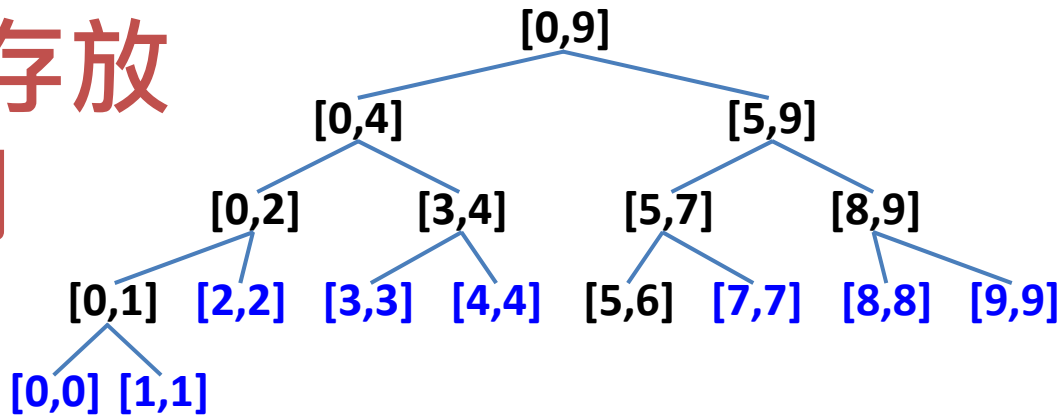
二元樹以陣列存放需要的空間

- 二元線段樹



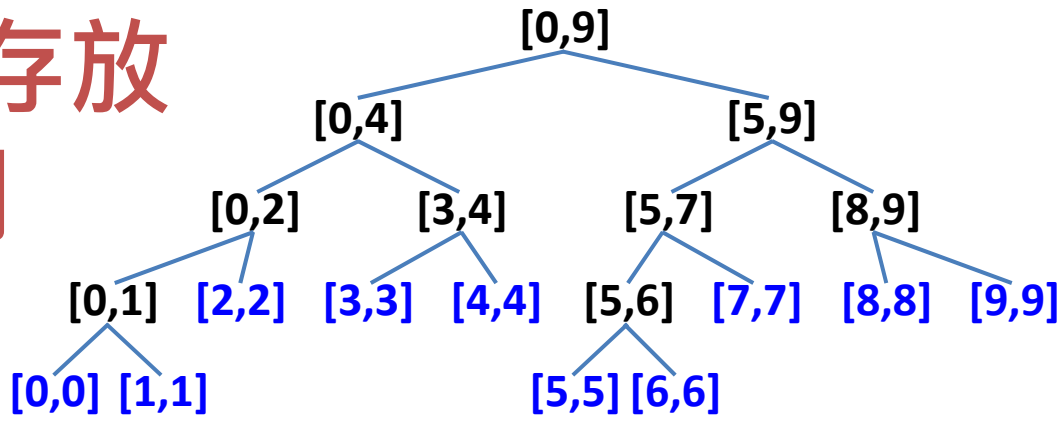
二元樹以陣列存放需要的空間

- 二元線段樹

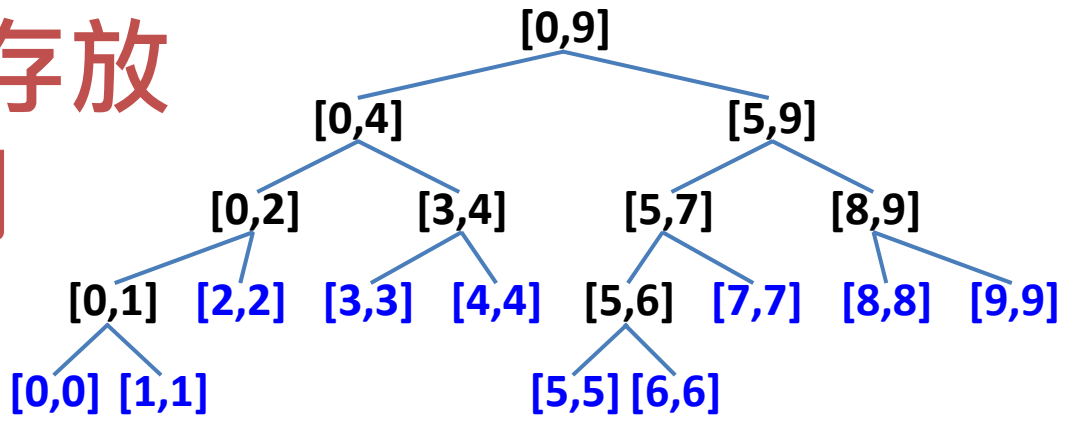


二元樹以陣列存放需要的空間

- 二元線段樹



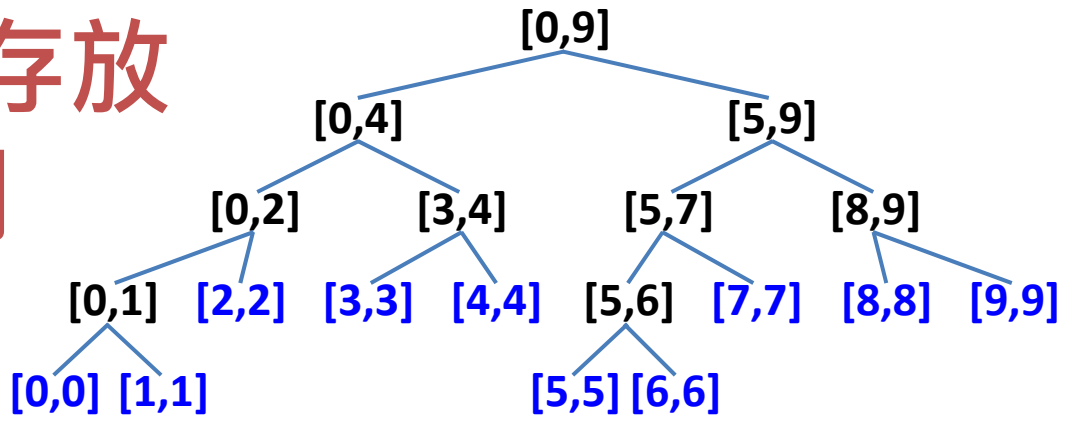
二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

二元樹以陣列存放需要的空間

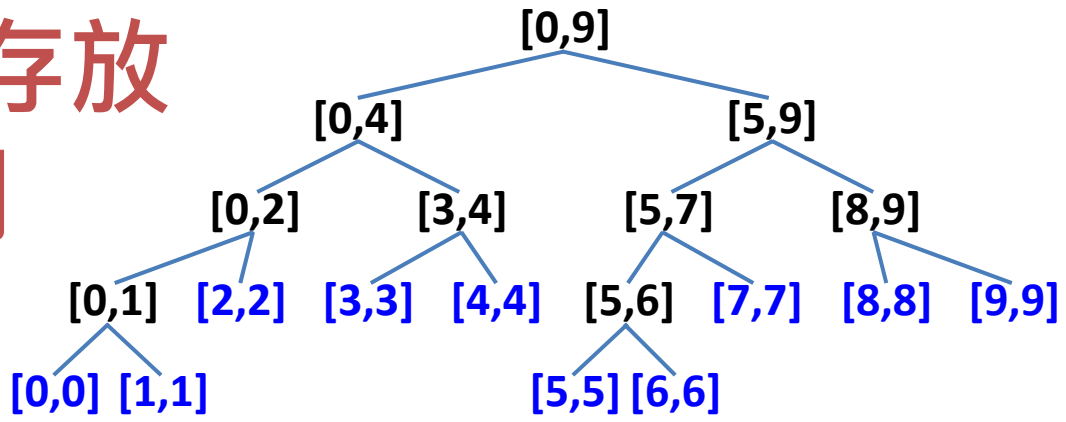


- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

  n = 10

二元樹以陣列存放需要的空間

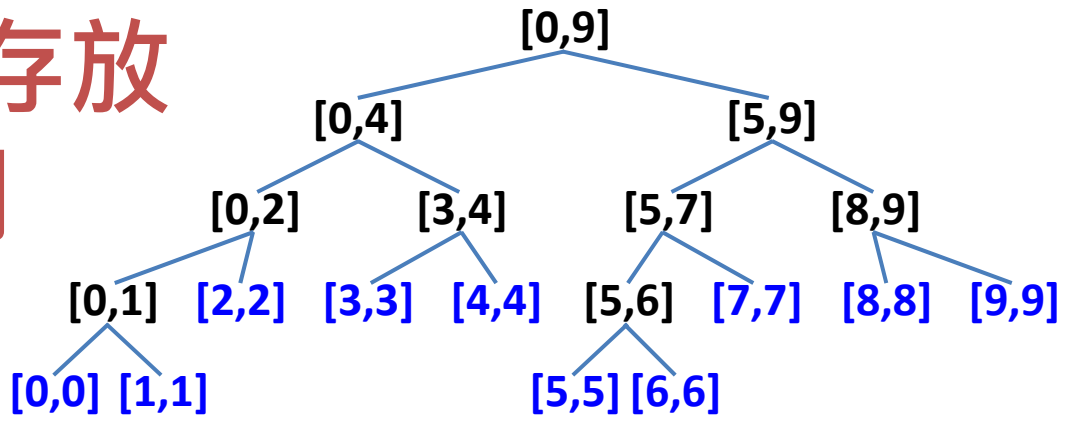


- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

  n = 10 ⇒ $2^3 < n \leq 2^4$

二元樹以陣列存放需要的空間

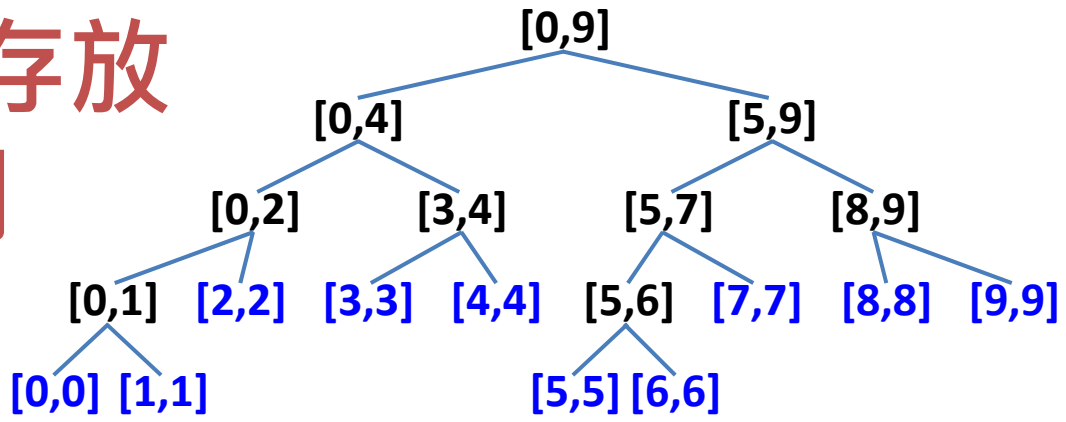


- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

  n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

二元樹以陣列存放需要的空間



- 二元線段樹

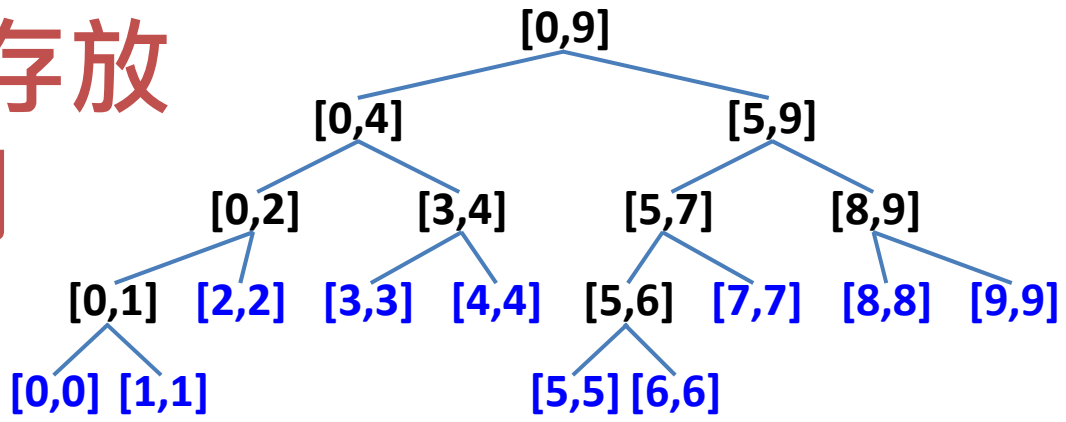
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

  3n=30 不夠大

  n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

二元樹以陣列存放需要的空間

- 二元線段樹

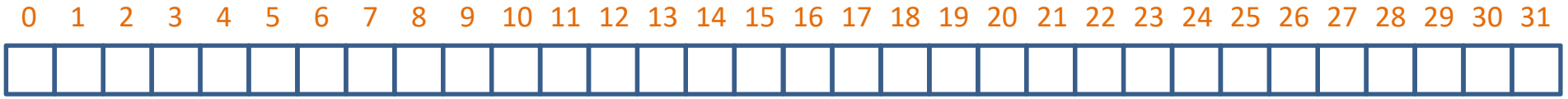


- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

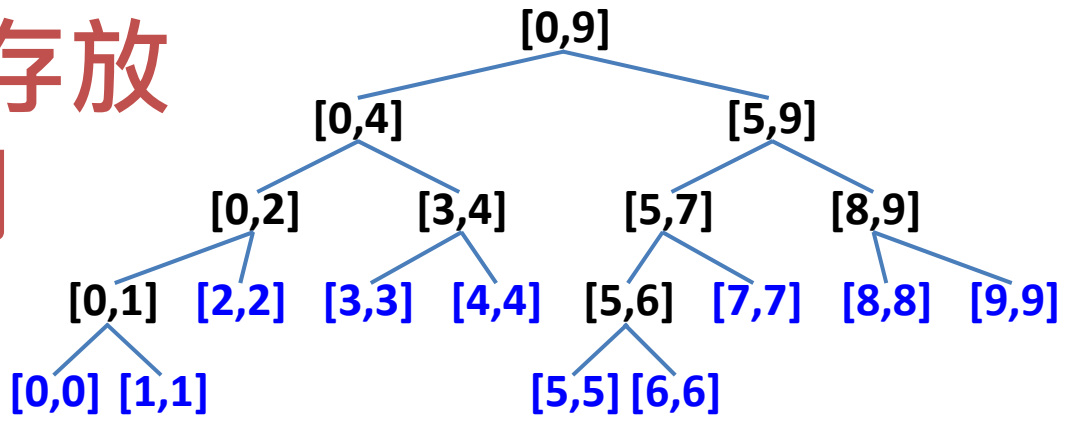
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間

- 二元線段樹

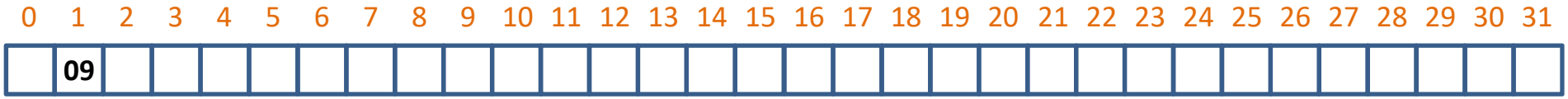


- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

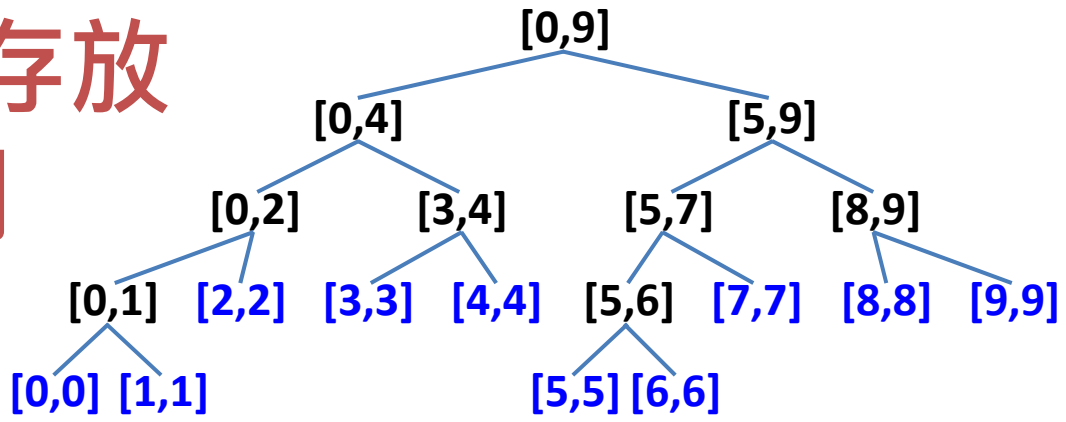
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間



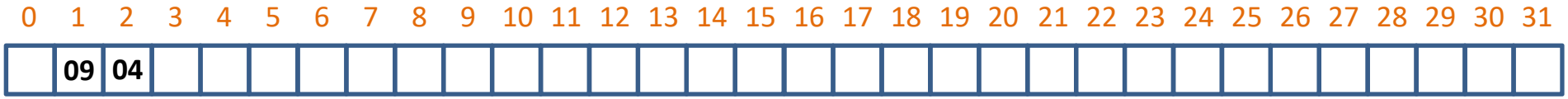
- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

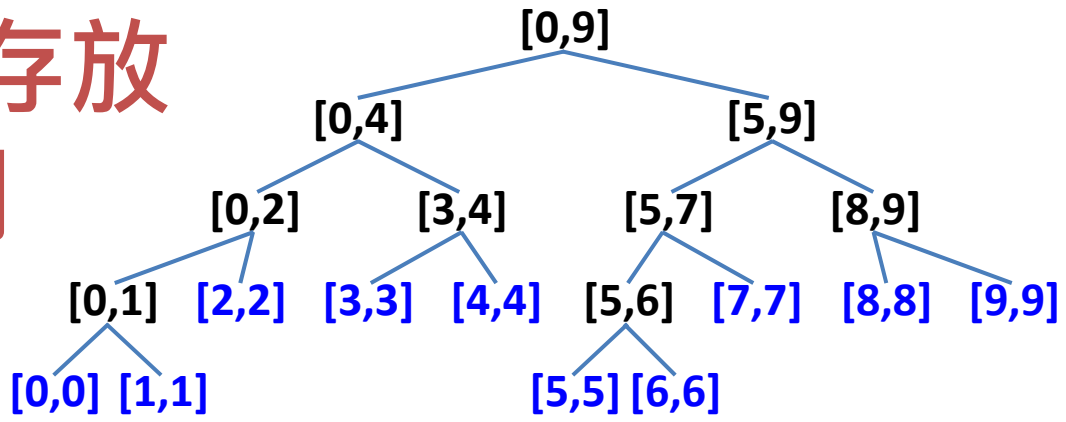
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間



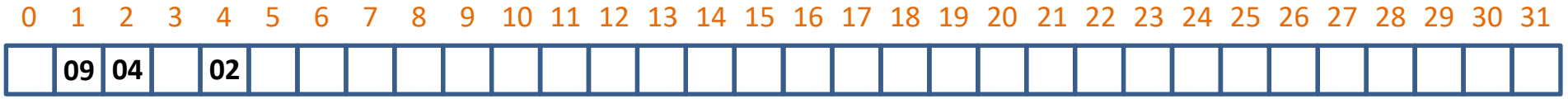
- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

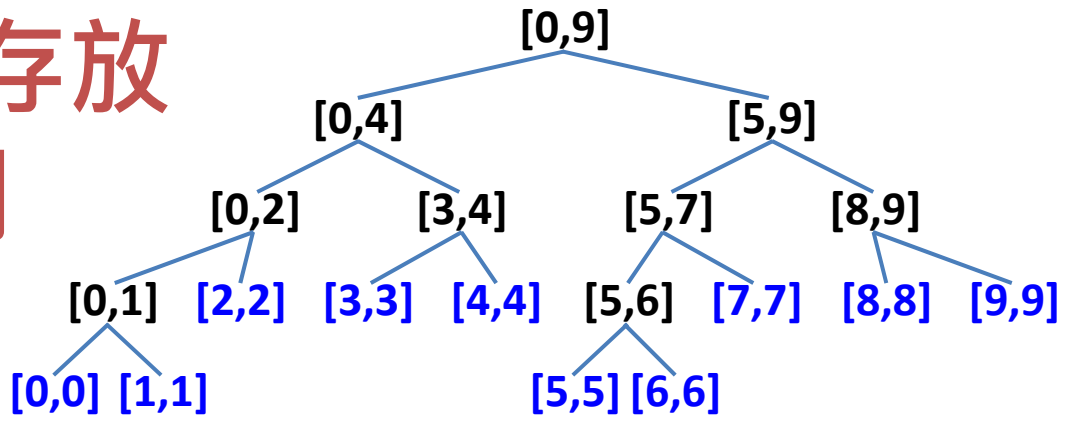
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間



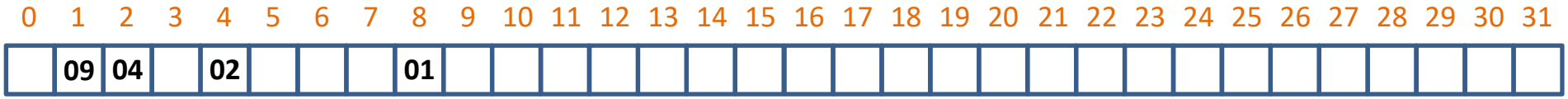
- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

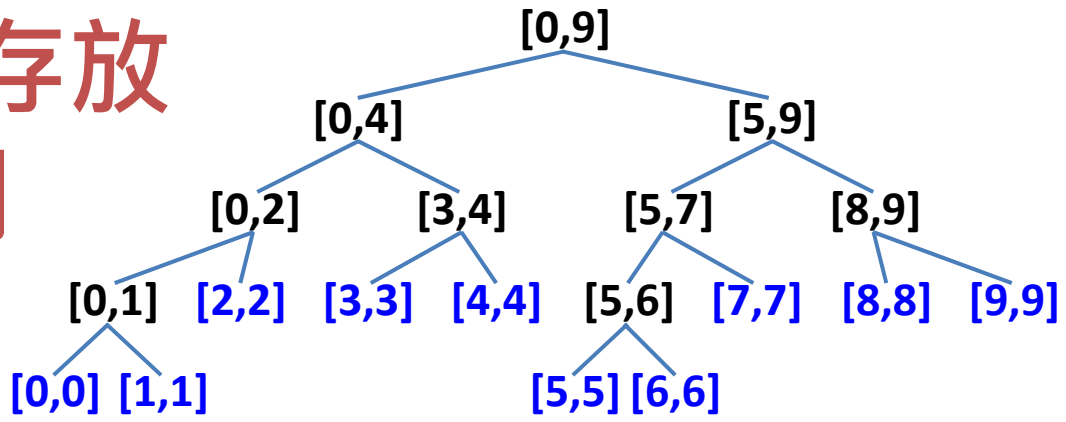
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間



- 二元線段樹

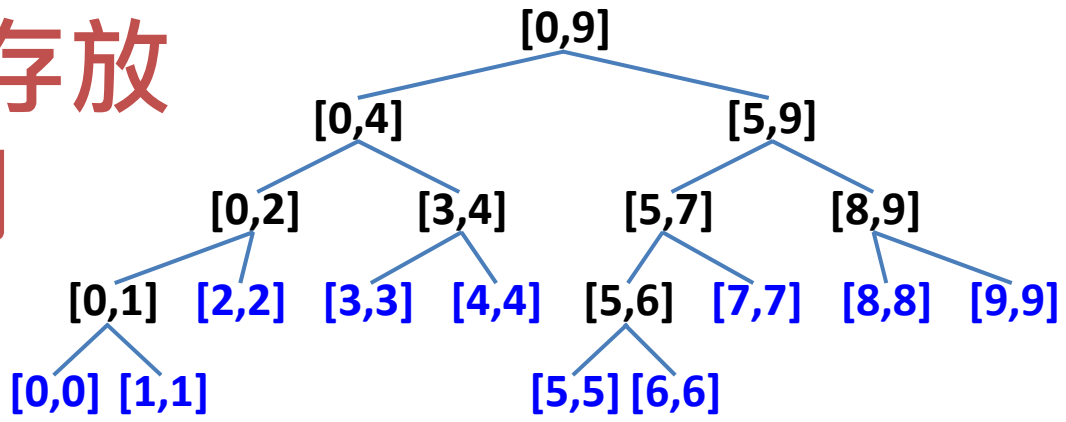
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

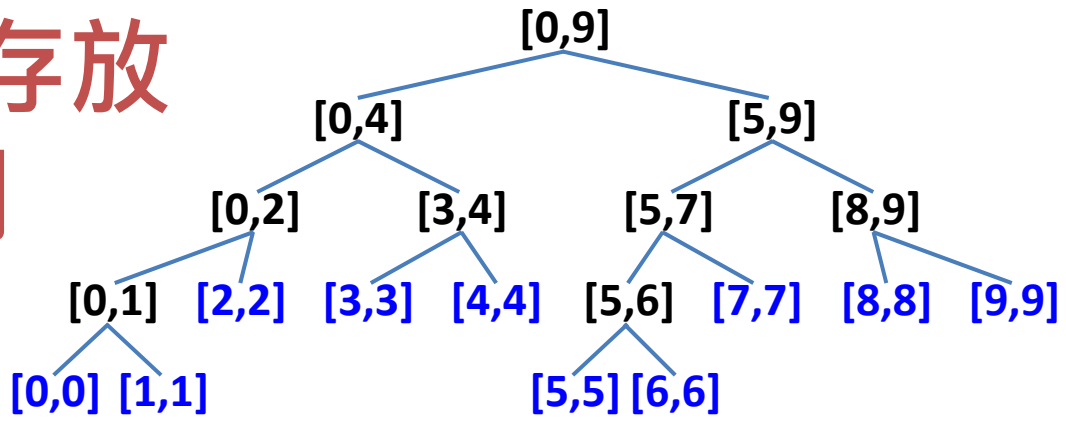
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04		02				01								00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

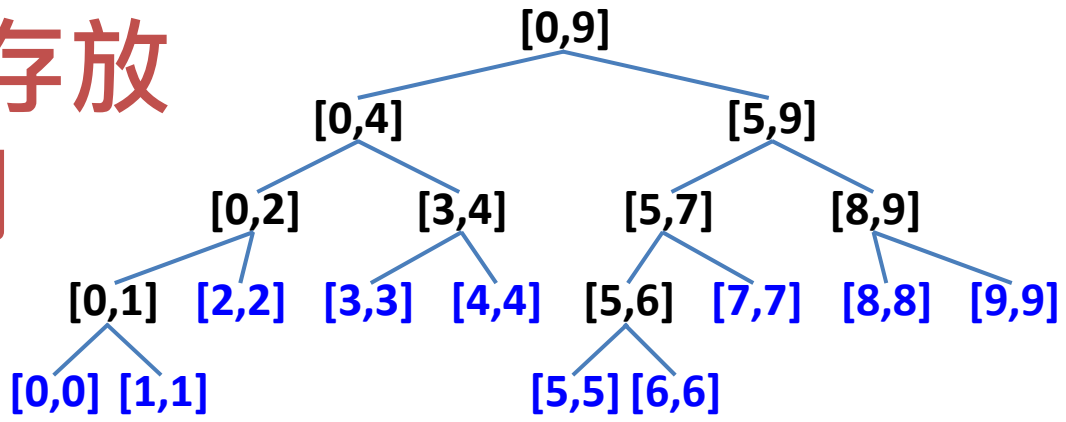
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04		02				01	22							00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

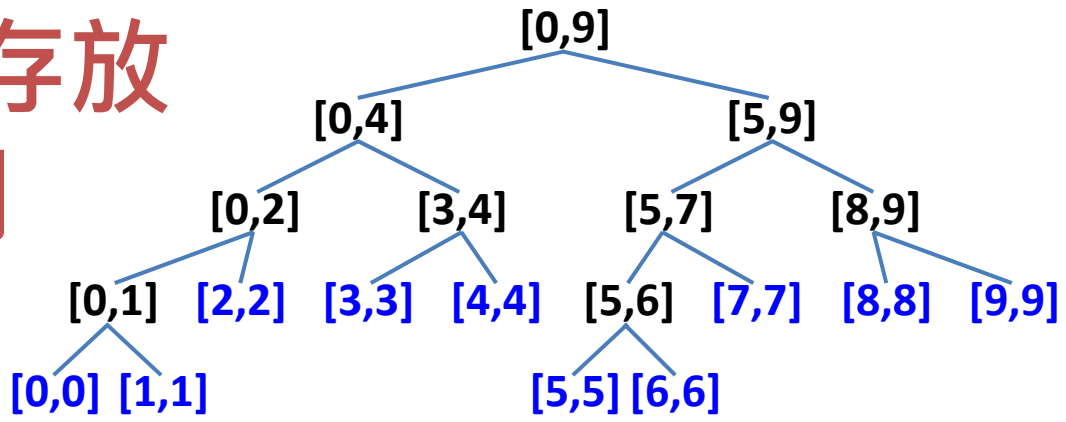
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04		02	34			01	22							00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

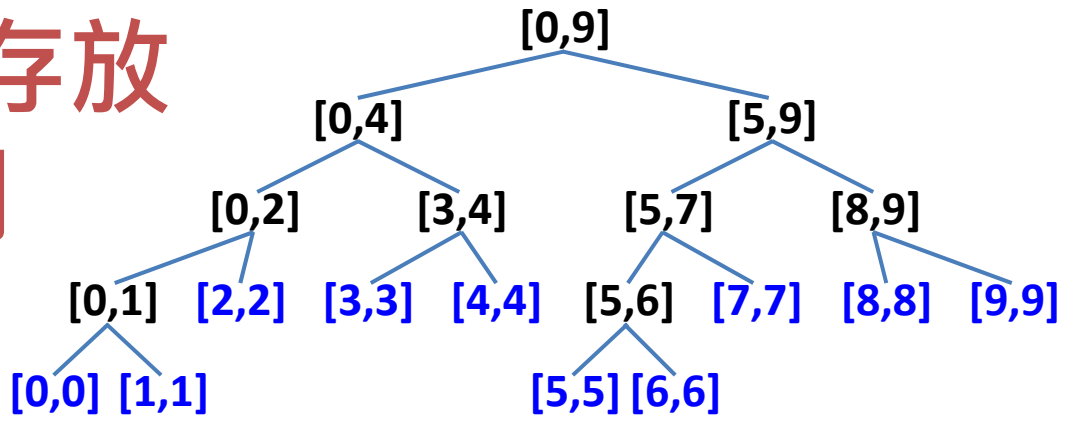
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04		02	34			01	22	33						00	11														

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

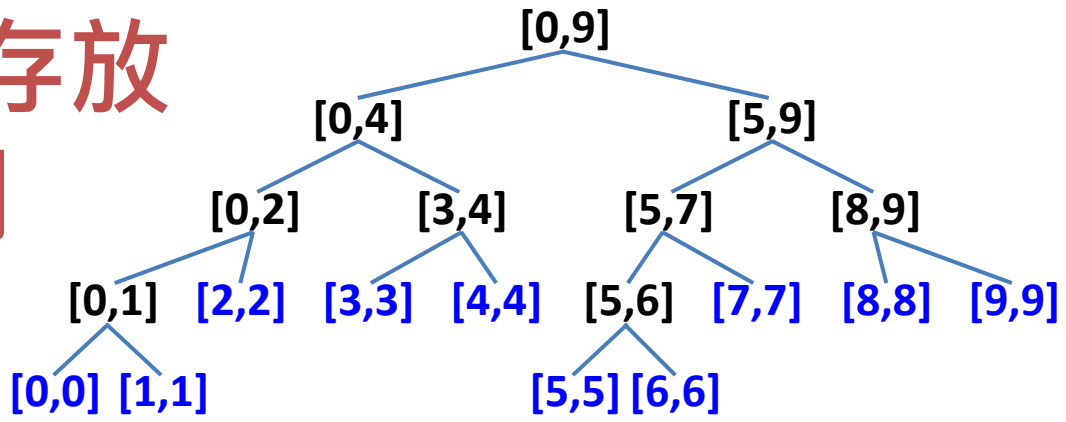
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04		02	34			01	22	33	44					00	11														

二元樹以陣列存放需要的空間

- 二元線段樹



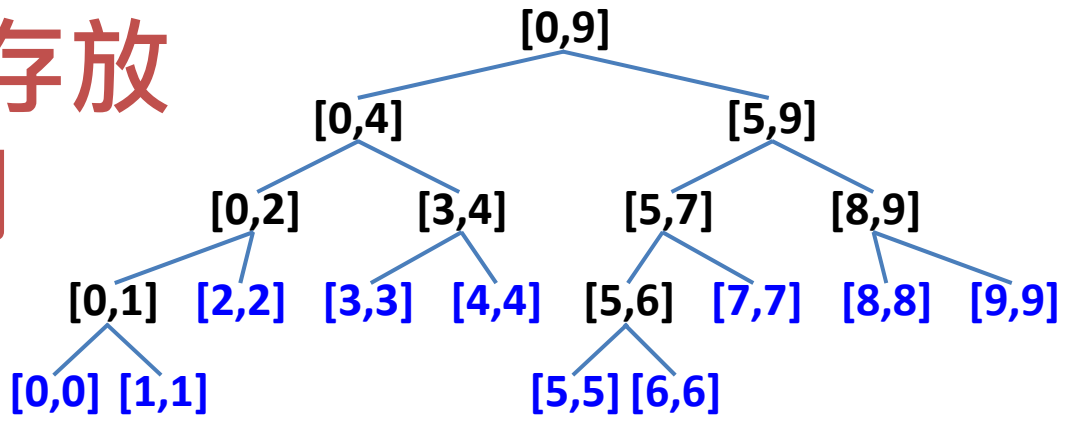
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34			01	22	33	44					00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

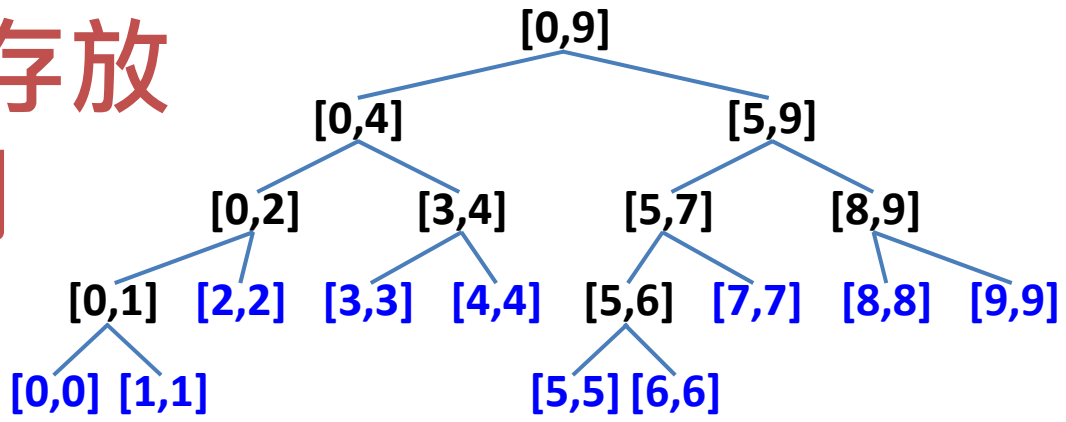
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57		01	22	33	44					00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

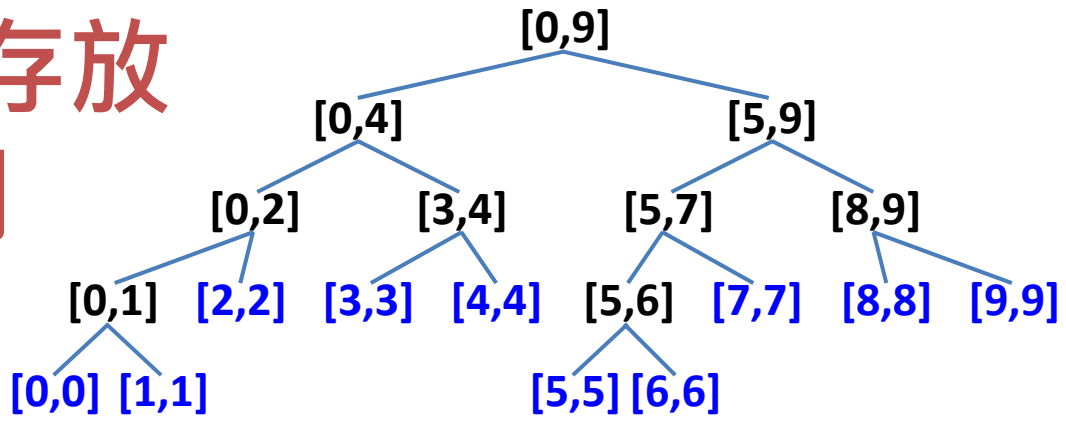
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57		01	22	33	44	56				00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

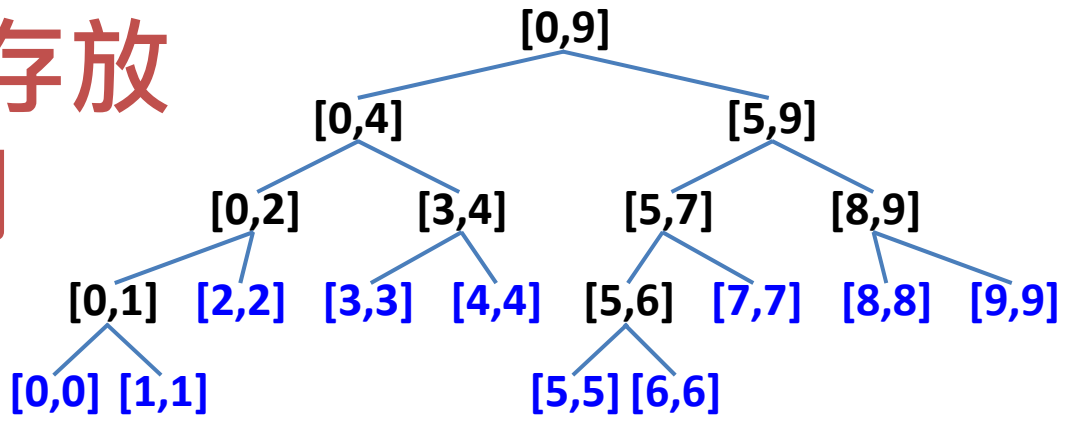
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57		01	22	33	44	56				00	11							55							

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

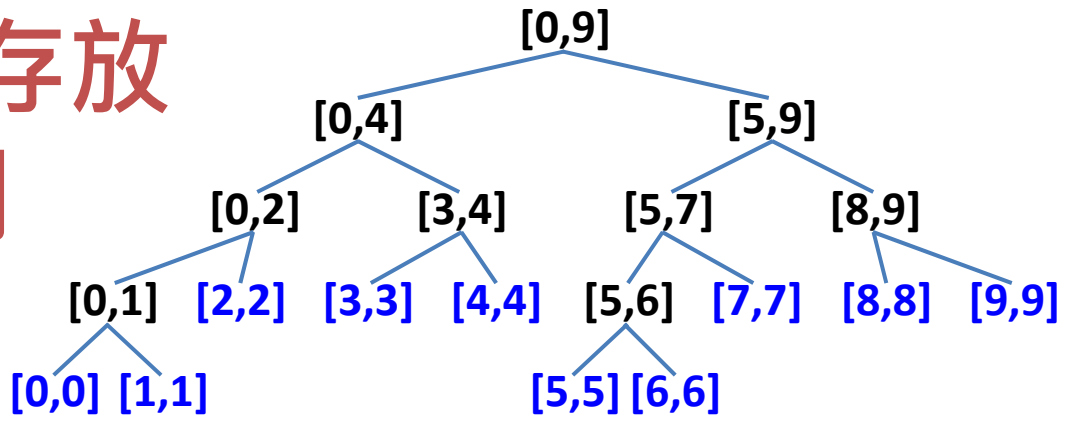
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57		01	22	33	44	56				00	11							55	66						

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

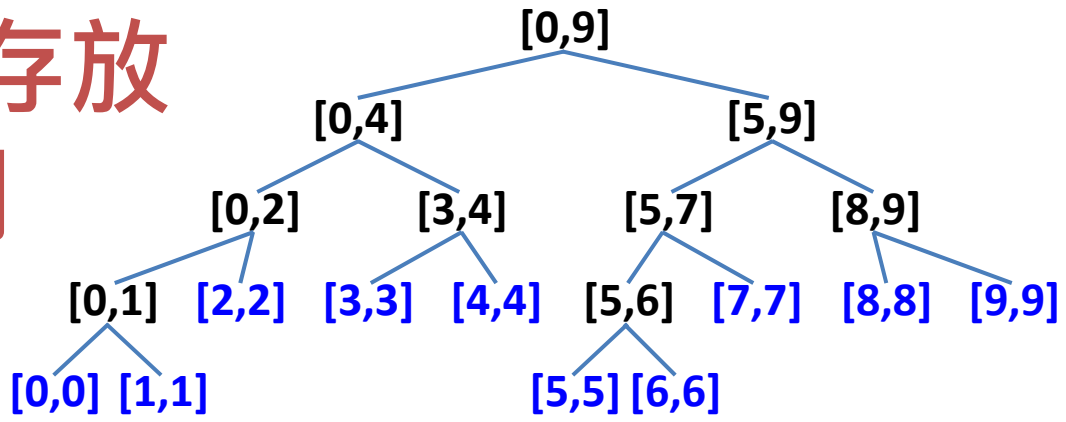
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57		01	22	33	44	56	77			00	11							55	66						

二元樹以陣列存放需要的空間

- 二元線段樹



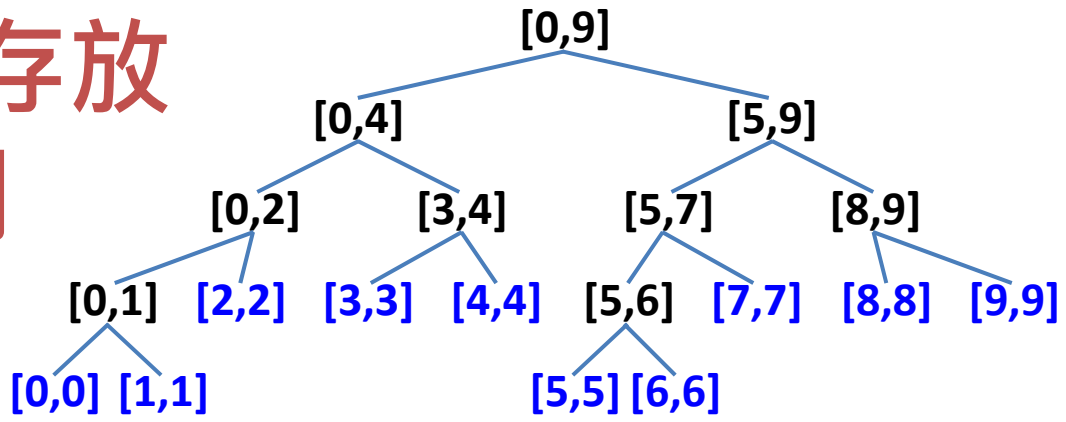
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77			00	11							55	66						

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

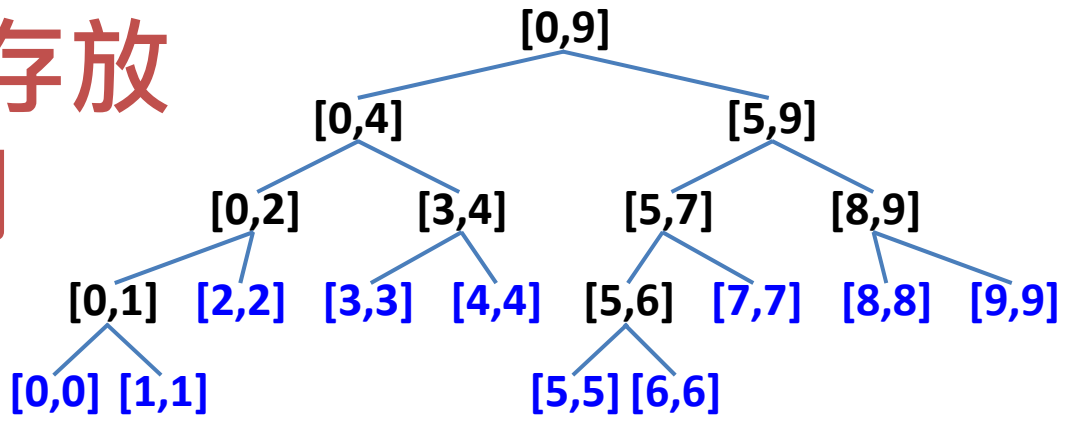
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88		00	11							55	66						

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

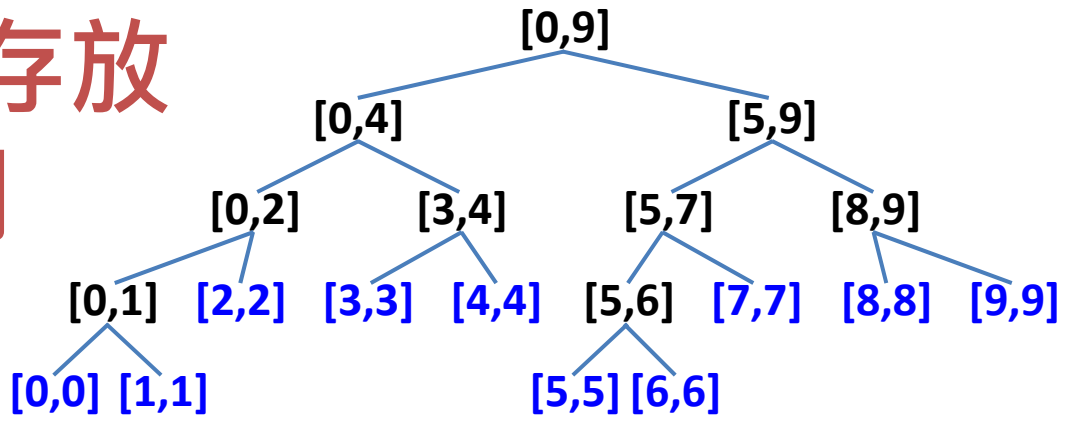
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						

二元樹以陣列存放需要的空間



- 二元線段樹

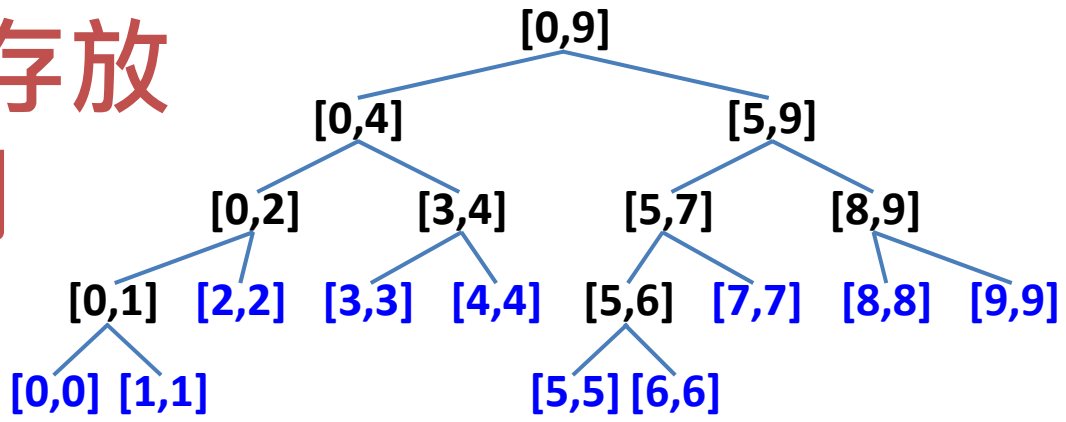
- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i																												

二元樹以陣列存放需要的空間



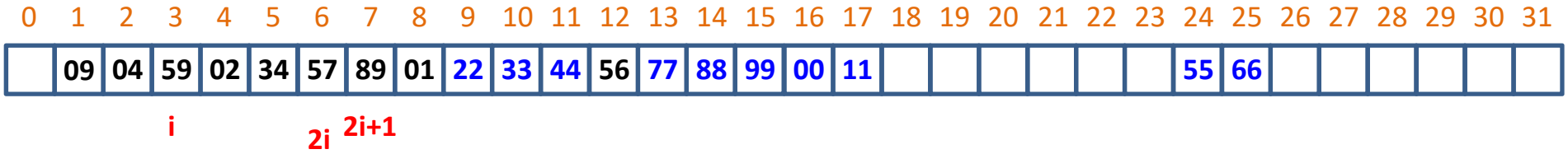
- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

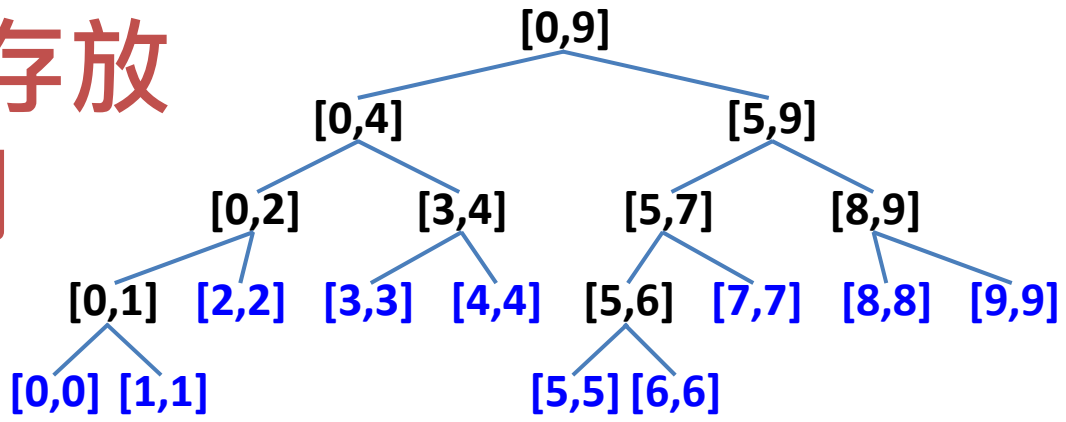
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)

int st[32];



二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

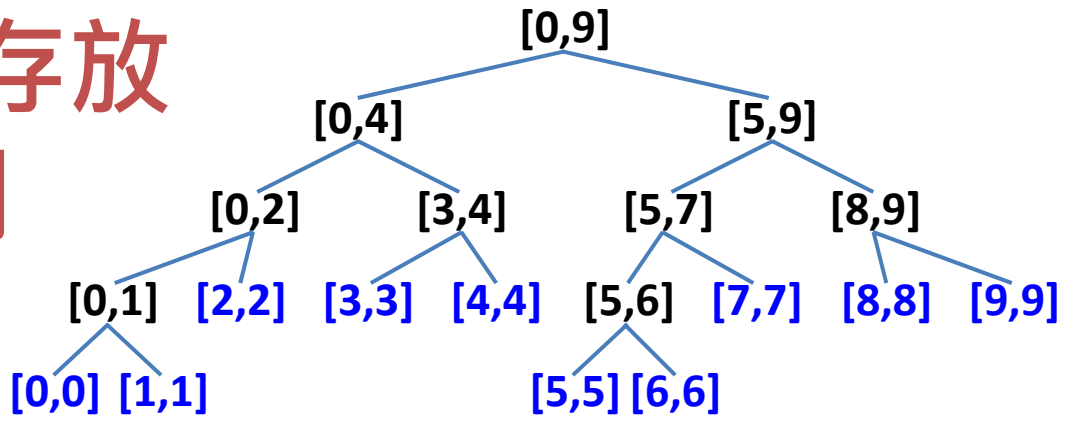
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

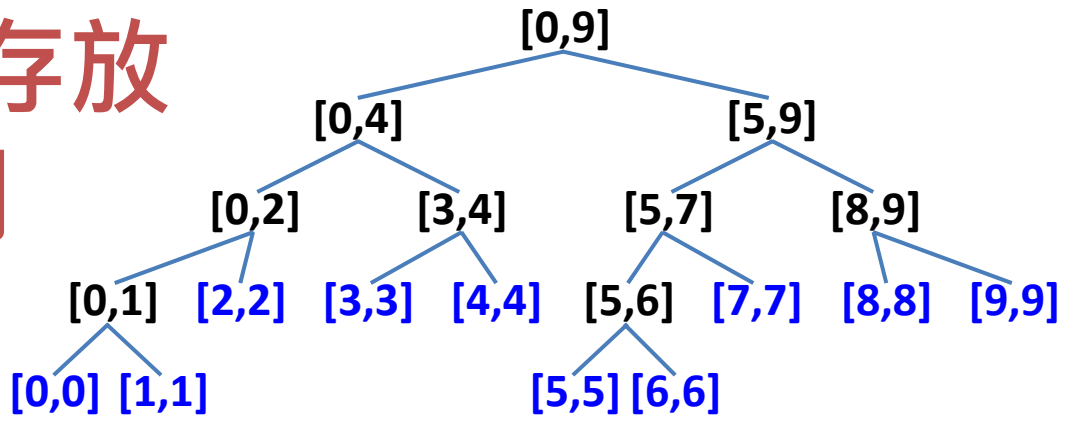
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

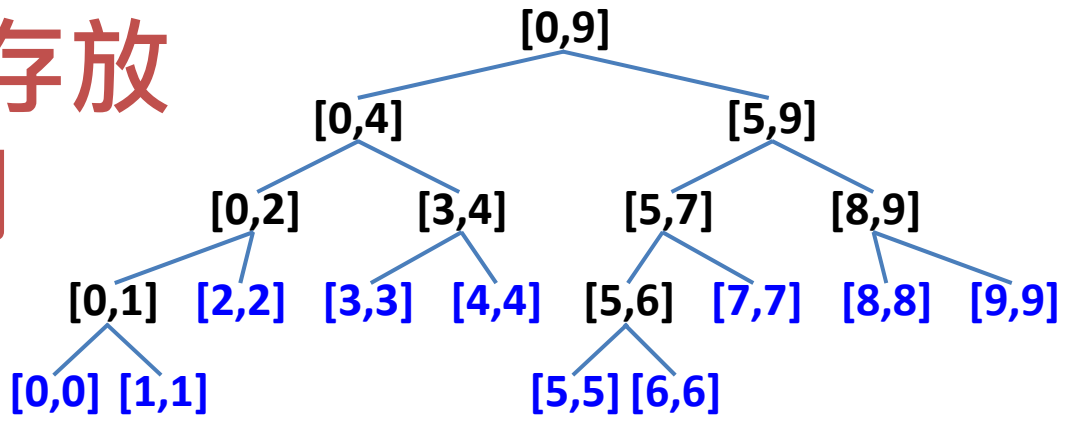
n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

二元樹以陣列存放需要的空間

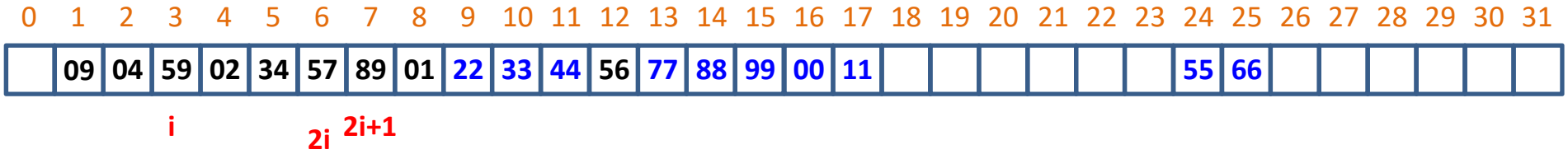


- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

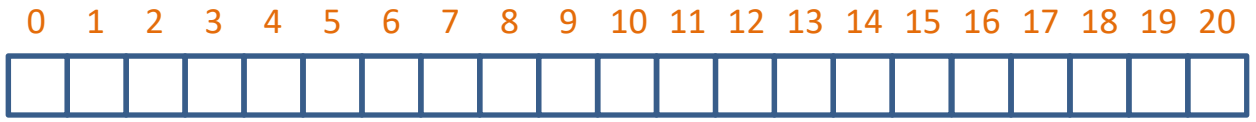
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

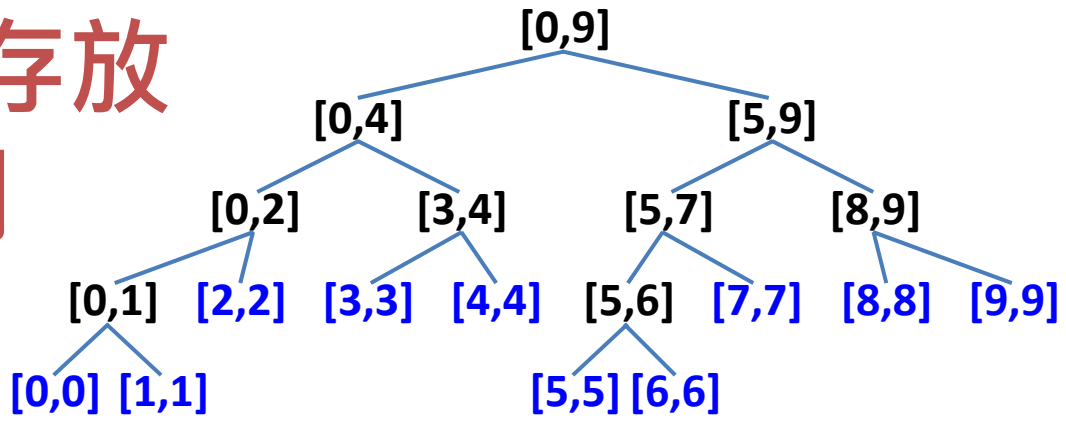


- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];



二元樹以陣列存放需要的空間

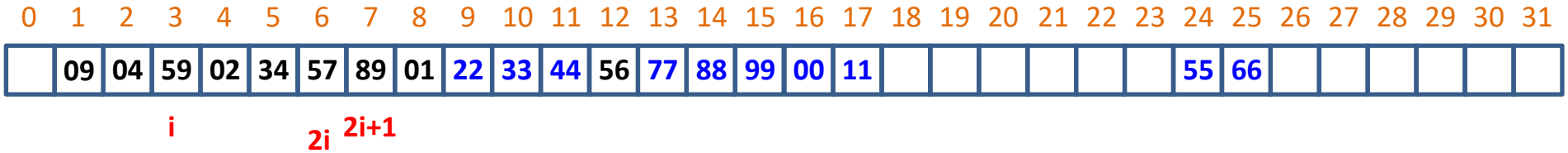


- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

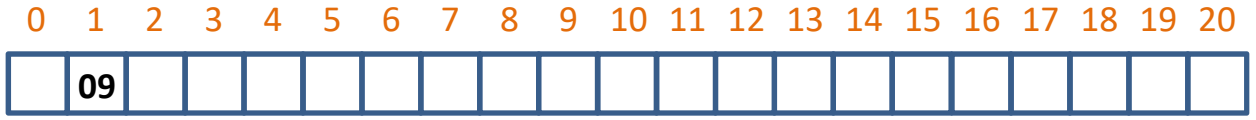
3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

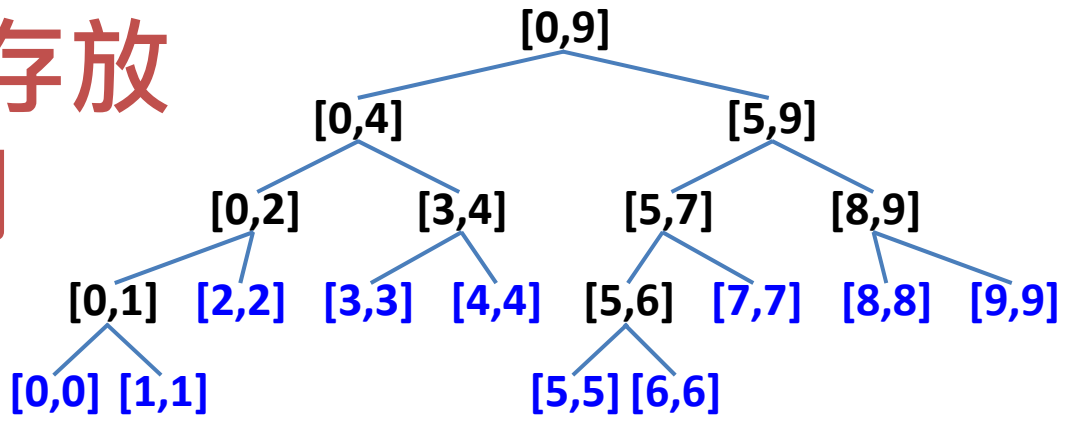


- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];



二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

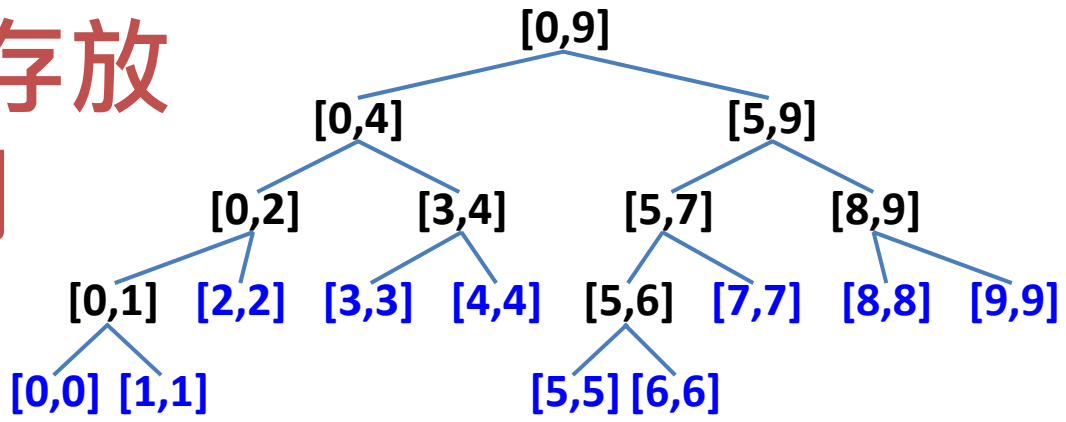
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04																		

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						

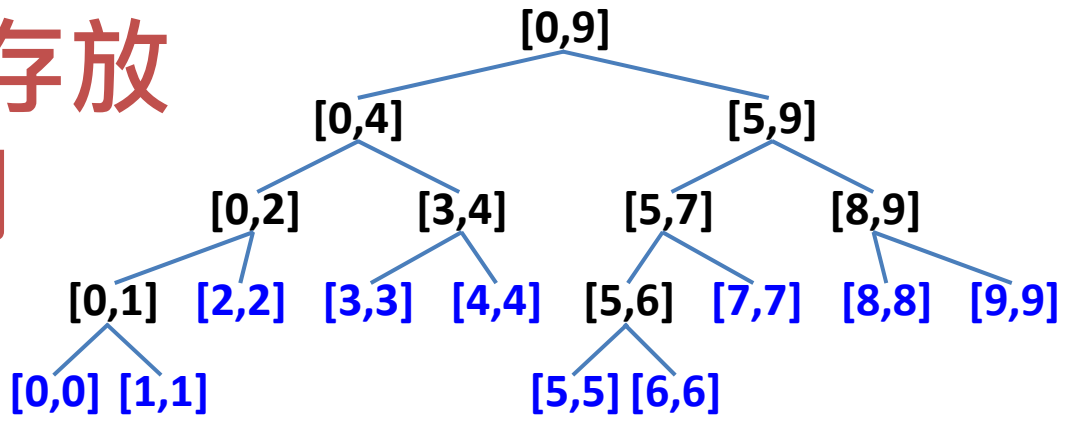
i 2i 2i+1

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02																	

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						

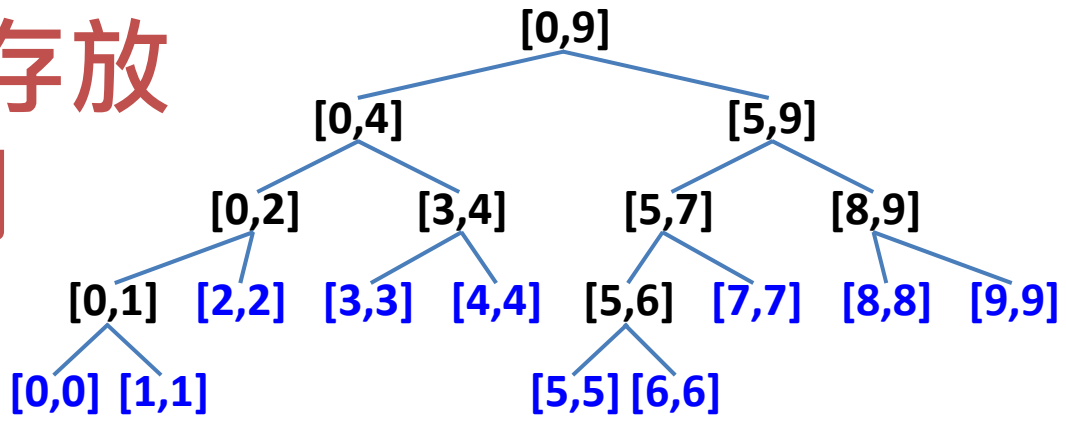
i 2i 2i+1

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01																

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

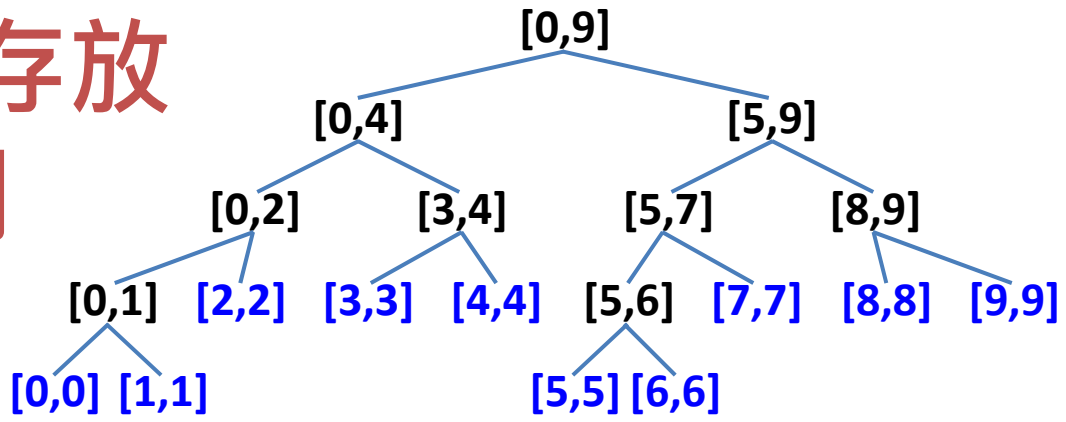
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00															

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

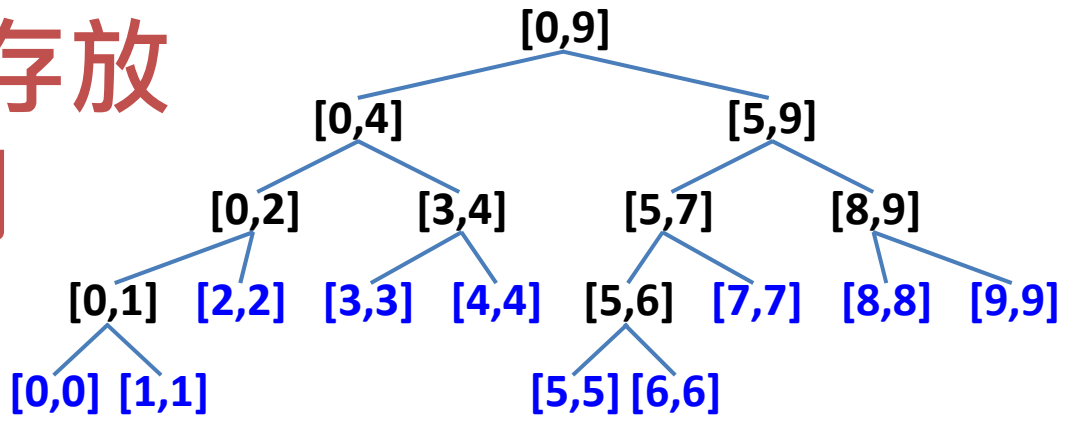
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11														

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

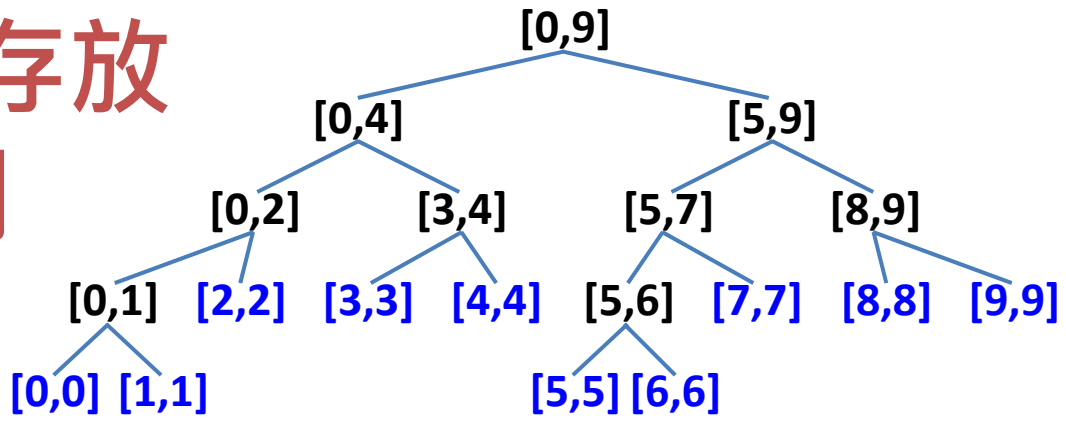
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22													

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

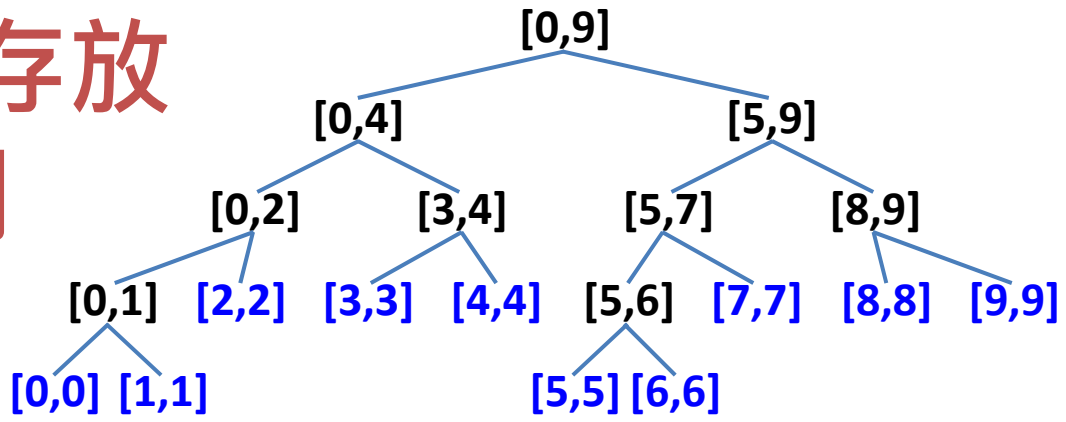
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34												

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

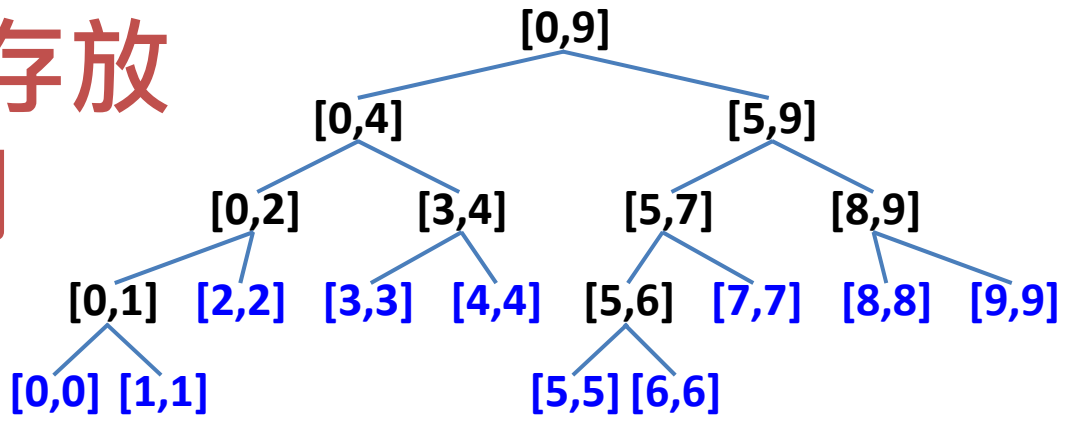
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33											

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

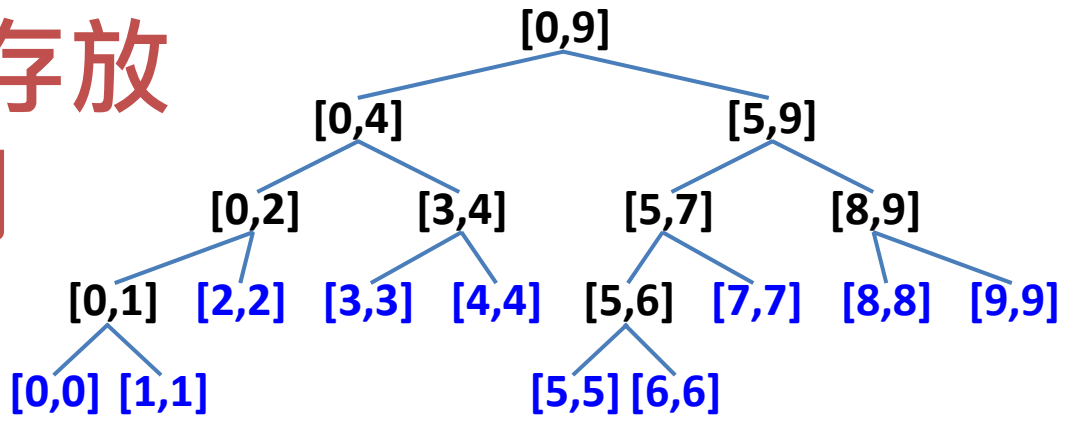
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44										

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

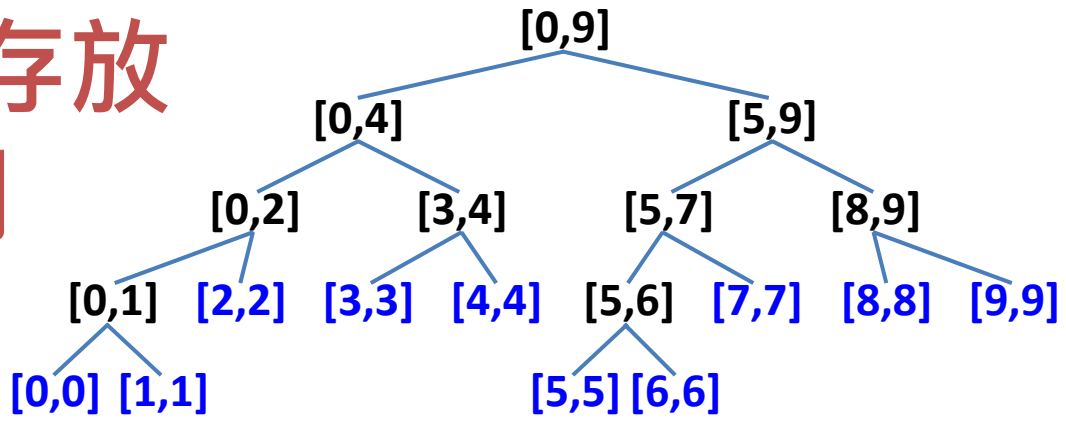
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59									

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

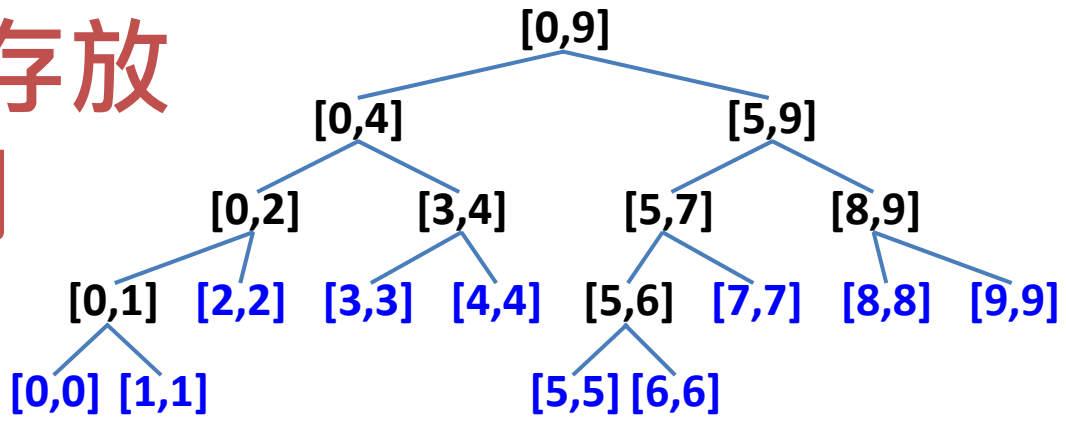
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57								

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

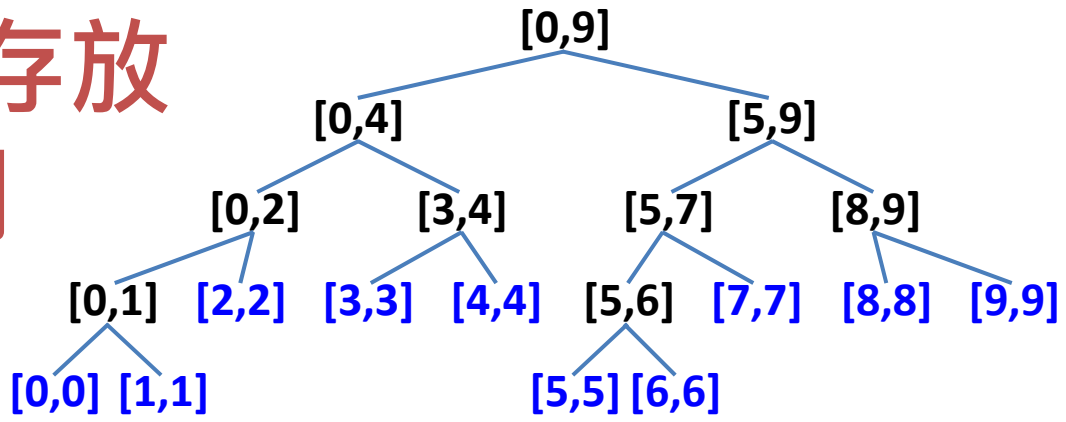
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56							

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

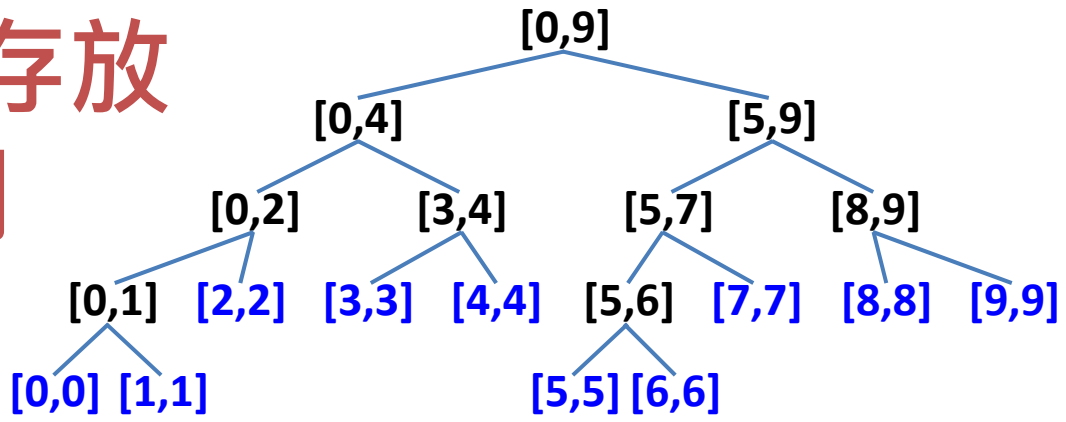
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55						

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

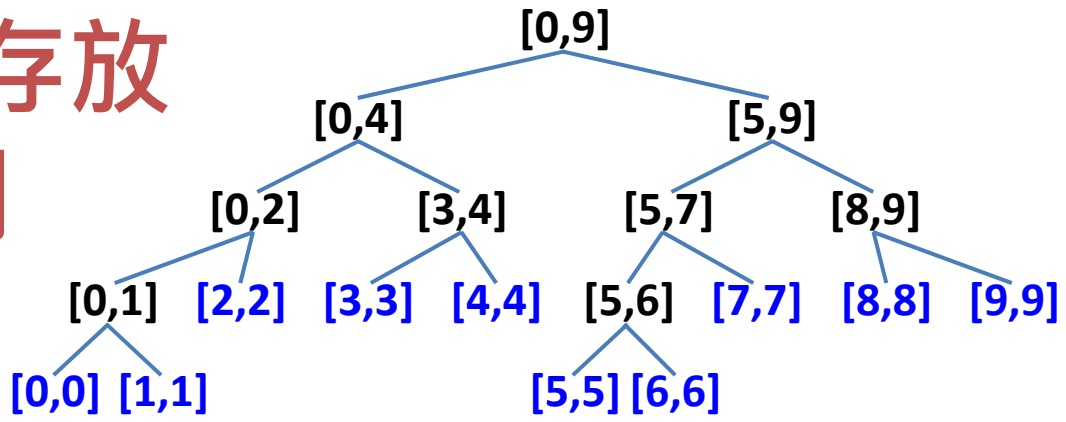
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66					

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

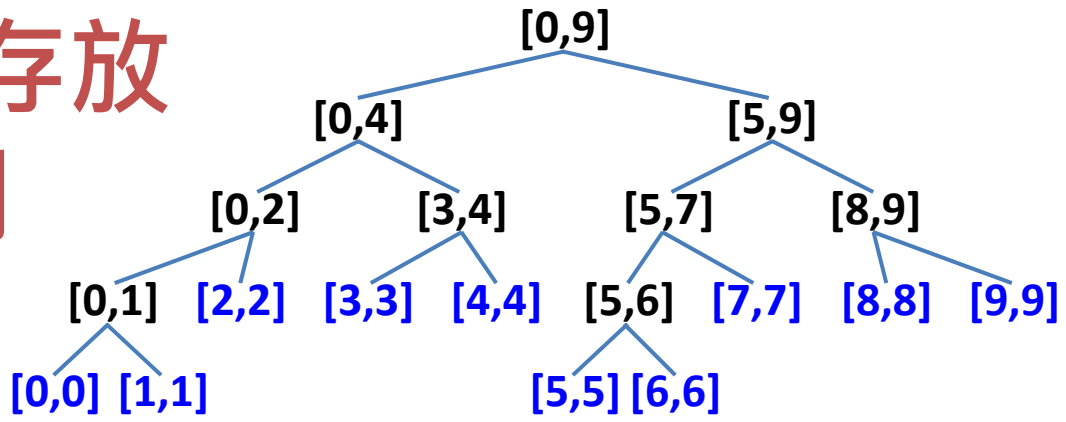
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77				

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

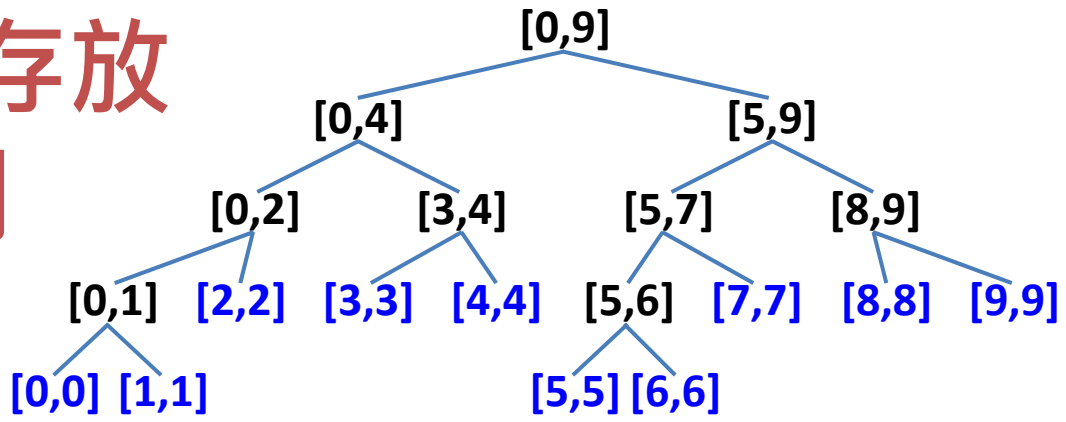
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89			

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

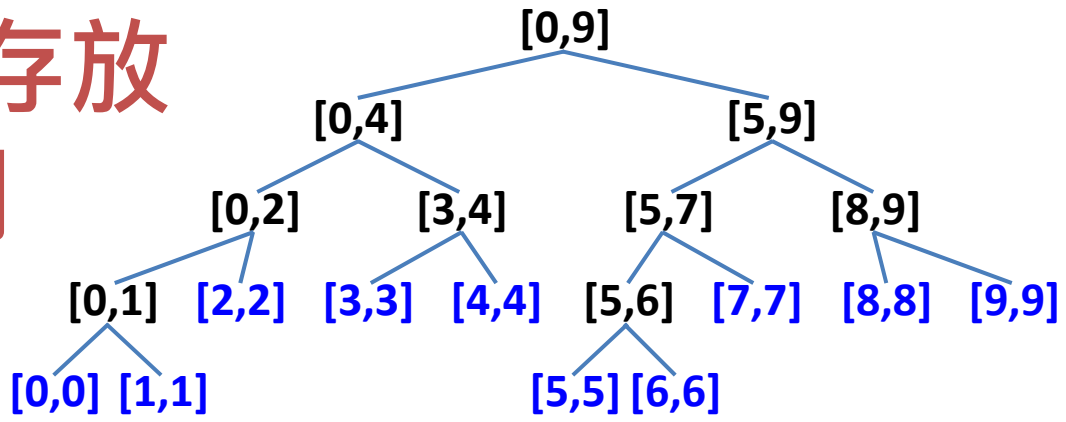
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89	88		

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

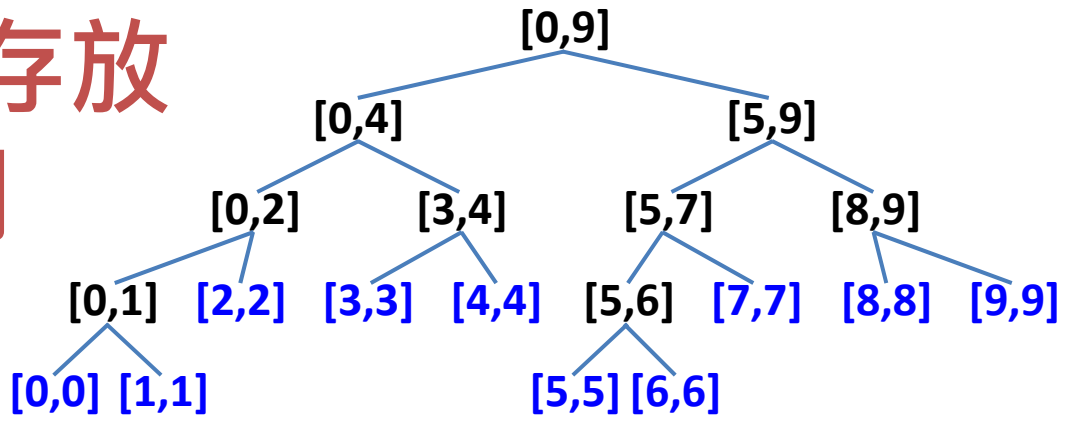
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89	88	99	

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

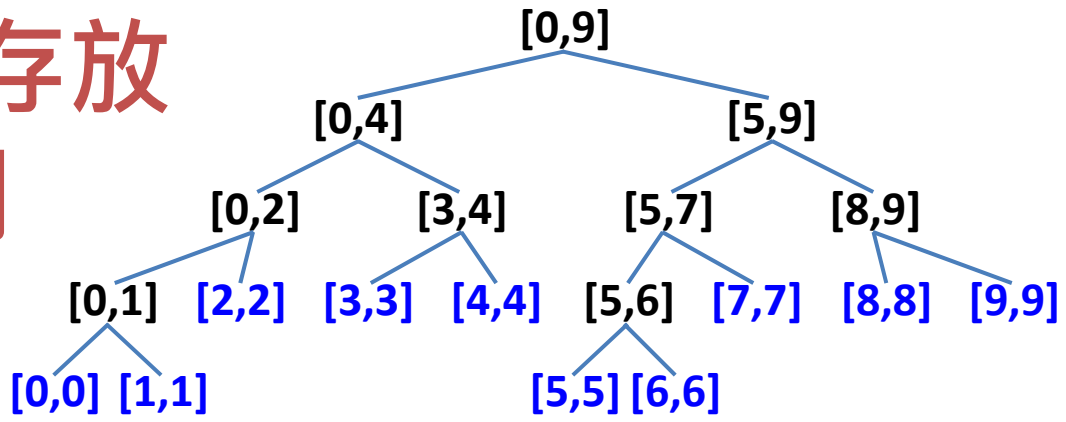
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89	88	99	
			i																	

二元樹以陣列存放需要的空間



- 二元線段樹

- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal) int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

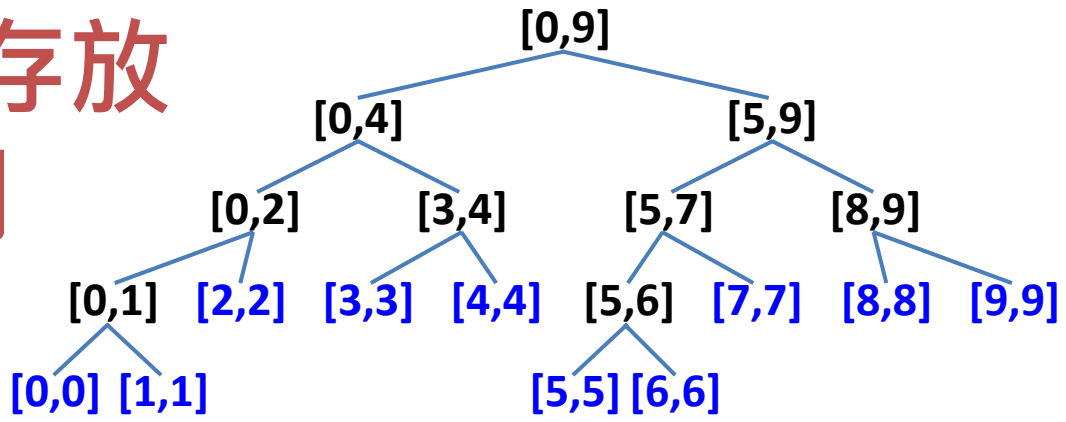
- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10 int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89	88	99	
			i	i+1			i+4													

二元樹以陣列存放需要的空間

- 二元線段樹



- n 個葉節點以 level-order 存放⇒ 陣列長度 O(4n)

3n=30 不夠大

n = 10 ⇒ $2^3 < n \leq 2^4$ ⇒ $2^4 * 2$ (internal)　　int st[32];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	09	04	59	02	34	57	89	01	22	33	44	56	77	88	99	00	11							55	66						
			i			2i	2i+1																								

- n 個葉節點以 pre-order 存放⇒ 陣列長度 O(2n)

n = 10　　int st[21];

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	09	04	02	01	00	11	22	34	33	44	59	57	56	55	66	77	89	88	99	
			i	i+1			i+4													

陣列中間完全沒有空著的元素

線段樹的實作 (I)

- **Range Min Query**

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for
  the storage of a full binary tree

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for
  the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for
  the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

- Recursive **construction**

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for
  the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

- Recursive **construction**

  build(1, 0,n-1);

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for
  the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

v

0~7							
0~3				4~7			
0~1		2~3		4~5		6~7	
0	1	2	3	4	5	6	7

- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v

0~7							
0~3				4~7			
0~1		2~3		4~5		6~7	
0	1	2	3	4	5	6	7

- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v

0~7							
0~3				4~7			
0~1		2~3		4~5		6~7	
0	1	2	3	4	5	6	7
8	**7**	**3**	**9**	**5**	**1**	**10**	**4**

- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

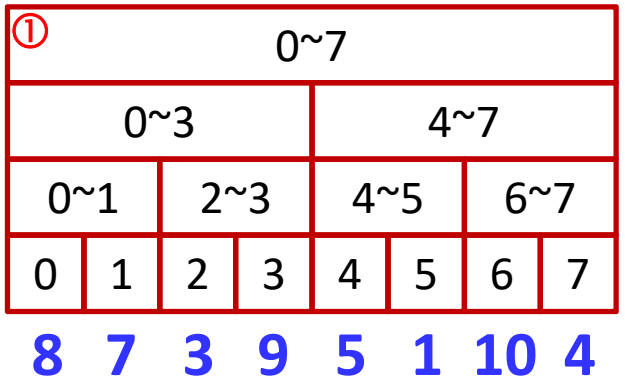
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

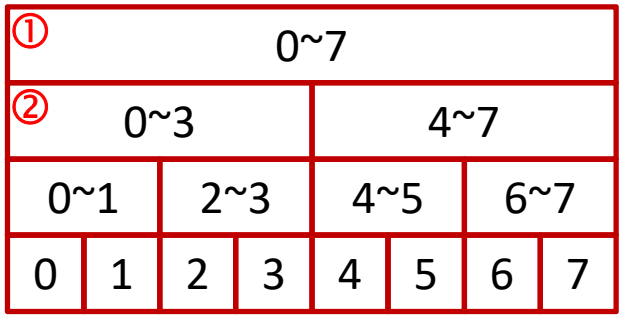
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

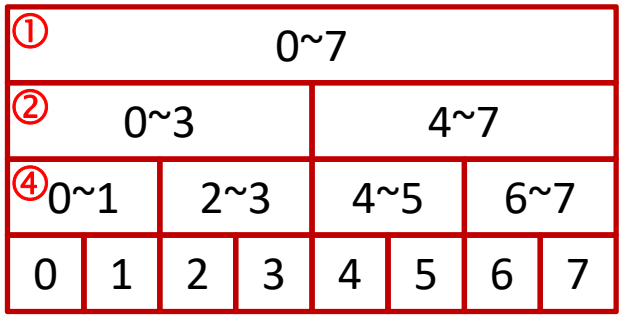
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

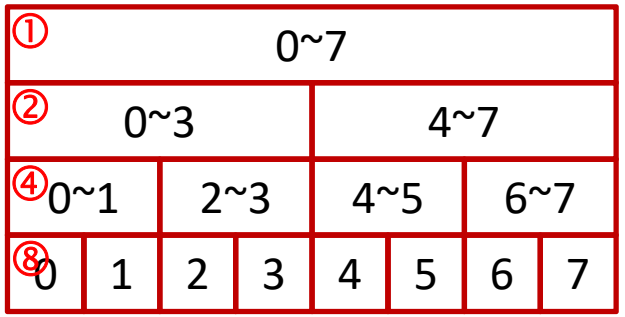
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

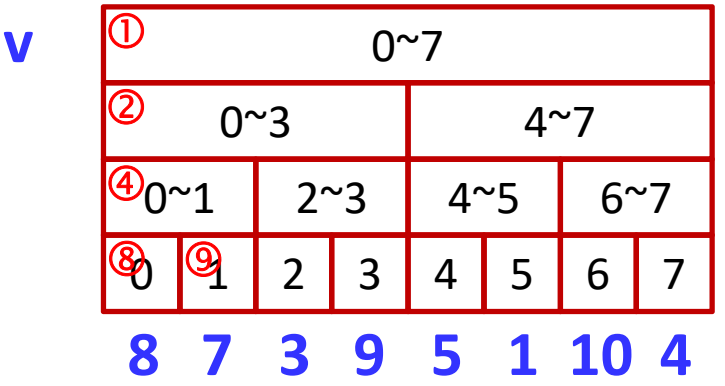
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

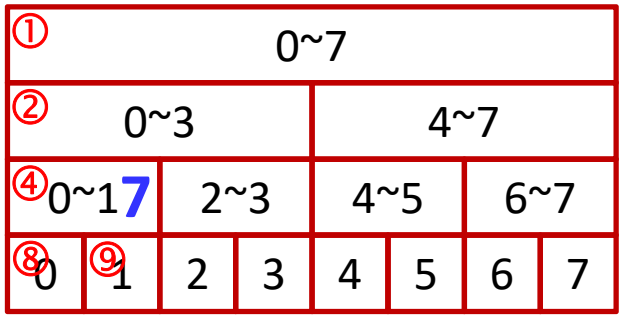
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

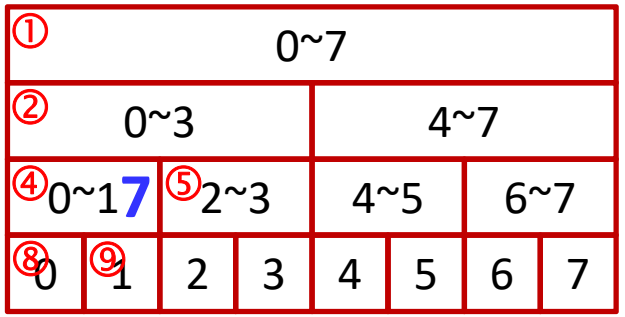
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

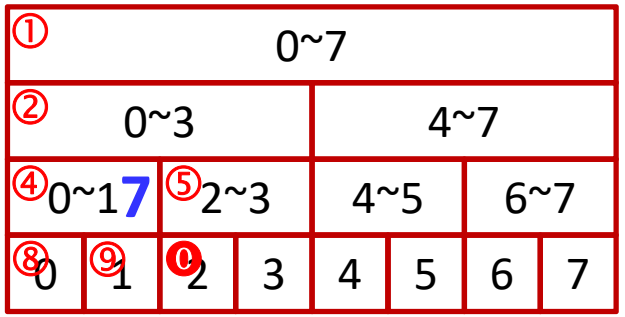
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

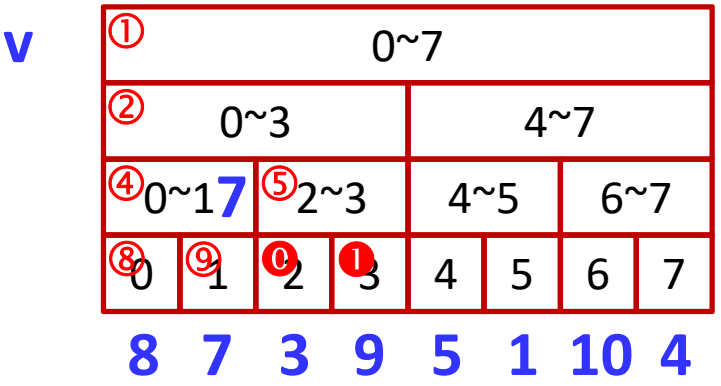
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

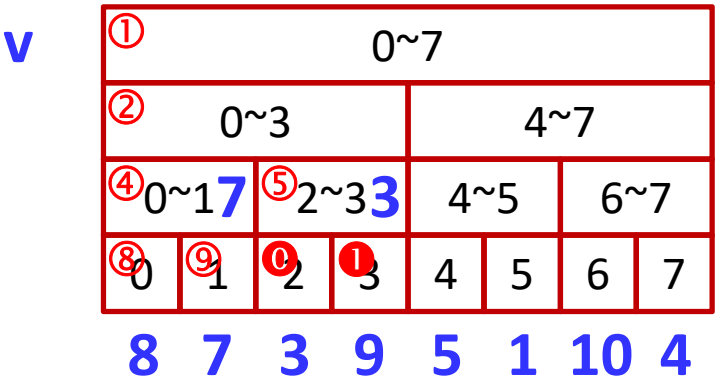
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

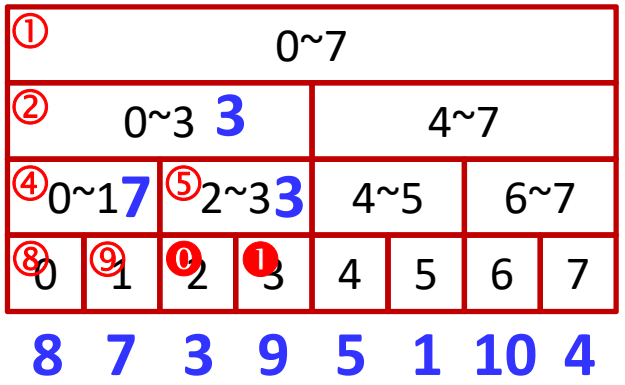
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

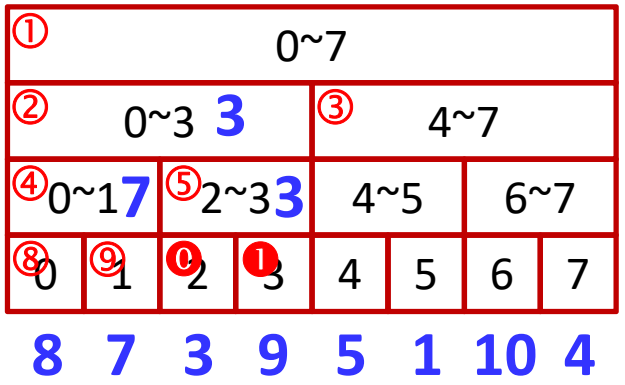
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

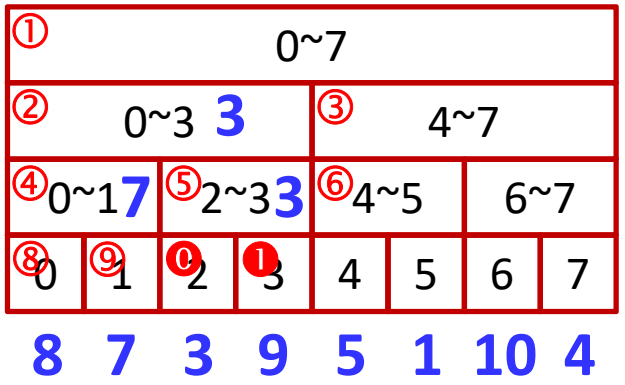
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

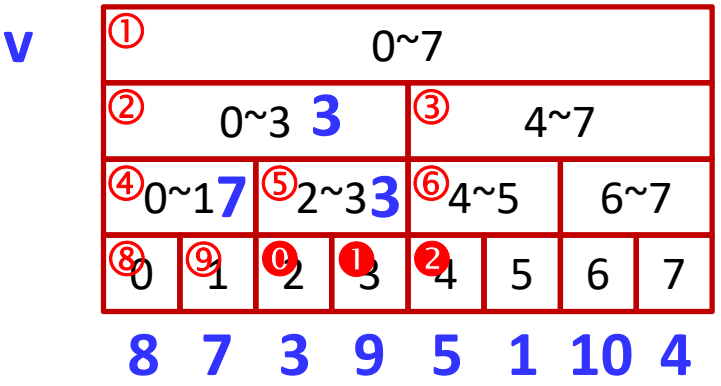
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

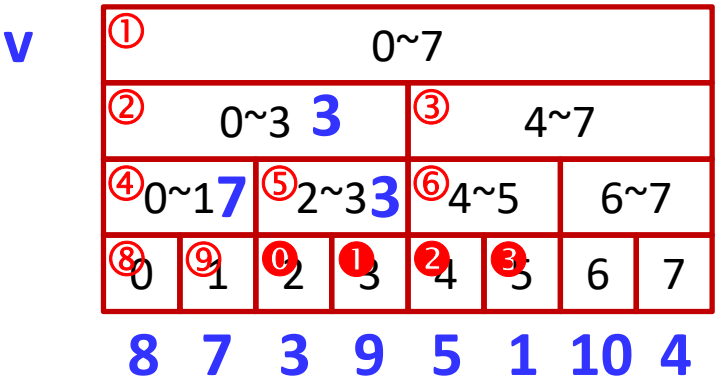
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

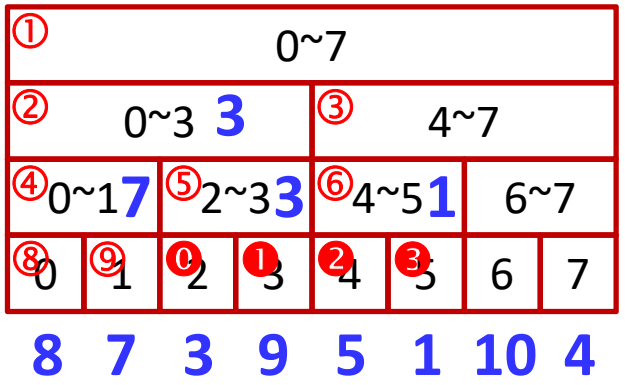
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv*2, ld,(ld+rd)/2),
     build(iv*2+1, (ld+rd)/2+1,rd),
     v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

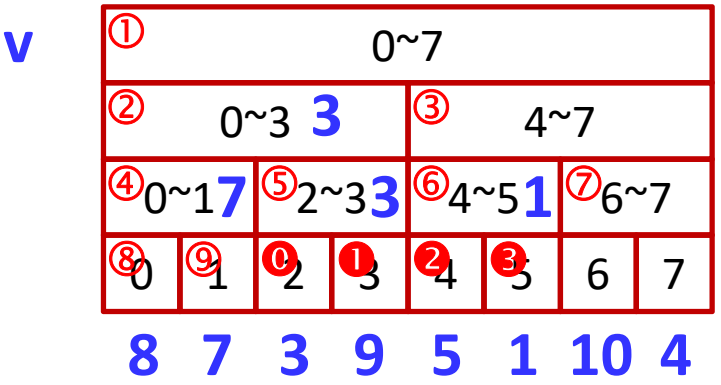
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv*2, ld,(ld+rd)/2),
     build(iv*2+1, (ld+rd)/2+1,rd),
     v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

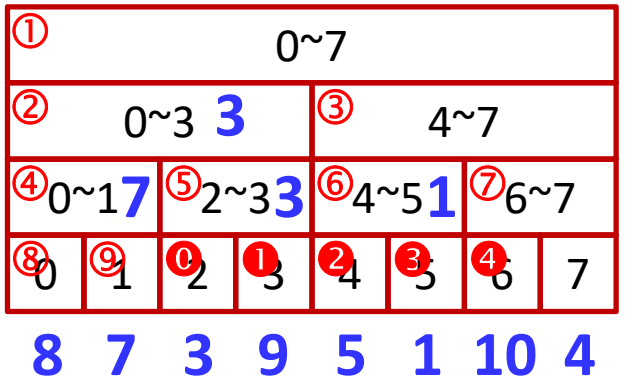
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

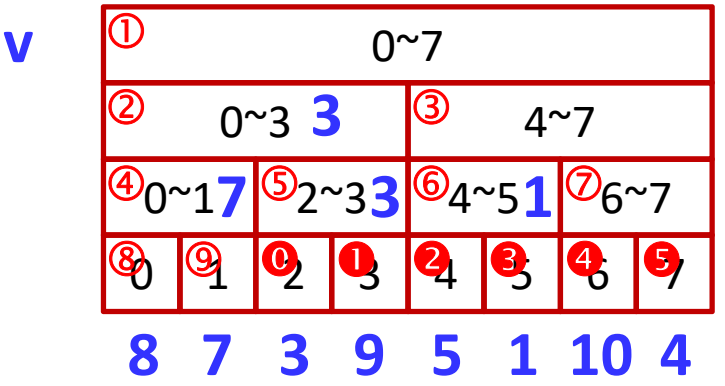
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

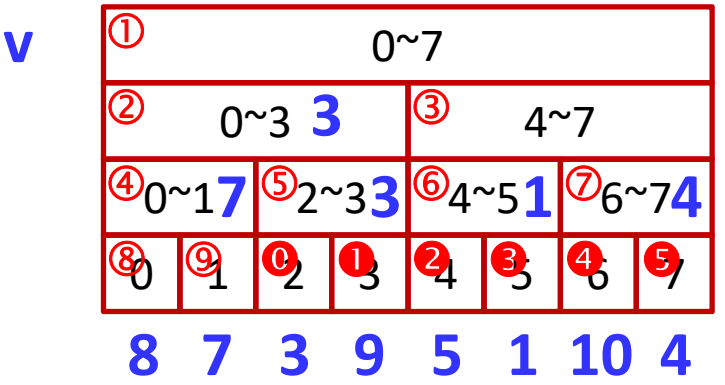
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

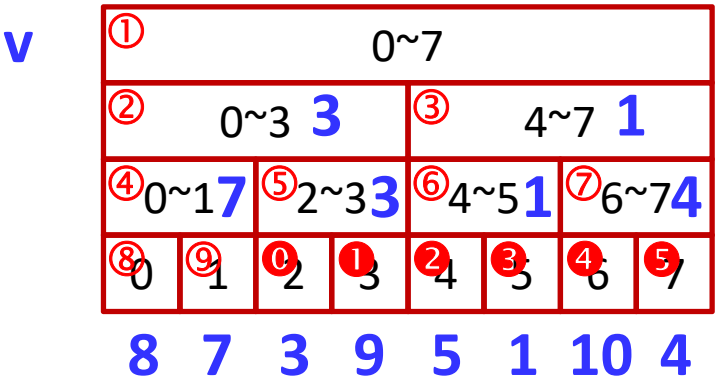
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

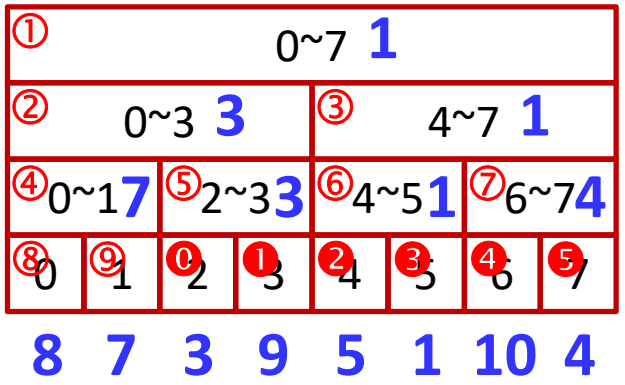
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

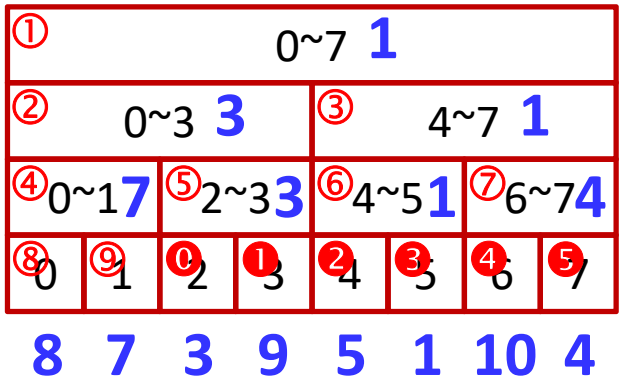
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	8
Array size	16

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

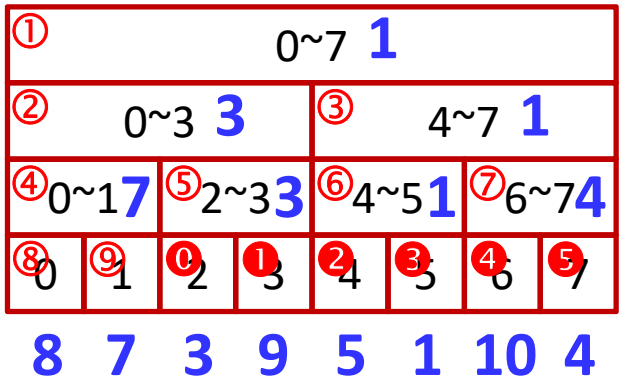
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8
Array size	14	16

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

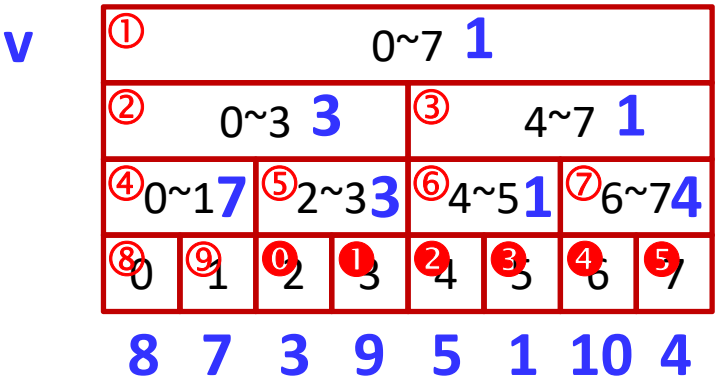
線段樹的實作 (I)

- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9
Array size	14	16	18

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

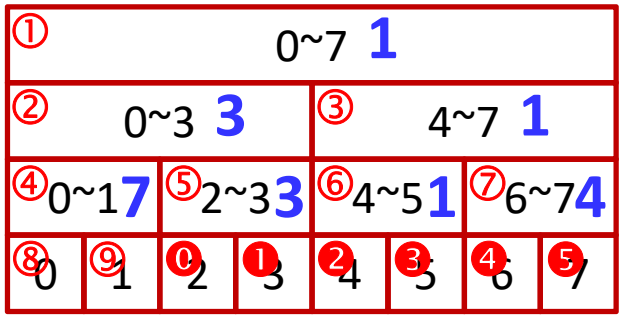
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv*2, ld,(ld+rd)/2),
     build(iv*2+1, (ld+rd)/2+1,rd),
     v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

線段樹的實作 (I)

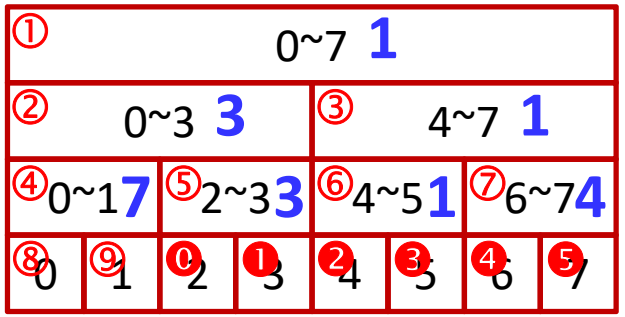
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

- Recursive **RmQ query [lq,rq]**

線段樹的實作 (I)

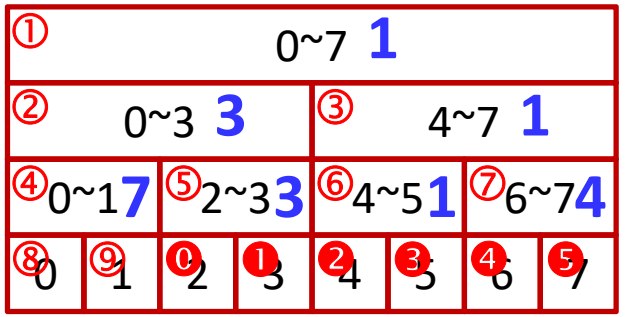
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

- Recursive **RmQ query [lq,rq]**

```
int RmQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x7fffffff;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;
  if (rq<=md) return RmQ(2*iv, ld,md, lq,rq);
  else if (lq>md) return RmQ(2*iv+1, md+1,rd, lq,rq);
  return min(RmQ(2*iv, ld,md, lq, md),
             RmQ(2*iv+1, md+1,rd, md+1,rq));
}
```

線段樹的實作 (I)

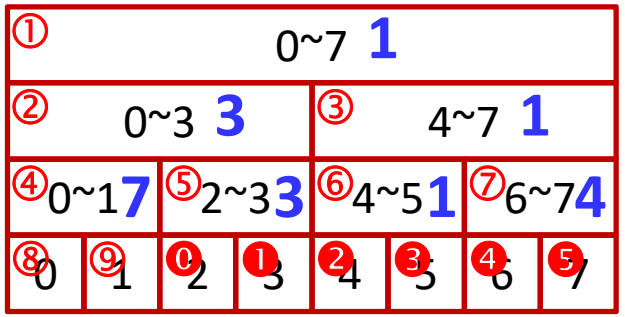
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

- Recursive **RmQ query [lq,rq]**

```
int RmQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x7fffffff;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;
  if (rq<=md) return RmQ(2*iv, ld,md, lq,rq);
  else if (lq>md) return RmQ(2*iv+1, md+1,rd, lq,rq);
  return min(RmQ(2*iv, ld,md, lq,md),
             RmQ(2*iv+1, md+1,rd, md+1,rq));
}
```

線段樹的實作 (I)

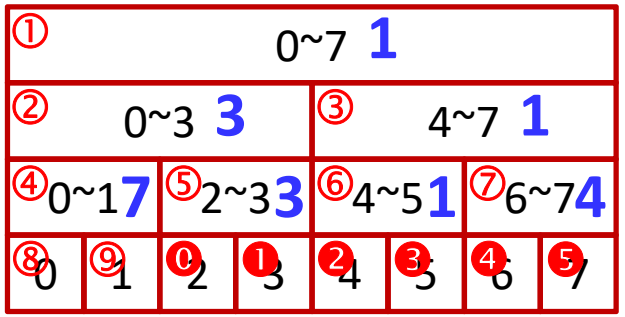
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

- Recursive **RmQ query [lq,rq]**

```
int RmQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x7fffffff;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;   這一段程式拿掉以後還是正確的
  if (rq<=md) return RmQ(2*iv, ld,md, lq,rq);
  else if (lq>md) return RmQ(2*iv+1, md+1,rd, lq,rq);
  return min(RmQ(2*iv, ld,md, lq,md),
             RmQ(2*iv+1, md+1,rd, md+1,rq));
}
```

線段樹的實作 (I)

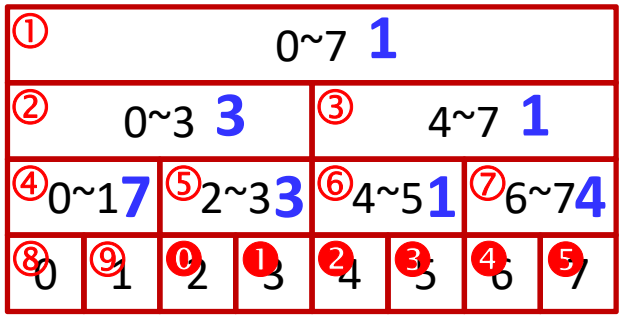
- **Range Min Query**

  O(4n) level-order array for the storage of a full binary tree

```
#define MAXN 500
int d[MAXN], v[4*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

  build(1, 0,n-1);

n	7	8	9	10	11	12	13	14	15	16
Array size	14	16	18	26	26	30	30	30	30	32

```
void build(int iv, int ld, int rd) {
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv*2, ld,(ld+rd)/2),
    build(iv*2+1, (ld+rd)/2+1,rd),
    v[iv]=min(v[iv*2], v[iv*2+1]);
}
```

- Recursive **RmQ query [lq,rq]**

```
int RmQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x7fffffff;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;
  if (rq<=md) return RmQ(2*iv, ld,md, lq,rq);
  else if (lq>md) return RmQ(2*iv+1, md+1,rd, lq,rq);
  return min(RmQ(2*iv, ld,md, lq,md),
             RmQ(2*iv+1, md+1,rd, md+1,rq));
}
```

但是執行時間變成 7 倍

這一段程式拿掉以後還是正確的

- test RmQ

- Recursive update

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
```

- Recursive update

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
```

- Recursive update

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

ld<=t<=rd

```
void update(int iv, int ld,int rd, int t,int val) {

}
```

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

ld<=t<=rd

```
void update(int iv, int ld,int rd, int t,int val) {
```

```
update(1, 0,7, 5,6);
```

```
}
```

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

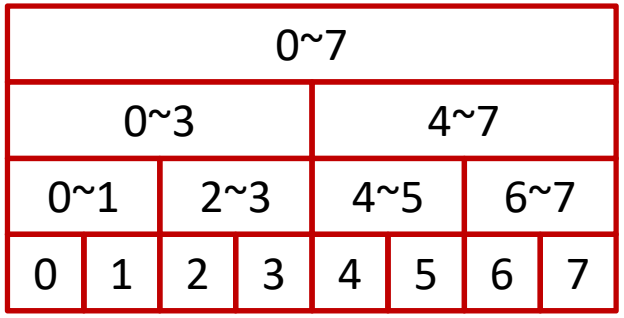
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);

v




```
}
```

- test RmQ

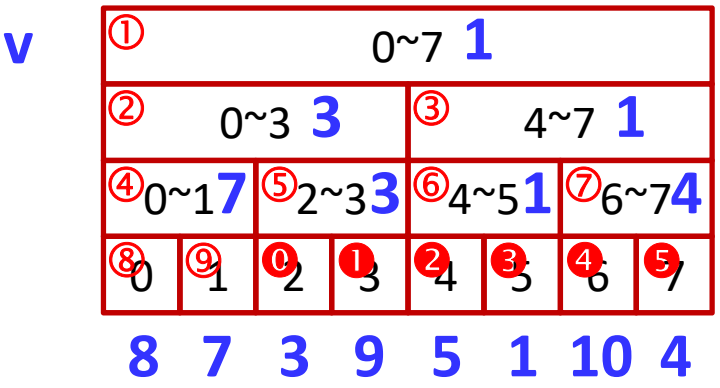
```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);




```
}
```

- test RmQ

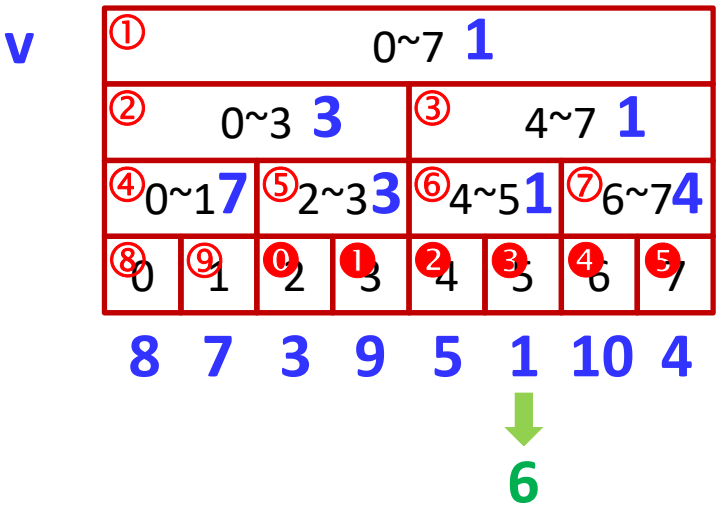
```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);




```
}
```

- test RmQ

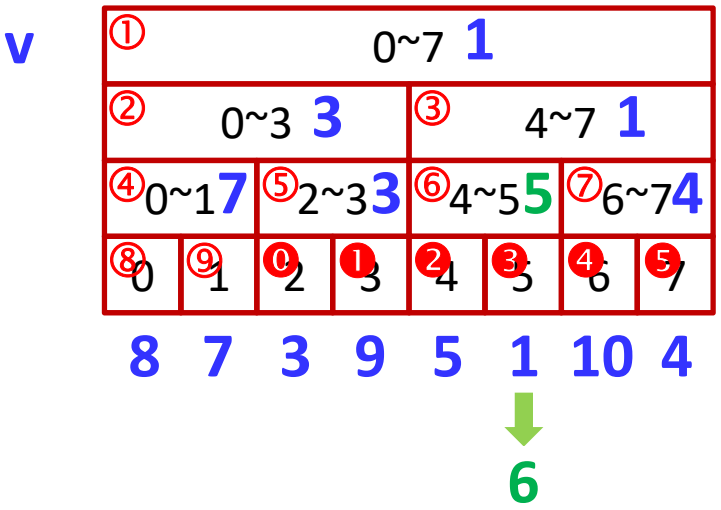
```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);




```
}
```

- test RmQ

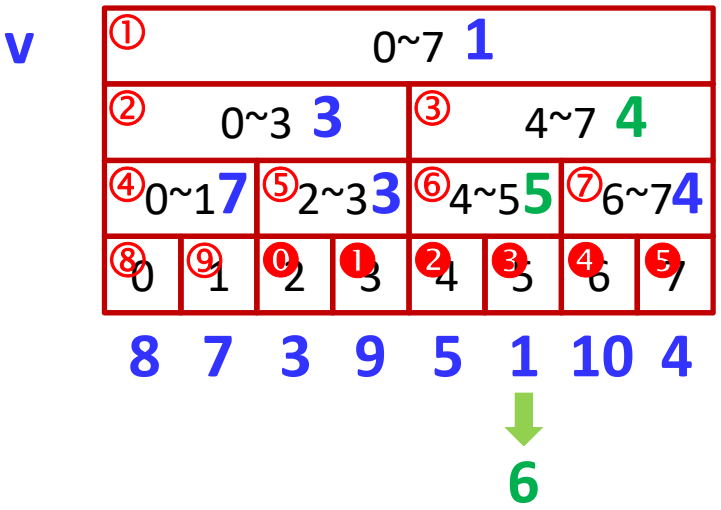
```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);




```
}
```

- test RmQ

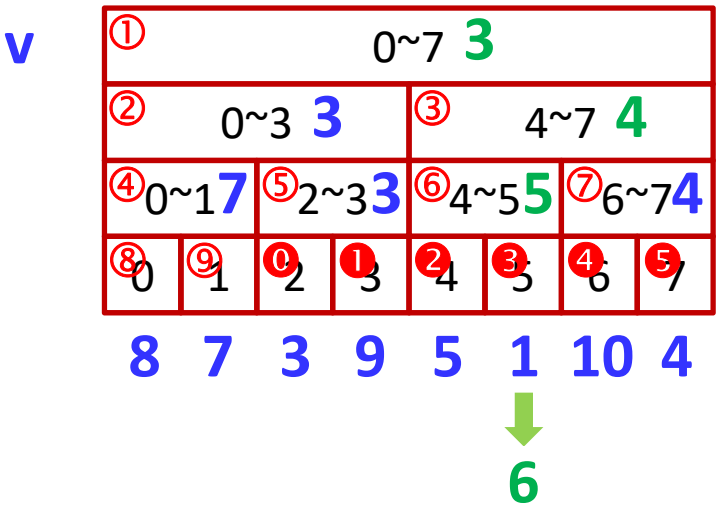
```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
```

ld<=t<=rd

update(1, 0,7, 5,6);




```
}
```

- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

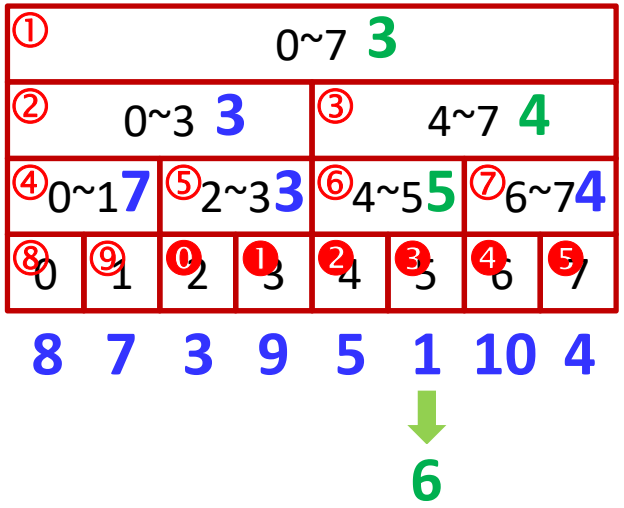
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {




    }
}
```

ld<=t<=rd

update(1, 0,7, 5,6);

v



- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

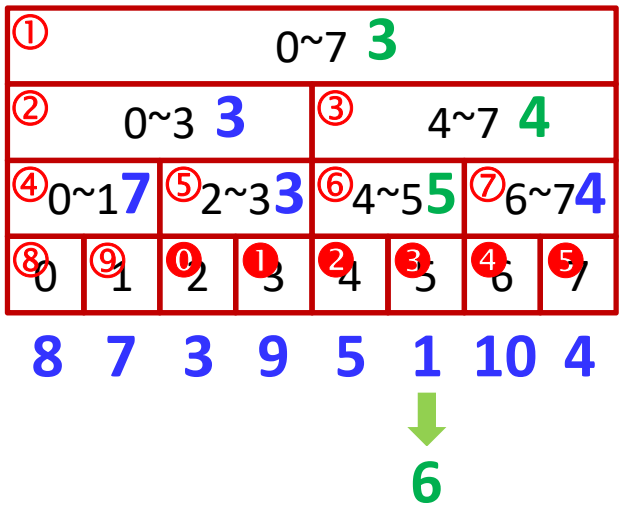
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
   if (ld==rd) // ==t
      v[iv] = val;
   else {
      int md=(ld+rd)/2;
      if (t<=md)
         update(2*iv, ld,md, t,val);
      else
         update(2*iv+1, md+1,rd, t,val);

   }
}
```

ld<=t<=rd

update(1, 0,7, 5,6);

v



- test RmQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
   scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
   printf("%d\n", RmQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

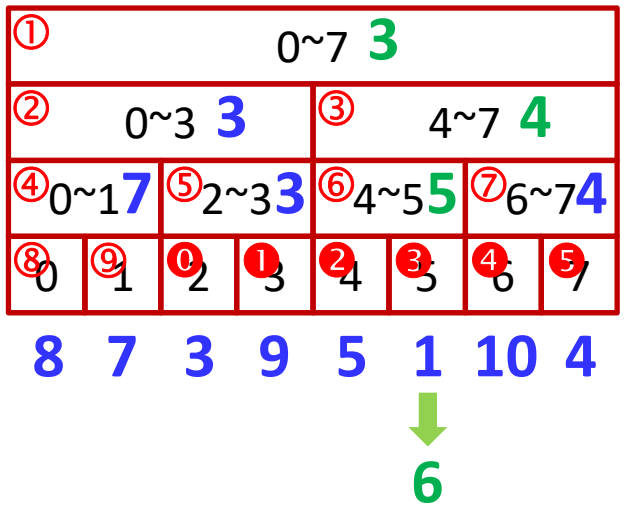
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
   if (ld==rd) // ==t
      v[iv] = val;
   else {
      int md=(ld+rd)/2;
      if (t<=md)
         update(2*iv, ld,md, t,val);
      else
         update(2*iv+1, md+1,rd, t,val);
      v[iv] = min(v[2*iv], v[2*iv+1]);
   }
}
```

ld<=t<=rd

update(1, 0,7, 5,6);

v



線段樹的實作 (II)

- **Range Max Query**

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

- Recursive **construction**

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

- Recursive **construction**

```
build(1, 0,n-1);
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

- Recursive **construction**

```
build(1, 0,n-1);
```

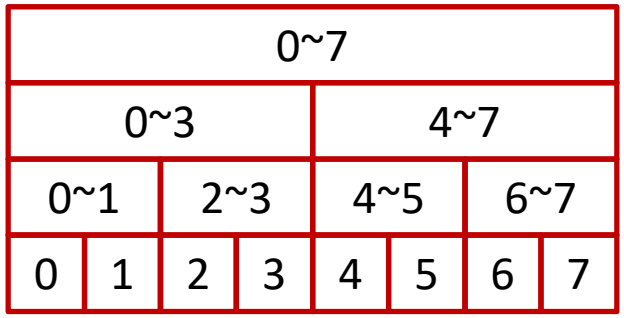
```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v

0~7							
0~3				4~7			
0~1		2~3		4~5		6~7	
0	1	2	3	4	5	6	7

- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

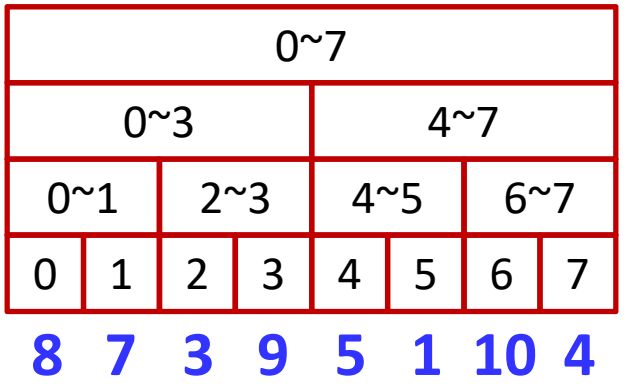
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

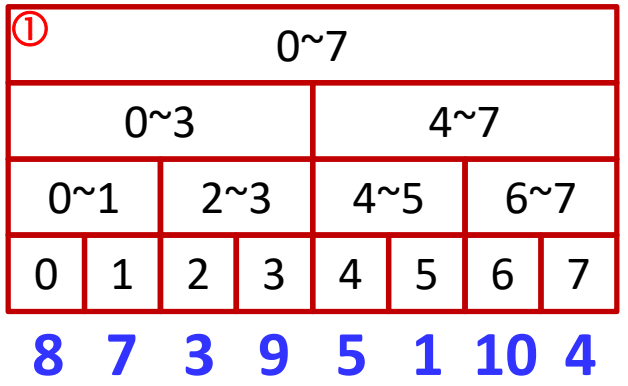
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

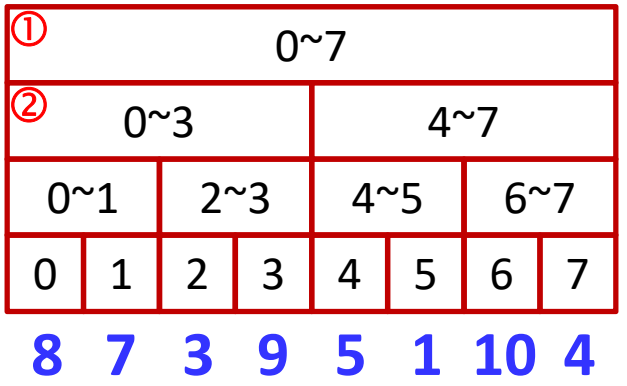
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

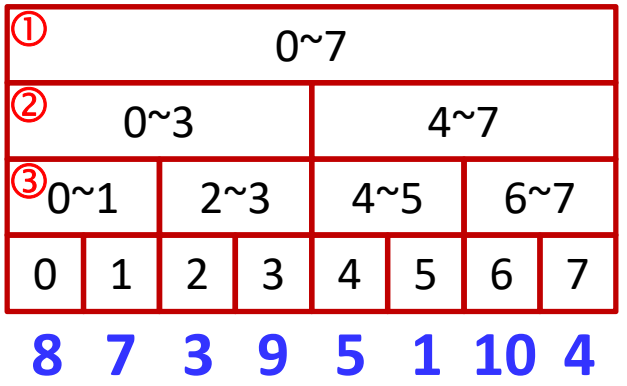
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v

① 0~7							
② 0~3				4~7			
③ 0~1		2~3		4~5		6~7	
④ 0	1	2	3	4	5	6	7
8	7	3	9	5	1	10	4

- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

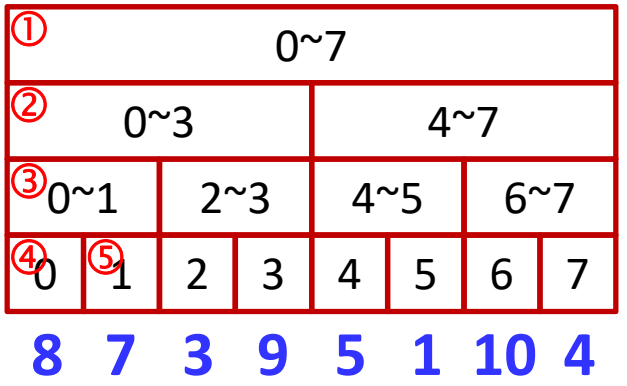
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

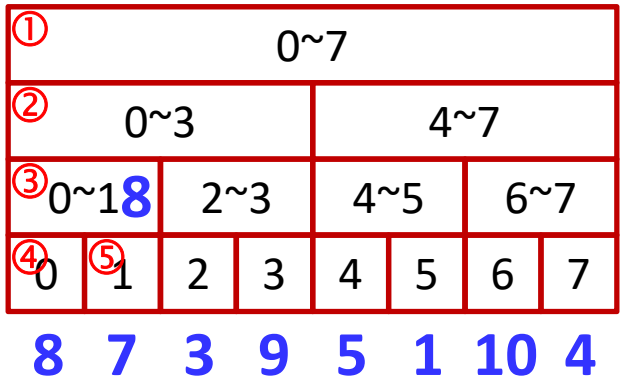
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

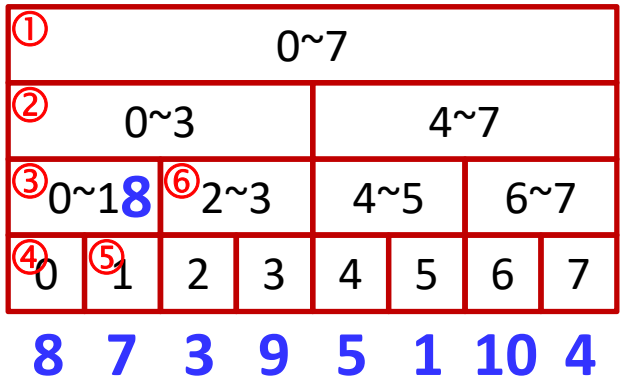
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

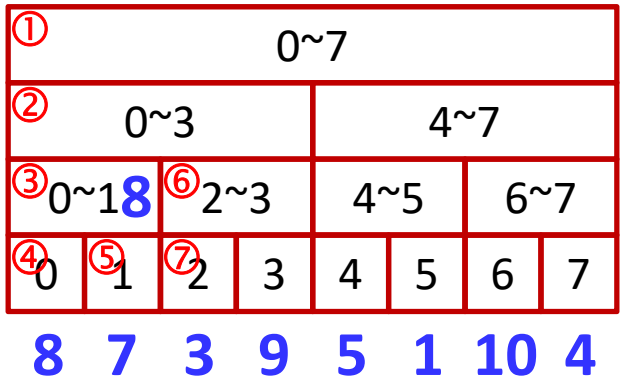
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

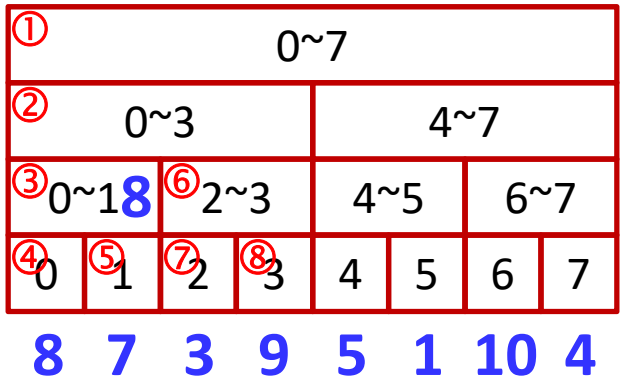
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

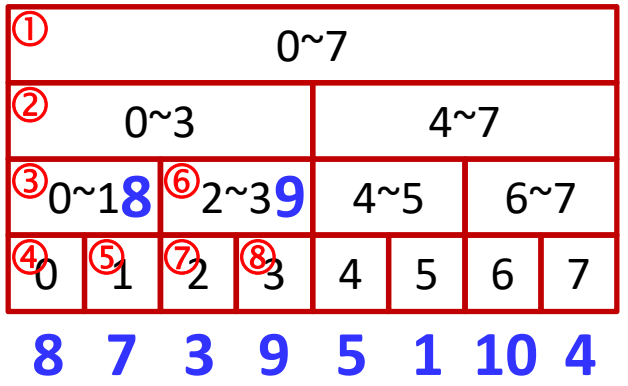
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

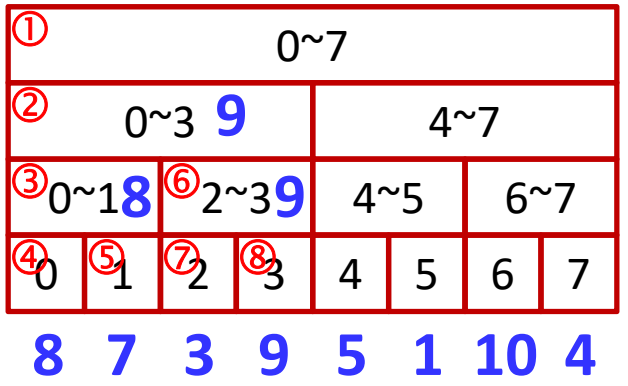
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

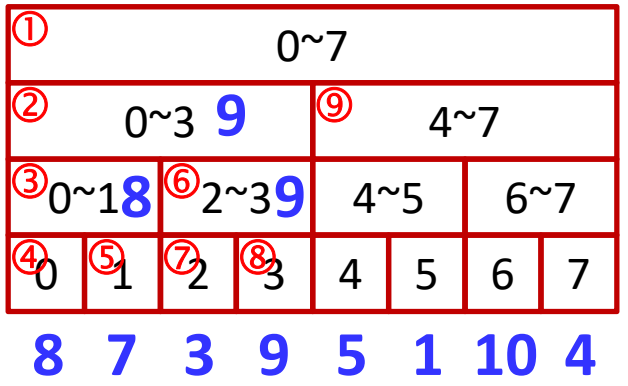
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

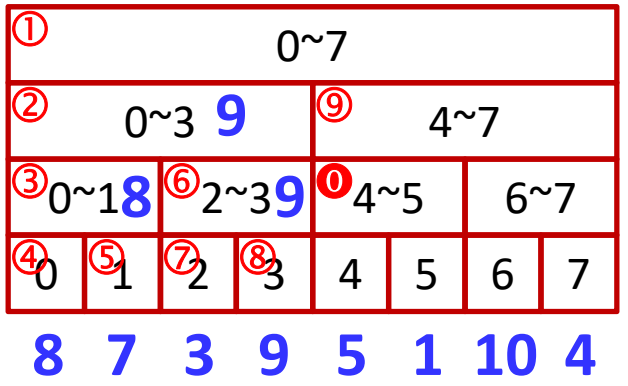
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

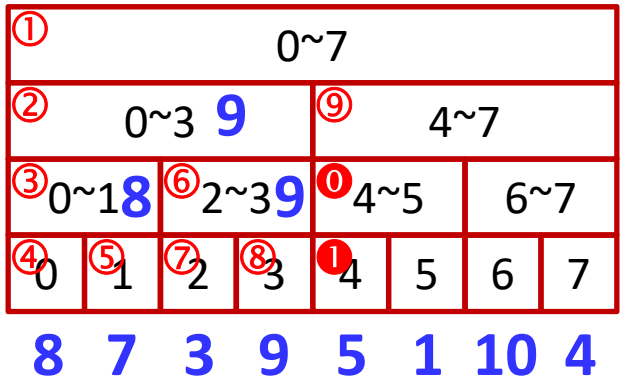
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

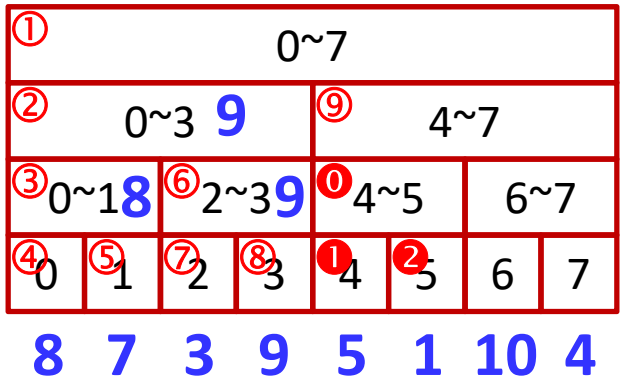
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

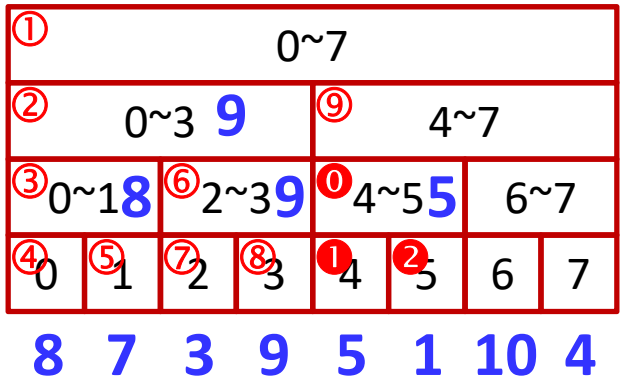
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

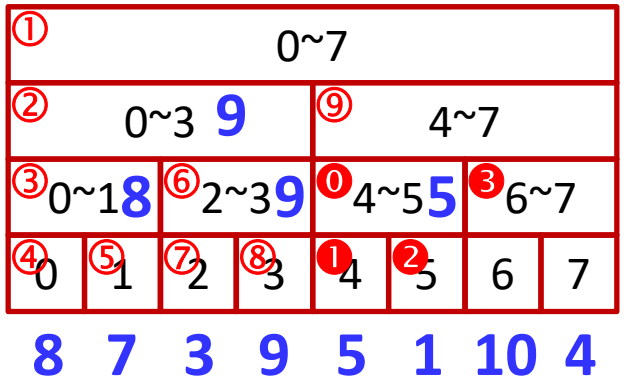
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

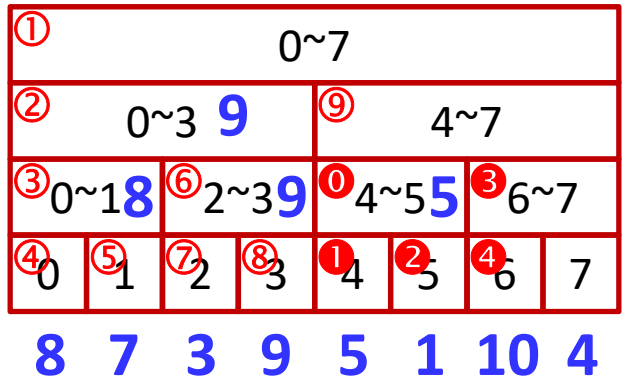
```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

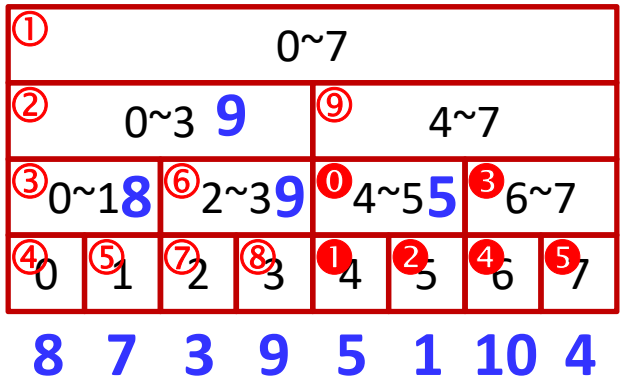
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

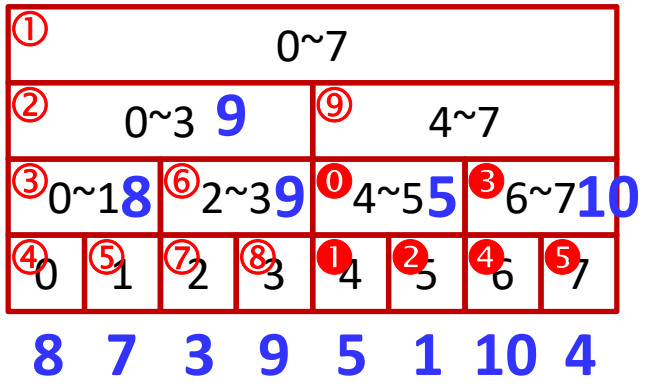
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

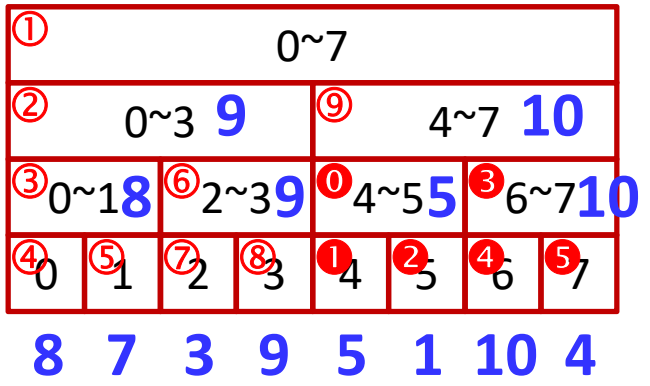
線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array

v



- Recursive **construction**

```
build(1, 0,n-1);
```

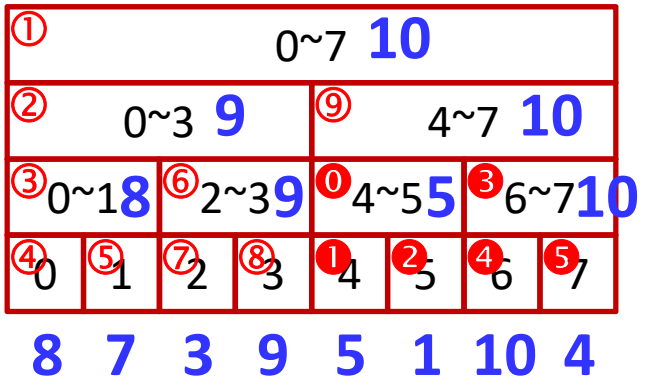
```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

```
build(1, 0,n-1);
```

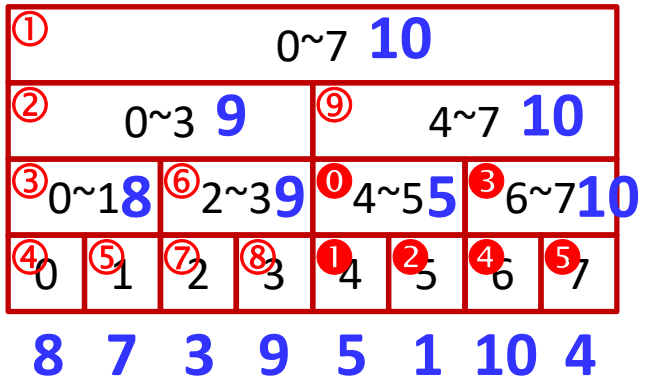
```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
    v[iv]=d[ld];
  else
    build(iv+1, ld,md=(ld+rd)/2),
    build(iv+(md-ld+1)*2, md+1,rd),
    v[iv]=max(v[iv+1],
              v[iv+(md-ld+1)*2]);
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
                  v[iv+(md-ld+1)*2]);
}
```

- Recursive **RMQ query [lq,rq]**

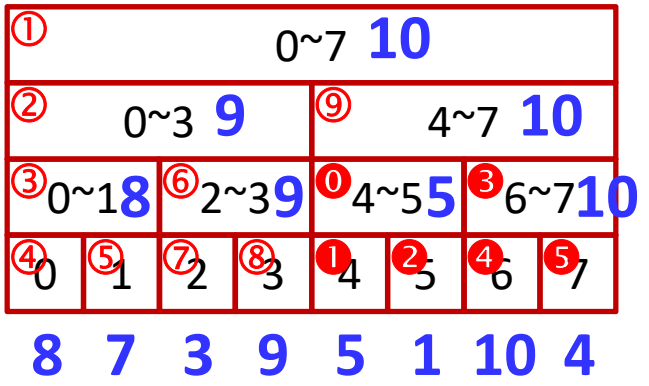
```
int RMQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x800000000;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;
  if (rq<=md) return RMQ(iv+1, ld,md, lq,rq);
  else if (lq>md)
     return RMQ(iv+(md-ld+1)*2, md+1,rd,lq,rq);
  return max(RMQ(iv+1, ld,md, lq,md),
                RMQ(iv+(md-ld+1)*2, md+1,rd,
                                       md+1,rq));
}
```

線段樹的實作 (II)

- **Range Max Query**
  - O(2n) pre-order array for the storage of the full binary tree

```
#define MAXN 500
int d[MAXN], v[2*MAXN];
```

n=8, requires 16-element array



- Recursive **construction**

```
build(1, 0,n-1);
```

```
void build(int iv, int ld, int rd) {
  int md;
  if (ld==rd)
     v[iv]=d[ld];
  else
     build(iv+1, ld,md=(ld+rd)/2),
     build(iv+(md-ld+1)*2, md+1,rd),
     v[iv]=max(v[iv+1],
               v[iv+(md-ld+1)*2]);
}
```

- Recursive **RMQ query [lq,rq]**

```
int RMQ(int iv, int ld,int rd, int lq,int rq) {
  if (lq>rq) return 0x800000000;
  if ((lq==ld)&&(rq==rd)) return v[iv];
  int md=(ld+rd)/2;
  if (rq<=md) return RMQ(iv+1, ld,md, lq,rq);
  else if (lq>md)
     return RMQ(iv+(md-ld+1)*2, md+1,rd,lq,rq);
  return max(RMQ(iv+1, ld,md, lq,md),
             RMQ(iv+(md-ld+1)*2, md+1,rd,
                 md+1,rq));}
```

- test RMQ

- Recursive update

- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

ld<=**t**<=rd

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

ld<=t<=rd

```
update(1, 0, 7, 5, 11);
```

- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

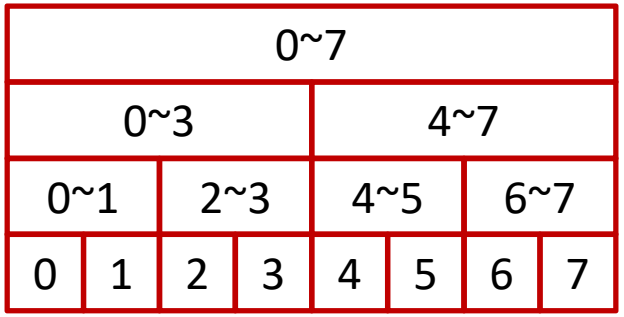
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

v

ld<=t<=rd

update(1, 0, 7, 5, 11);



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

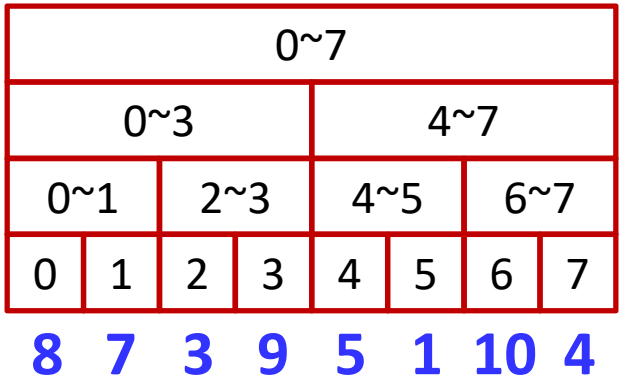
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

ld<=t<=rd

update(1, 0, 7, 5, 11);

v



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

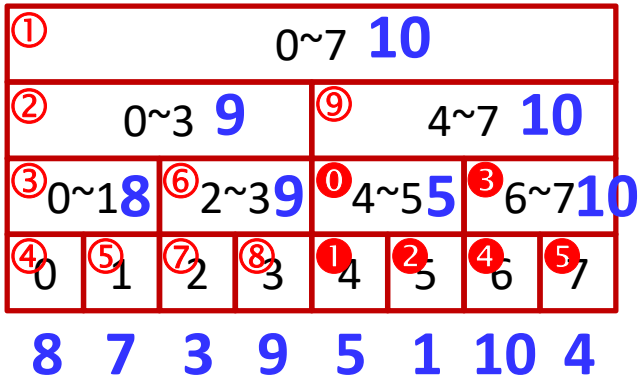
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

ld<=t<=rd

update(1, 0, 7, 5, 11);

v



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

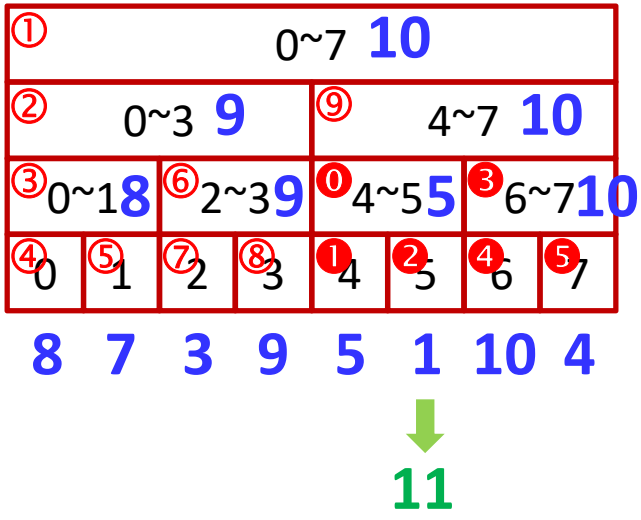
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

v

ld<=t<=rd

update(1, 0, 7, 5, 11);



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

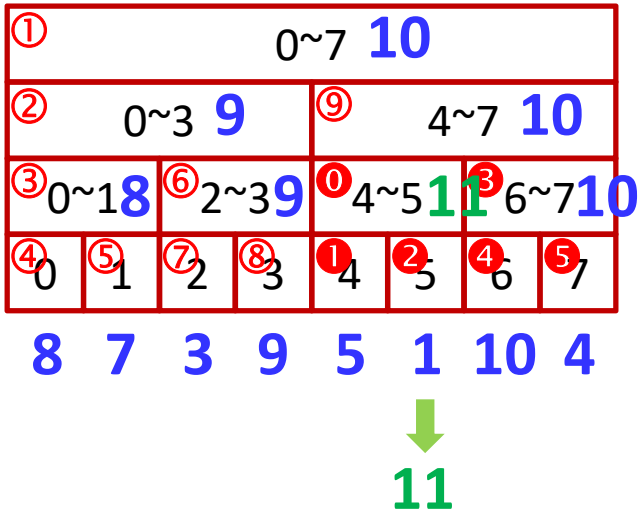
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

v

ld<=t<=rd

update(1, 0, 7, 5, 11);



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

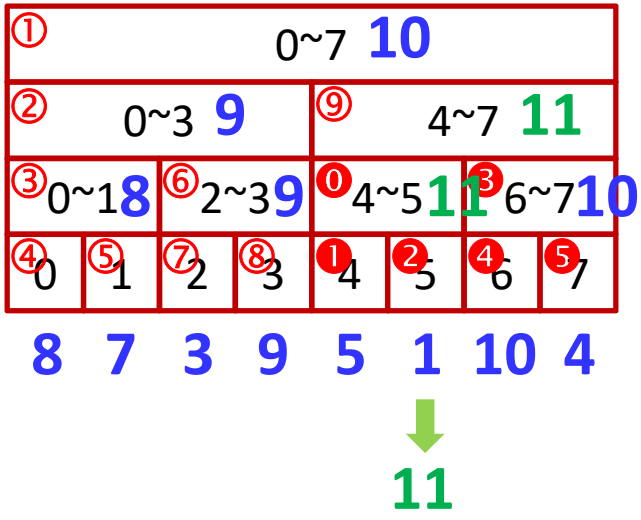
- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

v

ld<=t<=rd

update(1, 0, 7, 5, 11);



- test RMQ

```
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &d[i]);
build(1, 0,n-1);
while (2==scanf("%d%d", &lq, &rq))
    printf("%d\n", RMQ(1, 0,n-1, max(lq,0),min(rq,n-1)));
```

- Recursive update

```
void update(int iv, int ld,int rd, int t,int val) {
    if (ld==rd) // ==t
        v[iv] = val;
    else {
        int md=(ld+rd)/2;
        if (t<=md)
            update(iv+1, ld,md, t,val);
        else
            update(iv+(md-ld+1)*2, md+1,rd, t,val);
        v[iv] = max(v[iv+1], v[iv+(md-ld+1)*2]);
    }
}
```

v

ld<=t<=rd

update(1, 0, 7, 5, 11);