暴力法不給力的時候

不同問題有不同的特殊解法

丁培毅

尋找 Ugly Number

Ugly Number

- 一個正整數如果質因數只有 2 或是 3 或是 5 的時候稱為 Ugly number;一個正整數如果質因數只有 2, 3, 5, 7 時稱為 Humble number

Ugly Number

- 一個正整數如果質因數只有 2 或是 3 或是 5 的時候稱為 Ugly number;一個正整數如果質因數只有 2, 3, 5, 7 時稱為 Humble number
- 也就是一個 Ugly number 可以寫成 $2^a3^b5^c$ (定義 a=0, b=0, c=0 也是一個 Ugly number)

Ugly Number

- 一個正整數如果質因數只有 2 或是 3 或是 5 的時候稱為 Ugly number;一個正整數如果質因數只有 2, 3, 5, 7 時稱為 Humble number
- 也就是一個 Ugly number 可以寫成 $2^a3^b5^c$ (定義 a=0, b=0, c=0 也是一個 Ugly number)
- 要測試一個正整數是不是 Ugly number 可以把所有 2, 3, 5 的因數都除掉以後看看是不是 1, 例如要看一個數字是不 2 的次方數可以用下面的迴圈把 2 的因數都除掉

Ugly Number

- 一個正整數如果質因數只有 2 或是 3 或是 5 的時候稱為 Ugly number;一個正整數如果質因數只有 2, 3, 5, 7 時稱為 Humble number
- 也就是一個 Ugly number 可以寫成 $2^a3^b5^c$ (定義 a=0, b=0, c=0 也是一個 Ugly number)
- 要測試一個正整數是不是 Ugly number 可以把所有 2, 3, 5 的因數都除掉以後看看是不是 1, 例如要看一個數字是不 2 的次方數可以用下面的迴圈把 2 的因數都除掉

```
while (n%2==0) n /= 2;
if (n==1)
    printf("n is a power of 2\n");
else
    printf("n is not a power of 2\n");
```

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number
- 當然可以先想一下最簡單、最直覺的作法：用一個迴圈測試
  每一個自然數看看是不是 Ugly number

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number
- 當然可以先想一下最簡單、最直覺的作法：用一個迴圈測試
  每一個自然數看看是不是 Ugly number

```
int n, count = 1, i = 2;
scanf("%d", &n);
while (count < n)
    if (isUgly(i++)) count++;
```

```
int isUgly(int n) {
    while (n%2==0)  n /= 2;
    while (n%3==0)  n /= 3;
    while (n%5==0)  n /= 5;
    return n==1;
}
```

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number
- 當然可以先想一下最簡單、最直覺的作法：用一個迴圈測試
  每一個自然數看看是不是 Ugly number

```
int n, count = 1, i = 2;
scanf("%d", &n);
while (count < n)
    if (isUgly(i++)) count++;
```

```
int isUgly(int n) {
    while (n%2==0)  n /= 2;
    while (n%3==0)  n /= 3;
    while (n%5==0)  n /= 5;
    return n==1;
}
```

- 這個作法很快地會發現兩個問題

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number
- 當然可以先想一下最簡單、最直覺的作法：用一個迴圈測試
  每一個自然數看看是不是 Ugly number

```
int n, count = 1, i = 2;
scanf("%d", &n);
while (count < n)
    if (isUgly(i++)) count++;
```

```
int isUgly(int n) {
    while (n%2==0)  n /= 2;
    while (n%3==0)  n /= 3;
    while (n%5==0)  n /= 5;
    return n==1;
}
```

- 這個作法很快地會發現兩個問題
  1. i 的數值比 count 大很多很多, 如果預期的 n 值在 100000,
     那麼 i 一定要宣告成 long long, 否則很快就出現溢位

尋找第 n 個 Ugly Number

- Ugly number 依序排列是 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, …
  請寫一個程式尋找第 n 個 Ugly number
- 當然可以先想一下最簡單、最直覺的作法：用一個迴圈測試
  每一個自然數看看是不是 Ugly number

```
int n, count = 1, i = 2;
scanf("%d", &n);
while (count < n)
    if (isUgly(i++)) count++;
```

```
int isUgly(int n) {
    while (n%2==0)  n /= 2;
    while (n%3==0)  n /= 3;
    while (n%5==0)  n /= 5;
    return n==1;
}
```

- 這個作法很快地會發現兩個問題
  1. i 的數值比 count 大很多很多, 如果預期的 n 值在 100000,
     那麼 i 一定要宣告成 long long, 否則很快就出現溢位
  2. n 值變大的時候, 程式執行時間很久

尋找第 n 個 Ugly Number

- 能不能加快尋找的速度?

尋找第 n 個 Ugly Number

- 能不能加快尋找的速度?
  - 執行前面這個程式時看到的問題是自然數中不是 Ugly number 的數字太多了, 每個自然數都測試太慢了

尋找第 n 個 Ugly Number

- 能不能加快尋找的速度?
  - 執行前面這個程式時看到的問題是自然數中不是 Ugly number 的數字太多了, 每個自然數都測試太慢了
  - 如果可以只列舉出 Ugly number 的話, 應該就快得多

尋找第 n 個 Ugly Number

- 能不能加快尋找的速度?
  - 執行前面這個程式時看到的問題是自然數中不是 Ugly number 的數字太多了, 每個自然數都測試太慢了
  - 如果可以只列舉出 Ugly number 的話, 應該就快得多
  - 由Ugly number $x=2^a3^b5^c$ 的定義可以看到一種列出所有 Ugly number 的方法是列舉 (a,b,c) 的數值 (0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,0,2), (0,1,1), (1,0,1), (1,1,0), (0,2,0), (2,0,0), … 不過列舉的順序不太好決定, 列舉出來的 Ugly number 也不見得是由小排到大的

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1

2

3

5

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2

4

3 6

5 10

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3

4 6

6 9

5 10 15

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4

6 8

6 9 12

5 10 15 20

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5

6 8 10

6 9 12 15

10 15 20 25

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6

8 10 12

6 9 12 15 18

10 15 20 25 30

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6

8 10 12

9 12 15 18

10 15 20 25 30

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8

10 12 16

9 12 15 18 24

10 15 20 25 30 40

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9

10 12 16 18

12 15 18 24 27

10 15 20 25 30 40 45

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10

12 16 18 20

12 15 18 24 27 30

10 15 20 25 30 40 45 50

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10

12 16 18 20

12 15 18 24 27 30

15 20 25 30 40 45 50

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10 12

16 18 20 24

12 15 18 24 27 30 36

15 20 25 30 40 45 50 60

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10 12

16 18 20 24

15 18 24 27 30 36

15 20 25 30 40 45 50 60

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10 12 15

16 18 20 24 30

18 24 27 30 36 45

15 20 25 30 40 45 50 60 75

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10 12 15

16 18 20 24 30

18 24 27 30 36 45

20 25 30 40 45 50 60 75

尋找第 n 個 Ugly Number

- 加快尋找的速度?
  - 另一種看法是每一個 Ugly number 都定義出三個大於它的 Ugly number: {2x, 3x, 5x}, 這三個數字雖然不見得是緊接著 x 的三個 Ugly number, 也不見得是連續的三個 Ugly number, 但是這些集合的聯集的確是所有的 Ugly number {2,3,5}, {4,6,10}, {6,9,15}, {8,12,20}, {10,15,25}, {12,18, 30}, {16,24,40}, …　　注意: **1.** 順序不對 **2.** 有重複

1 2 3 4 5 6 8 9 10 12 15 16 …

18 20 24 30 32 …

18 24 27 30 36 45 48 …

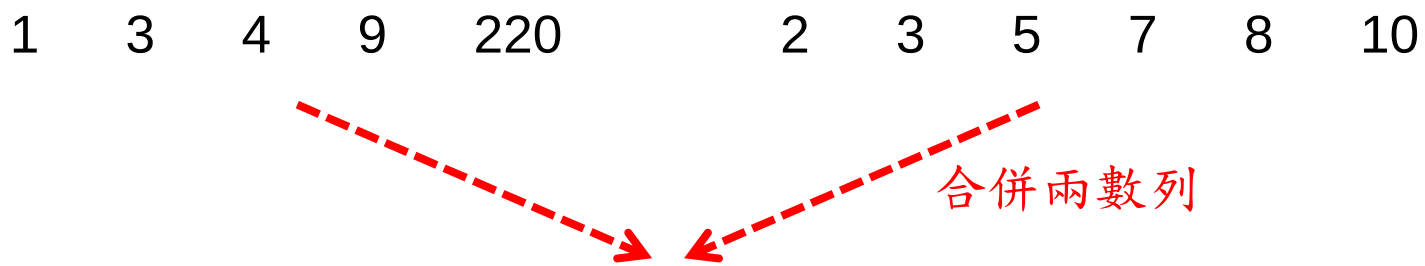
20 25 30 40 45 50 60 75 80 …

合併兩個數列

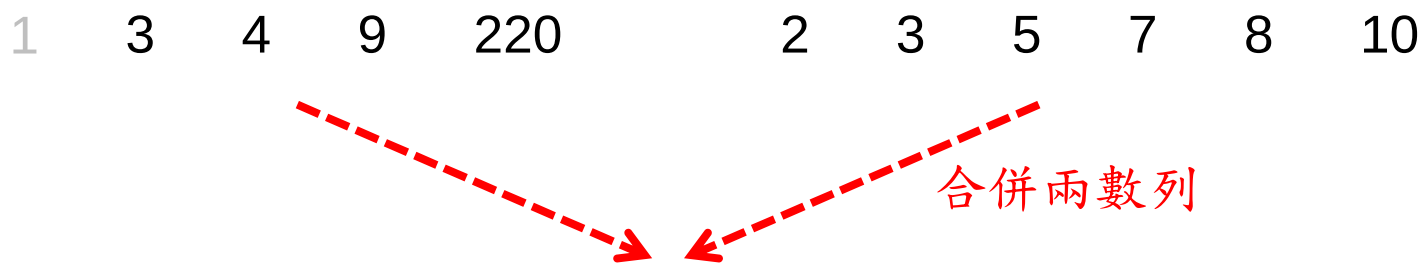
合併兩個數列

1　3　4　9　220　　2　3　5　7　8　10

合併兩個數列

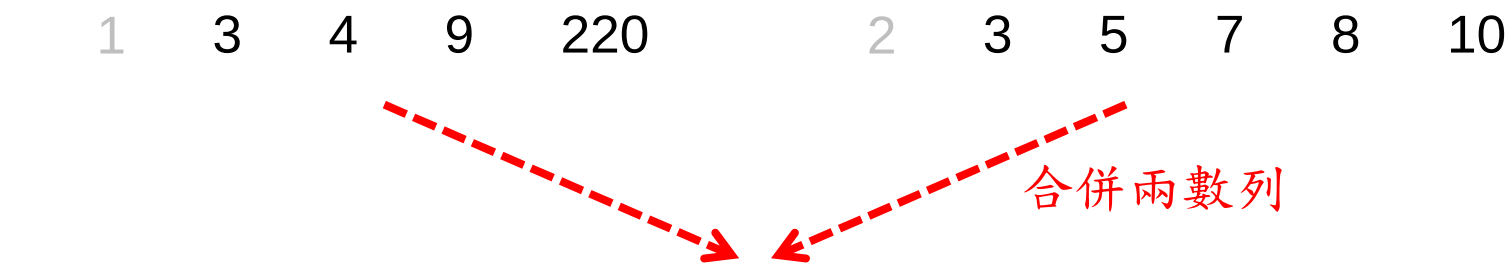


合併兩個數列



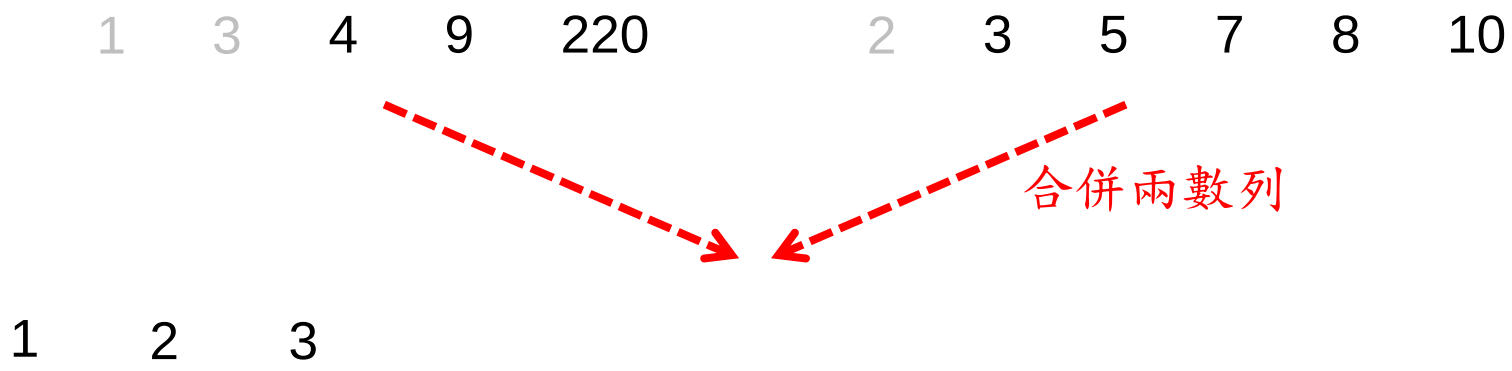
1

合併兩個數列

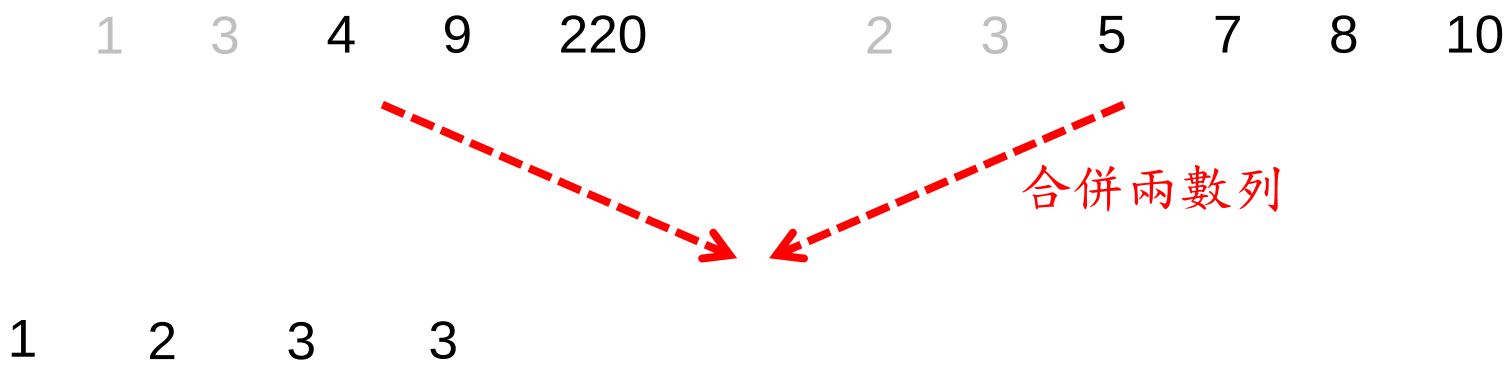


1 2

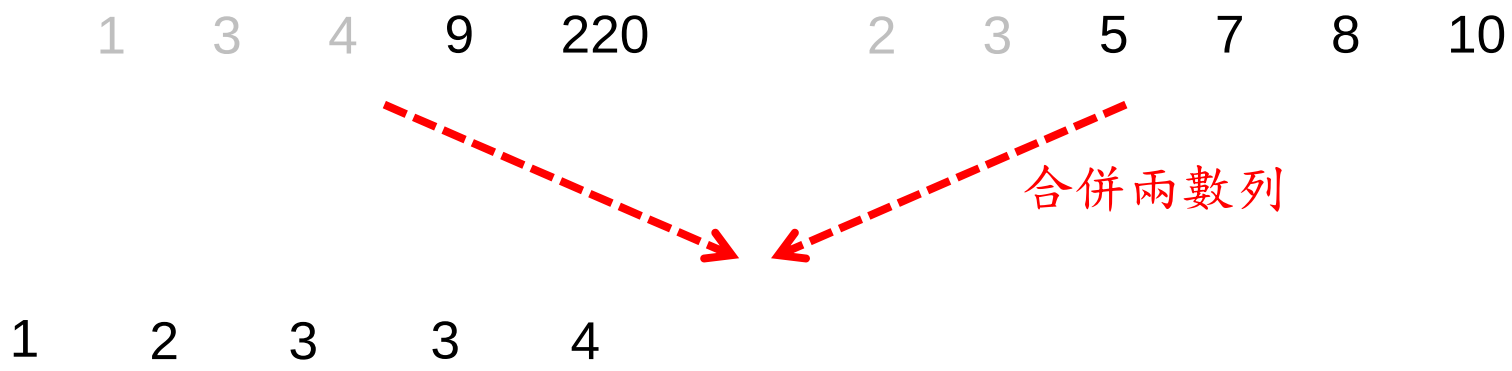
合併兩個數列



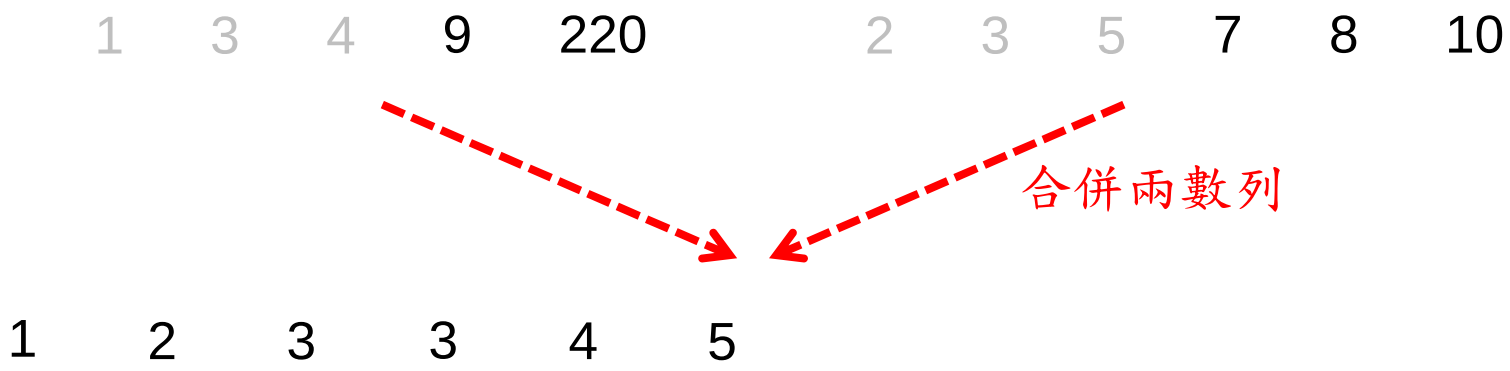
合併兩個數列



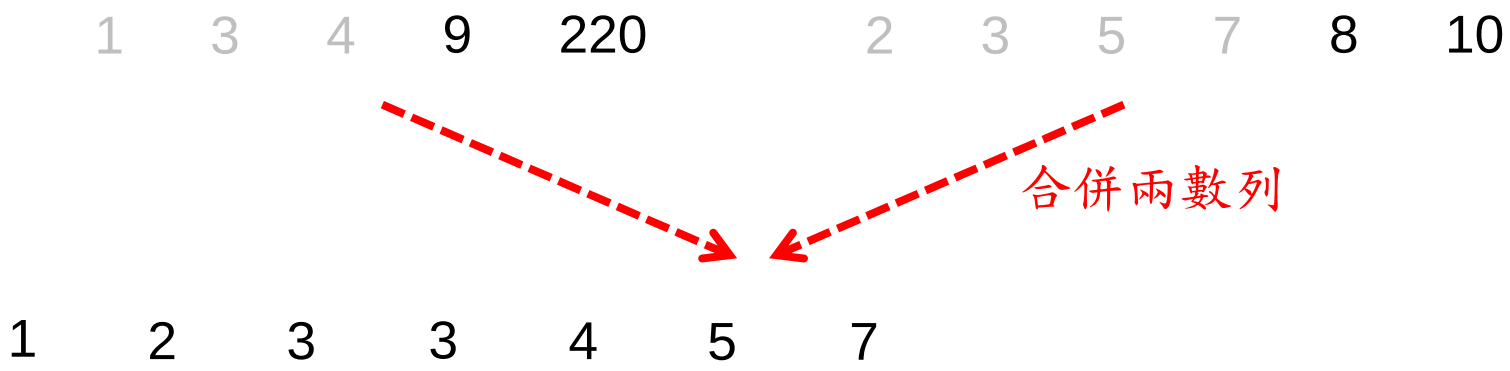
合併兩個數列



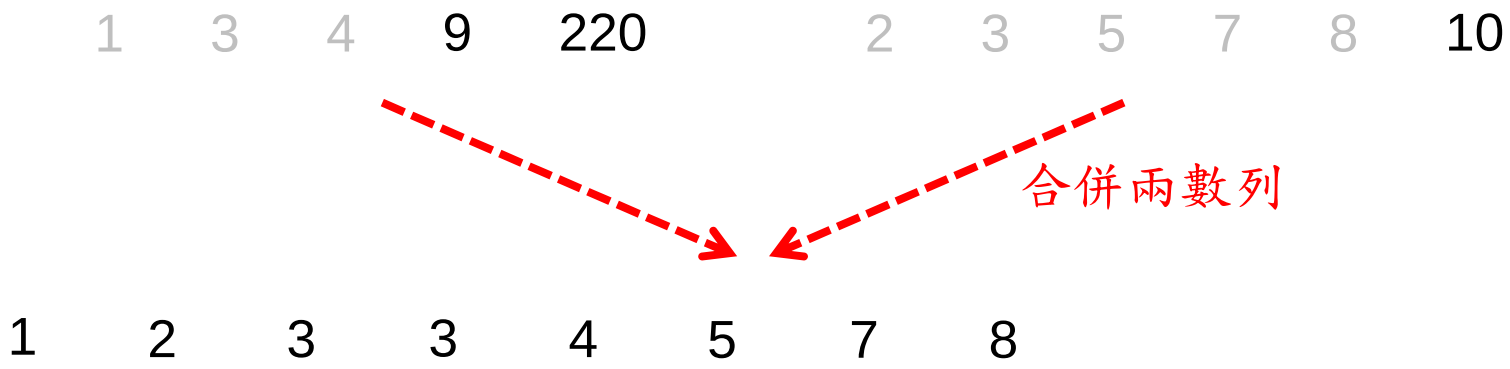
合併兩個數列



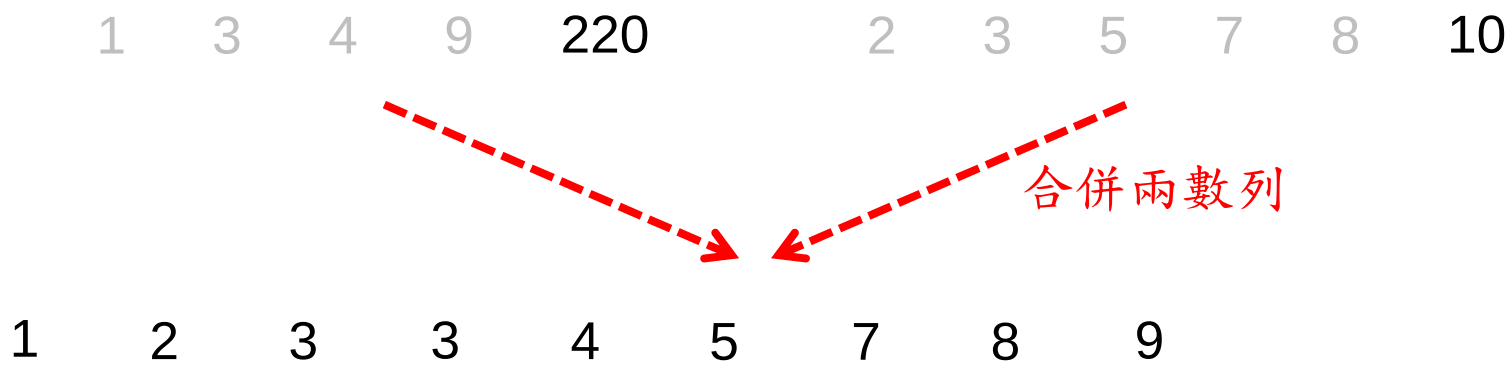
合併兩個數列



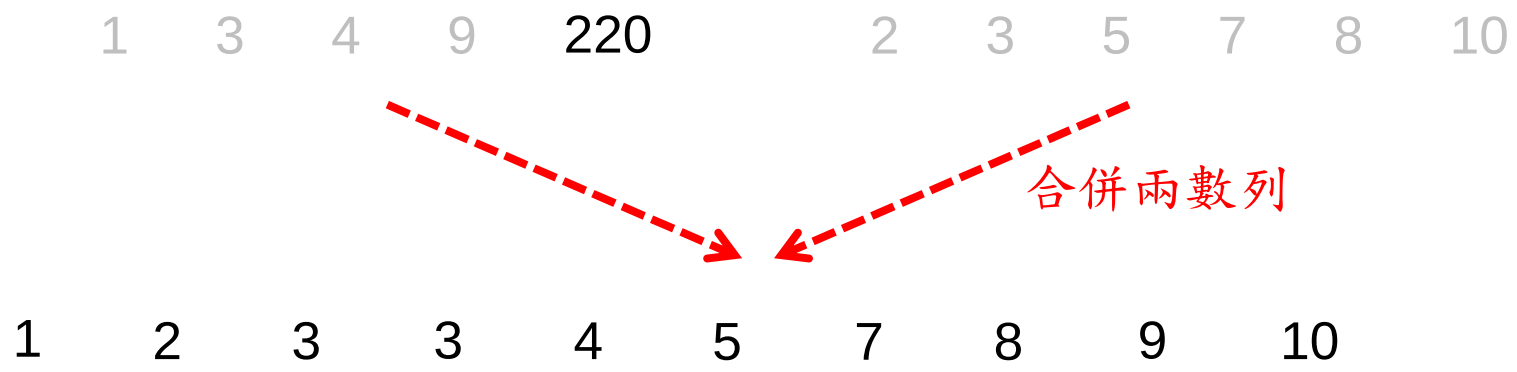
合併兩個數列



合併兩個數列

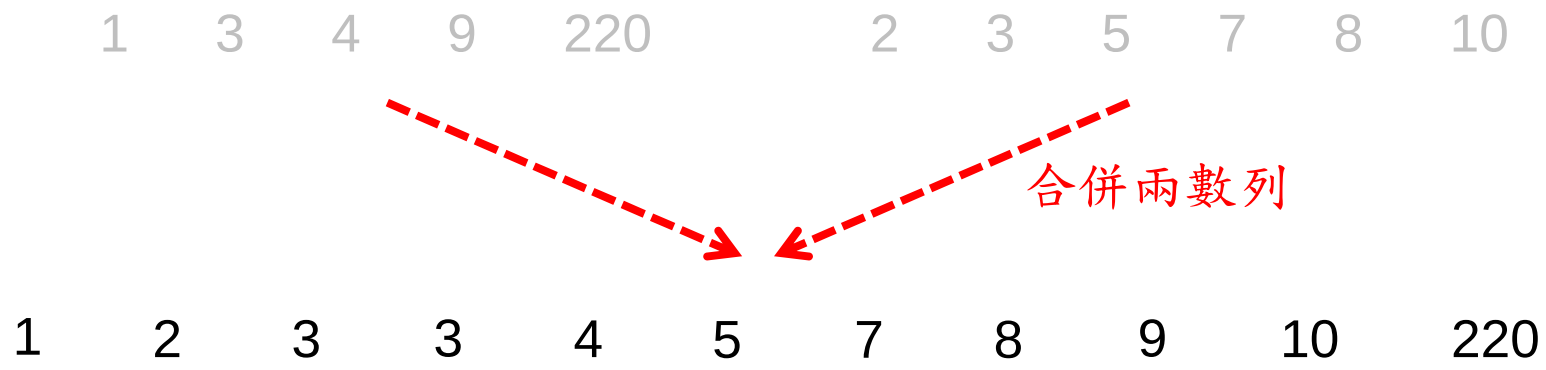


合併兩個數列

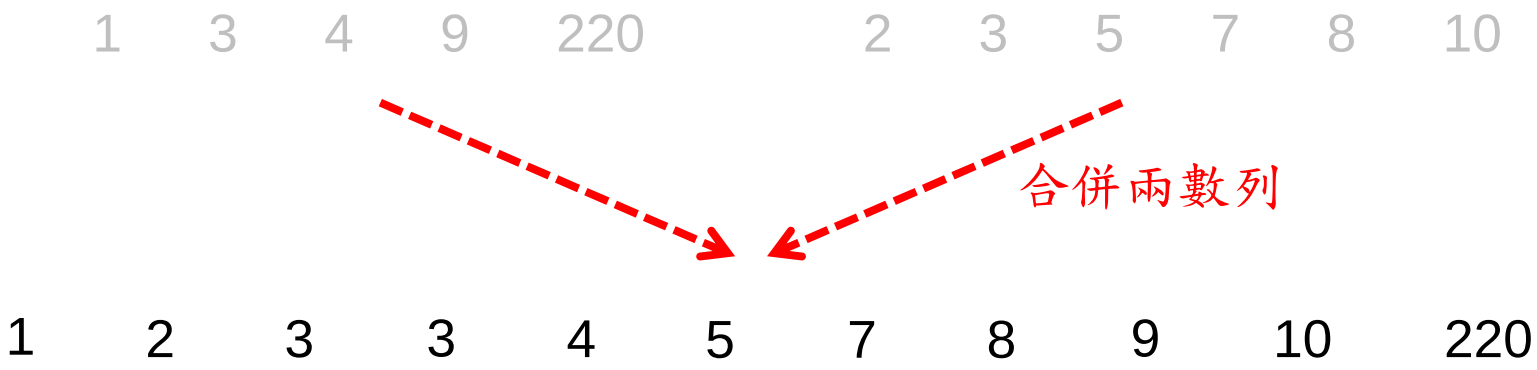


1 2 3 3 4 5 7 8 9 10

合併兩個數列



合併兩個數列




```
void merge(int data1[], int len1, int data2[], int len2, int rst[]) {
    int idx1=0, idx2=0, rst_idx=0;
    while (idx1 < len1 || idx2 < len2) {
        if (idx1 < len1 && idx2 < len2)
            rst[rst_idx++] = (data1[idx1]<=data2[idx2]) ?  data1[idx1++] : data2[idx2++];
        else if (idx1 < len1)
            rst[rst_idx++] = data1[idx1++];
        else
            rst[rst_idx++] = data2[idx2++];
    }
}
```

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了

2 4 6 8 10 12 16 18 20 24 30 32 …

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了

2 4 6 8 10 12 16 18 20 24 30 32 …

3 6 9 12 15 18 24 27 30 36 45 48 …

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了

2 4 6 8 10 12 16 18 20 24 30 32 …

3 6 9 12 15 18 24 27 30 36 45 48 …

5 10 15 20 25 30 40 45 50 60 75 80 …

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了

2 4 6 8 10 12 16 18 20 24 30 32 …

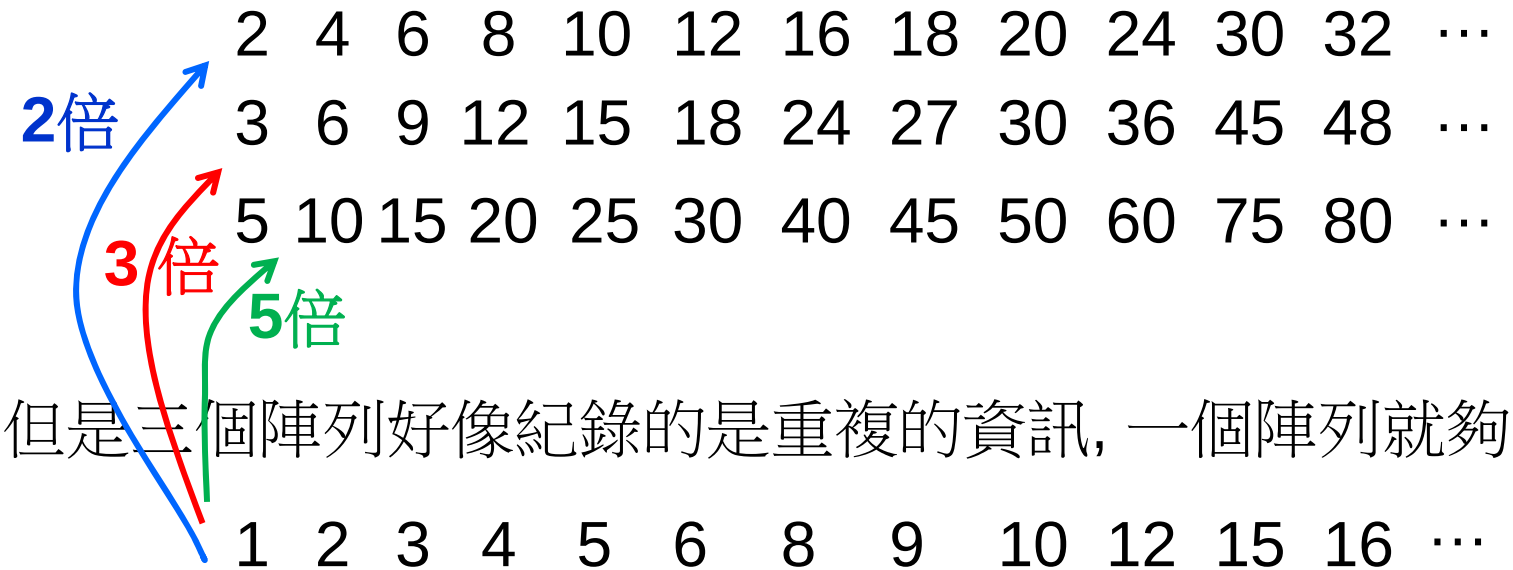
3 6 9 12 15 18 24 27 30 36 45 48 …

5 10 15 20 25 30 40 45 50 60 75 80 …

- 但是三個陣列好像紀錄的是重複的資訊, 一個陣列就夠了

尋找第 n 個 Ugly Number

- 需要以陣列紀錄接下來有哪些可能性, 由其中找到下一個Ugly number; 因為若 a<b 則 2a<2b, 3a<3b, 5a<5b, 所以讓這些數字排成三隊, 每次只需要考慮排在最前面的三個數字就可以了



- 但是三個陣列好像紀錄的是重複的資訊, 一個陣列就夠了

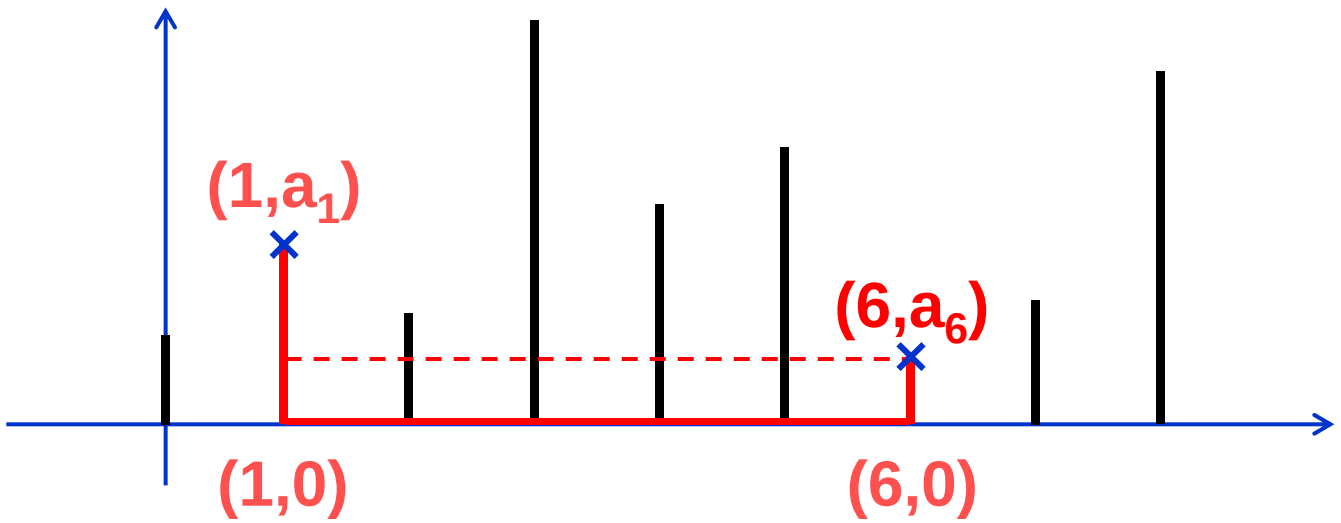
1 2 3 4 5 6 8 9 10 12 15 16 …

就是所有小於 n 的 Ugly number 的陣列

尋找第 n 個 Ugly Number

```
int n, p2=1, p3=1, p5=1;
long long int i=1, x2, x3, x5;
long long int a[10001] = {0,1};
scanf("%d", &n);
x2 = a[p2]*2, x3 = a[p3]*3, x5 = a[p5]*5;
while (++i<=n) {
    if (x2<x3&&x2<x5) {
        a[i] = x2; x2 = a[++p2]*2;
        if (x2==x3) x2 = a[++p2]*2;
        if (x2==x5) x2 = a[++p2]*2;
    }
    else if (x3<x2&&x3<x5) {
        a[i] = x3; x3 = a[++p3]*3;
        if (x3==x2) x3 = a[++p3]*3;
        if (x3==x5) x3 = a[++p3]*3;
    }
    else if (x5<x2&&x5<x3) {
        a[i] = x5; x5 = a[++p5]*5;
        if (x5==x2) x5 = a[++p5]*5;
        if (x5==x3) x5 = a[++p5]*5;
    }
}
printf("%lld\n", a[n]);
```

最大容積的容器



- 問題描述: 有 n 個非負整數 $\{a_i : 0 \leq i \leq n-1\}$, $1000 \leq n \leq 100000$, $0 \leq a_i \leq 32767$, 這些數字描述出 n 條與 x-軸 垂直的線段, 線段的端點為 (i,0) 與 (i, a_i), 任意兩條線段 (i, a_i) 與 (j,a_j) 加上 x-軸 可以看成是一個盛水的容器, 可以盛水的容積為 $| j-i | * \min(a_i, a_j)$, 假設 n 個整數已經在一個整數陣列 a[] 裡面 (a[0] 的資料是 a_0, a[1] 的資料是 a_1, …) , 在下列要求下撰寫程式片段找出能夠圍出最大容積的兩個線段, 以及最大容積

完整列舉

- 既然每一對 (i,j), 0≤i,j≤n-1, 1000≤n≤100000, 都可以定義出一個容器, 容積為 $|\ j-i\ |*\min(a_i,a_j)$, 反正計算機的運算速度很快, 最直接的方法就是把每一對 (i,j) 都列舉出來, 計算出容積, 並且尋找最大值

完整列舉

- 既然每一對 (i,j), 0≤i,j≤n-1, 1000≤n≤100000, 都可以定義出一個容器, 容積為 | j-i |*min(a_i,a_j), 反正計算機的運算速度很快, 最直接的方法就是把每一對 (i,j) 都列舉出來, 計算出容積, 並且尋找最大值

```
int max, maxi, maxj, i, j, tmp;
for (max=i=0; i<n; i++)
    for (j=i; j<n; j++)
        if ((tmp=a[a[i]<=a[j]?i:j]*(j-i))>max)
            max=tmp, maxi=i, maxj=j;
printf("a[%d]:a[%d]=%d:%d (最大容積為%d)\n",
        maxi, maxj, a[maxi], a[maxj], max);
```

完整列舉

- 既然每一對 (i,j), 0≤i,j≤n-1, 1000≤n≤100000, 都可以定義出一個容器, 容積為 $|j-i|*\min(a_i,a_j)$, 反正計算機的運算速度很快, 最直接的方法就是把每一對 (i,j) 都列舉出來, 計算出容積, 並且尋找最大值

```
int max, maxi, maxj, i, j, tmp;
for (max=i=0; i<n; i++)
    for (j=i; j<n; j++)
        if ((tmp=a[a[i]<=a[j]?i:j]*(j-i))>max)
            max=tmp, maxi=i, maxj=j;
printf("a[%d]:a[%d]=%d:%d (最大容積為%d)\n",
        maxi, maxj, a[maxi], a[maxj], max);
```

- 像這樣的程式, 迴圈內部計算與比對的動作需要執行 n(n+1)/2 次, 整體運算時間就是正比於 n^2, 當 n 的數值變得比較大的時候, 就發現程式執行得很緩慢, 我們希望仔細分析一下看能不能有比較快速的方法

如何加快執行速度

- 第一個加速的方向是迴圈內部的運算, 有沒有可以不做的, 如果原先有三個動作, 能不能變成兩個, 能不能簡化, 如果處理的時間變成一半, 總共的執行時間就變成一半, 不過目前迴圈內執行的動作很少, 可以著墨的不多, 況且其實我們更關心程式在 n 變大時的表現, n 變成 10 倍時, 運算量變成 10 倍還是 100 倍影響是比較大的

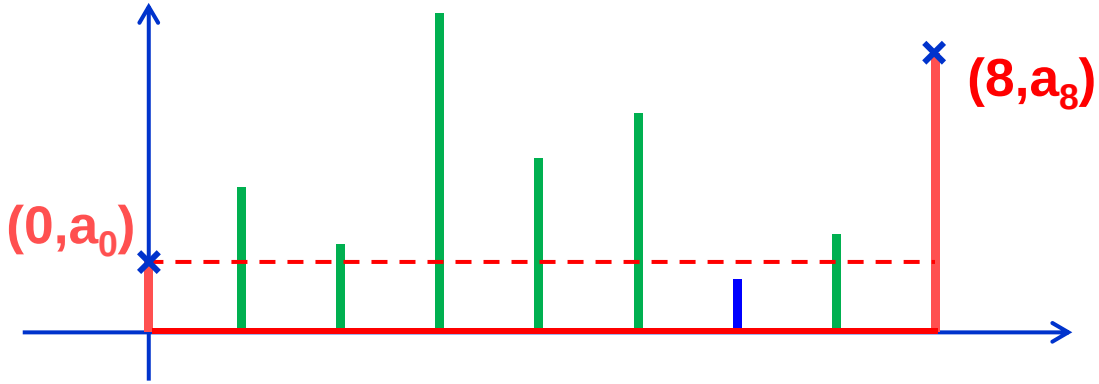
如何加快執行速度

- 第一個加速的方向是迴圈內部的運算, 有沒有可以不做的, 如果原先有三個動作, 能不能變成兩個, 能不能簡化, 如果處理的時間變成一半, 總共的執行時間就變成一半, 不過目前迴圈內執行的動作很少, 可以著墨的不多, 況且其實我們更關心程式在 n 變大時的表現, n 變成 10 倍時, 運算量變成 10 倍還是 100 倍影響是比較大的

- 我們希望找的方法的運算時間最好是正比於 n log n, 正比於 n, 或是正比於 log n, 想要找到比正比於 n^2 快的演算法, 需要仔細分析看看是不是有哪些 (a_i,a_j) 是不需要考量可以跳過去的

加快執行速度

- 考慮 (a_0,a_8), $a_0<a_8$



容積是為 $8*\min(a_0,a_8)=8a_0$

考量 (a_0,a_7), 容積是為 $7*\min(a_0,a_7)=\mathbf{7}a_0$

考量 (a_0,a_6), 容積是為 $6*\min(a_0,a_6)=6\mathbf{a_6}$

考量 (a_0,a_5), 容積是為 $5*\min(a_0,a_5)=\mathbf{5}a_0$

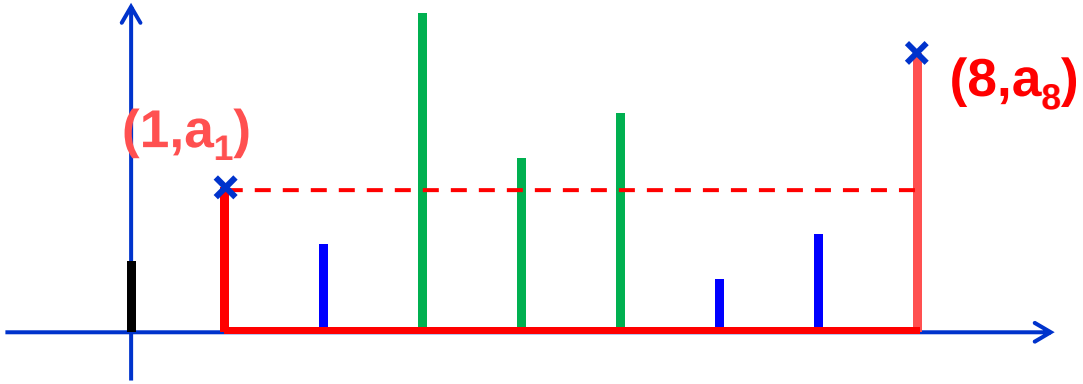
考量 (a_0,a_4), 容積是為 $4*\min(a_0,a_4)=\mathbf{4}a_0$

…

考量 (a_0,a_1), 容積是為 $1*\min(a_0,a_1)=\mathbf{1}a_0$

底變小或是高也同時變小, 看過 (a_0,a_8) 以後 (a_0,a_j), $0<j<8$ 都可以跳過不考慮, 也就是接下去考量 (a_1,a_8) 就足夠了

- 考慮 (a_1,a_8), $a_1<a_8$



容積是為 $7*\min(a_1,a_8)=7a_1$, 需要與剛才的 $8a_0$ 比較大小

考量 (a_1,a_7), 容積是為 $6*\min(a_1,a_7)=6\mathbf{a_7}$

考量 (a_1,a_6), 容積是為 $5*\min(a_1,a_6)=5\mathbf{a_6}$

考量 (a_1,a_5), 容積是為 $4*\min(a_1,a_5)=\mathbf{4}a_1$

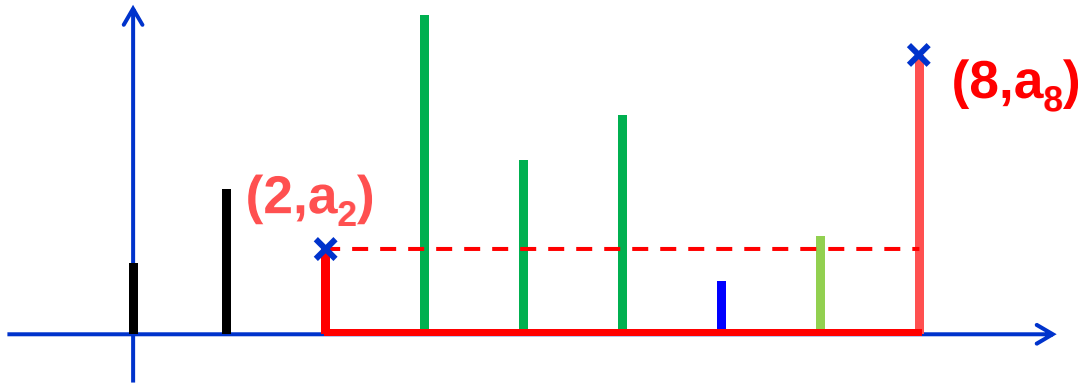
考量 (a_1,a_4), 容積是為 $3*\min(a_1,a_4)=\mathbf{3}a_1$

考量 (a_1,a_3), 容積是為 $2*\min(a_1,a_3)=\mathbf{2}a_1$

考量 (a_1,a_2), 容積是為 $1*\min(a_1,a_2)=\mathbf{a_2}$

底變小或是高也同時變小, 看過 (a_1,a_8) 以後 (a_1,a_j), $1<j<8$ 都可以跳過不考慮, 也就是接下去考量 (a_2,a_8) 就足夠了

- 考慮 (a_2,a_8), $a_2<a_8$



容積是為 $6*\min(a_2,a_8)=6a_2$, 需要與前兩個步驟的最大值比較

考量 (a_2,a_7), 容積是為 $5*\min(a_2,a_7)=\mathbf{5}a_1$

考量 (a_2,a_6), 容積是為 $4*\min(a_2,a_6)=4\mathbf{a_6}$

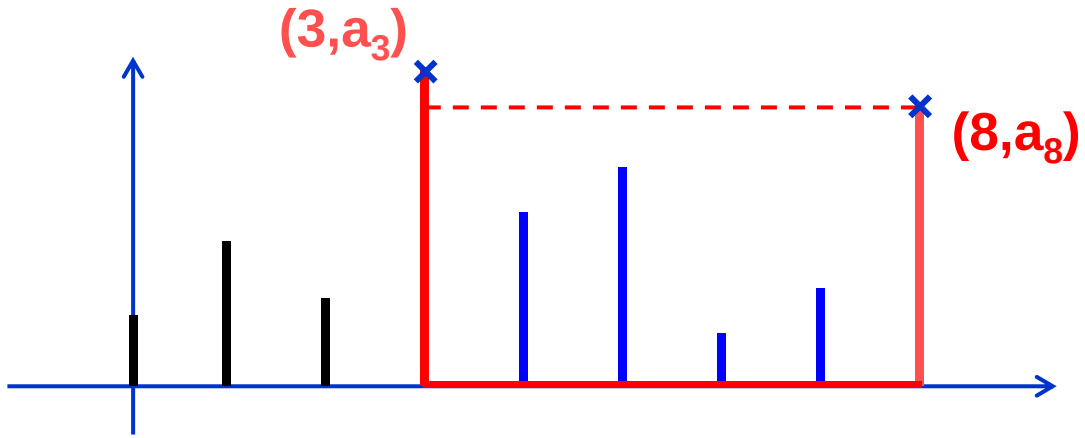
考量 (a_2,a_5), 容積是為 $3*\min(a_2,a_5)=\mathbf{3}a_1$

考量 (a_2,a_4), 容積是為 $2*\min(a_2,a_4)=\mathbf{2}a_1$

考量 (a_2,a_3), 容積是為 $1*\min(a_2,a_3)=\mathbf{1}a_1$

底變小或是高也同時變小, 看過 (a_2,a_8) 以後 (a_2,a_j),$2<j<8$ 都可以跳過不考慮, 也就是接下去考量 (a_3,a_8) 就足夠了

- 考慮 (a_3,a_8), $a_3>a_8$

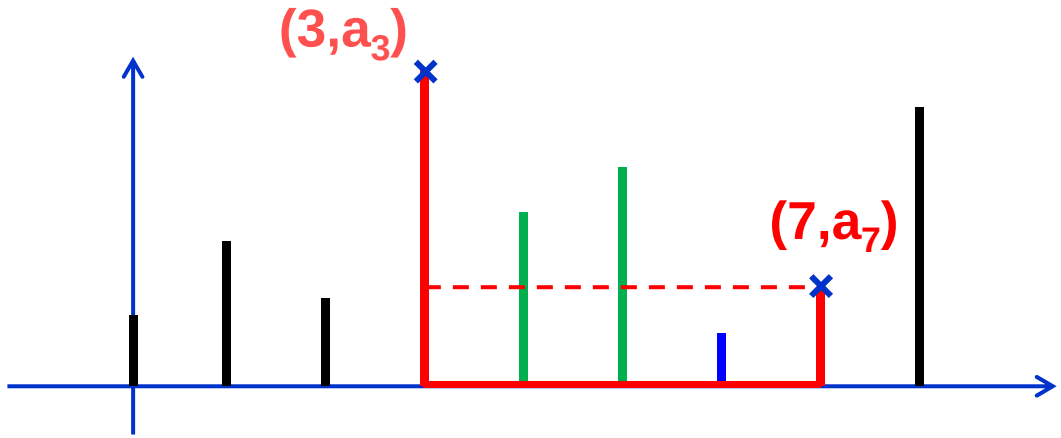


容積是為 $5*\min(a_3,a_8)=5a_8$, 需要與前三個步驟的最大值比較

考量 (a_4,a_8) 容積是為 $4*\min(a_4,a_8)=4\mathbf{a_4}$
考量 (a_5,a_8) 容積是為 $3*\min(a_5,a_8)=3\mathbf{a_5}$
考量 (a_6,a_8) 容積是為 $2*\min(a_6,a_8)=2\mathbf{a_6}$
考量 (a_7,a_8) 容積是為 $1*\min(a_7,a_8)=1\mathbf{a_7}$

底變小或是高也同時變小, 看過 (a_3,a_8) 以後 (a_i,a_8), $3<i<8$ 都可以跳過不考慮, 也就是接下去考量 (a_3,a_7) 就足夠了

- 考慮 (a_3,a_7), $a_3>a_7$

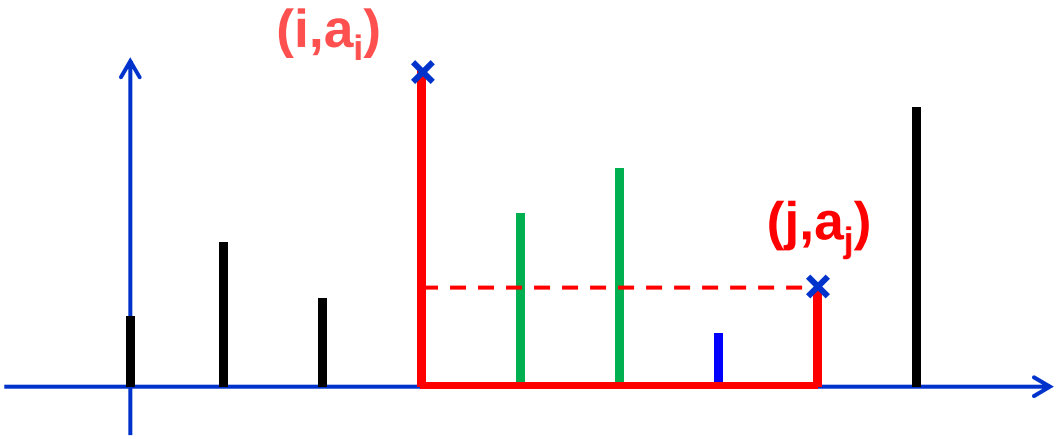


容積是為 $4*\min(a_3,a_7)=4a_7$, 需要與前四個步驟的最大值比較

考量 (a_4,a_7), 容積是為 $3*\min(a_4,a_7)=\mathbf{3}a_7$
考量 (a_5,a_7), 容積是為 $2*\min(a_5,a_7)=\mathbf{2}a_7$
考量 (a_6,a_7), 容積是為 $1*\min(a_6,a_7)=1\mathbf{a_6}$

底變小或是高也同時變小, 看過 (a_3,a_7) 以後 (a_i,a_7), $3<i<7$ 都可以跳過不考慮, 也就是接下去考量 (a_3,a_6) 就足夠了

一般化前面的步驟



- 由 (0, a_0) 及 (n-1, a_{n-1}) 的容積開始考量
- 如果目前考量的是 (i, a_i) 及 (j, a_j)
  - 當 $a_i <= a_j$ 時下一個考量的是 (i+1, a_{i+1}) 及 (j, a_j)
  - 當 $a_i > a_j$ 時下一個考量的是 (i, a_i) 及 (j+1, a_{j+1})
- 直到 i>j 為止

程式實作

```
01   long long tmp, max; int i, j, maxi, maxj;
02   i=0, j=n-1;
03   while (i<=j)
04       if (a[i]<=a[j]) {
05           if ((tmp=(long long)a[i]*(j-i))>max)
06               max=tmp, maxi=i, maxj=j;
07           i++;
08       }
09       else { // a[i]>a[j]
10           if ((tmp=(long long)a[j]*(j-i))>max)
11               max=tmp, maxi=i, maxj=j;
12           j--;
13       }
14   printf("a[%d]:a[%d]=%d:%d (amount of water is %lld)\n",
15        maxi, maxj, a[maxi], a[maxj], max);
```

正確性證明

- 解的空間有 n(n-1)/2 個元素, 一個一個去測試時得到一個運算時間正比於 n^2 的演算法, 前面的演算法沒有把每一個解都測試過, 如果能夠得到正確的答案, 需要滿足一些特別的性質
  基本上演算法是『當 a[i]<=a[j] 時把 i 值加 1, 反之把 j 值減 1』
- 令容積 V(a[i],a[j]) = (j-i) * min(a[i],a[j]), j>=i, 考慮下面兩種狀況
- 當a[i]<=a[j] 時, 演算法把 i 值加1 時, 相當於跳過了 (a[i],a[j-1]), (a[i],a[j-2]), …, (a[i],a[i+1]) 的這些組合, 後續完全不會再檢查這些組合的容積, 實際上因為 a[i]<=a[j], 不論a[j-1]大於、等於或是小於a[i], 都可以推論出V(a[i],a[j-1]) < V(a[i],a[j]), 同理可以推論出 V(a[i],a[j-2]) < V(a[i],a[j]), …, 以及 V(a[i],a[i+1]) < V(a[i],a[j]), 所以跳過這些組合是絕對安全的, 不會影響到結果的正確性; 至於 (a[i],a[j+1]), (a[i],a[j+2]), …, (a[i],a[n-1]), 這些組合不是在先前的步驟中考慮過, 就是在先前的步驟中被跳過了
- 當a[i]>a[j] 時, 演算法把 j 值減 1 時情況也完全一樣

統計長方圖中的最大矩形
-- 左右掃描法

由 $O(n^3)$ 到 $O(n)$

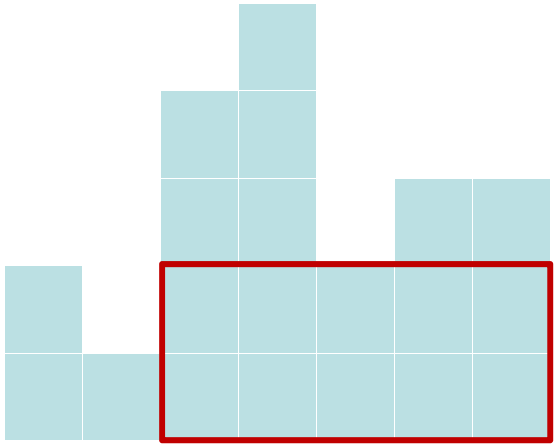
- ZOJ 1985 Largest Rectangle in a Histogram

由 $O(n^3)$ 到 $O(n)$

- ZOJ 1985 Largest Rectangle in a Histogram
  - 例: 2 1 4 5 1 3 3

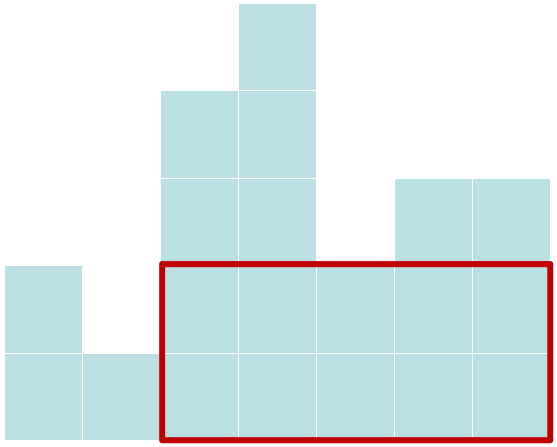
由 $O(n^3)$ 到 $O(n)$

- ZOJ 1985 Largest Rectangle in a Histogram
  - 例: 2 1 4 5 1 3 3



由 $O(n^3)$ 到 $O(n)$

- ZOJ 1985 Largest Rectangle in a Histogram
  - 例: 2 1 4 5 1 3 3



- 基本的搜尋方法運算時間是 $O(n^3)$, 綜合運用資料表示方式以及額外記憶體來減少搜尋的時間, 目標是將運算時間降為 $O(n)$

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

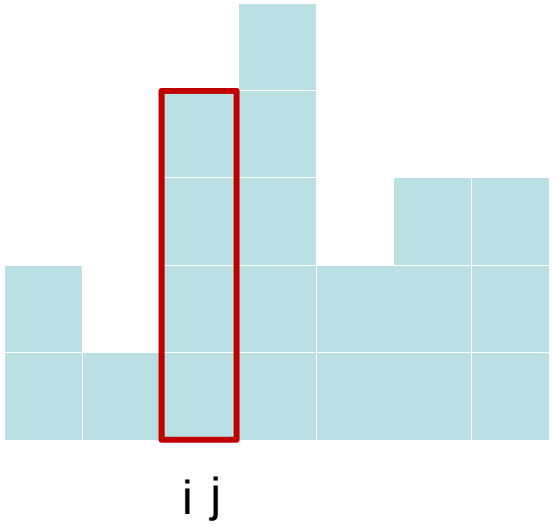
每一對 (i, j), i≤j, 定義一個長方形


```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形

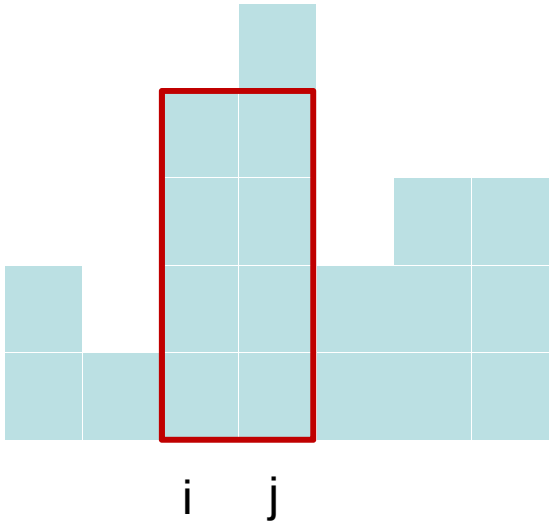



```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形

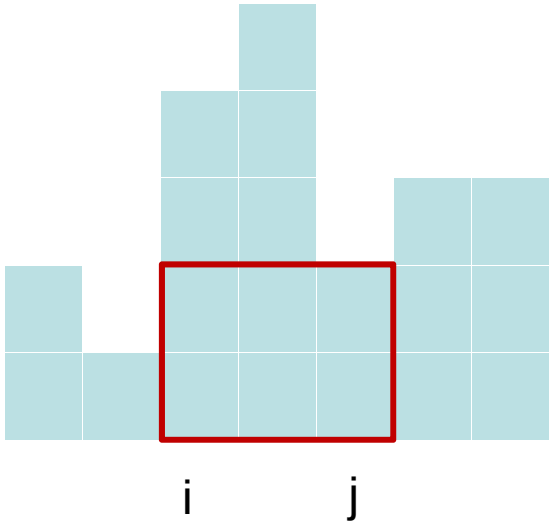



```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形

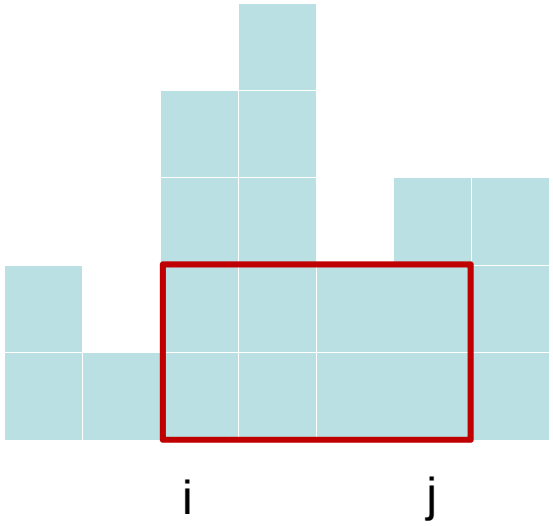



```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形

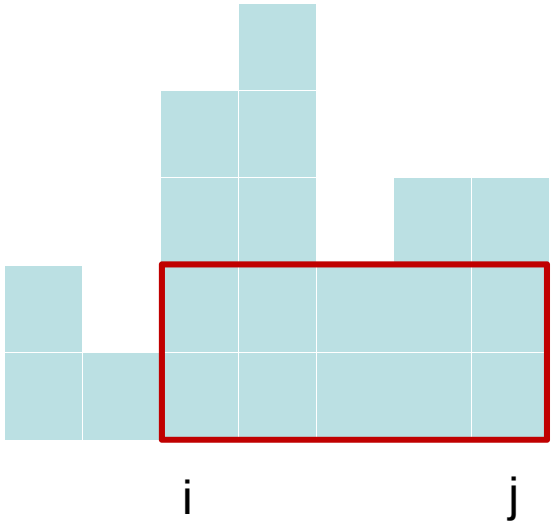



```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形




```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^3)$ 直接列舉法

- 任意兩點可以作為邊界共 n(n+1)/2 種, 其最大高度可以掃描一次中間所有的高度(平均 n/2 個)得到

每一對 (i, j), i≤j, 定義一個長方形


```
int maxArea=0, minHeight, i, j, area;
for (i=0; i<n; i++)
    for (j=i; j<n; j++) {
        minHeight = data[i];
        for (k=i+1; k<=j; k++)
            if (data[k] < minHeight) minHeight = data[k];
        area = (j-i+1) * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

$O(n^2)$： 去除尋找最小高度時重複的運算

- 前一個方法在尋找最小高度時重複許多的運算

$O(n^2)$： 去除尋找最小高度時重複的運算

- 前一個方法在尋找最小高度時重複許多的運算

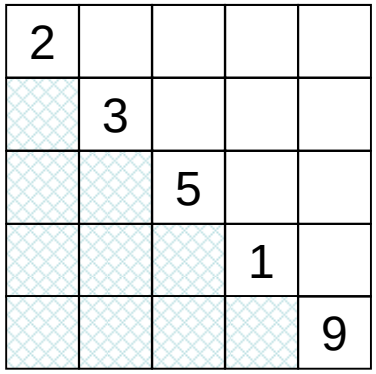
每一對 (i, j), i≤j, 定義一個長方形

```
int maxArea=0, minHeight, i, j, area, w;
for (i=0; i<n; i++) {
    minHeight = INT_MAX; // limits.h
    for (w=1, j=i; j<n; j++, w++) {
        if (data[j] < minHeight) minHeight = data[j];
        area = w * minHeight;
        if (area > maxArea) maxArea = area;
    }
}
```

對一個固定的 i 而言, 配上逐漸增加的 j
可以用 O(1) 的運算量順道累計最小高度

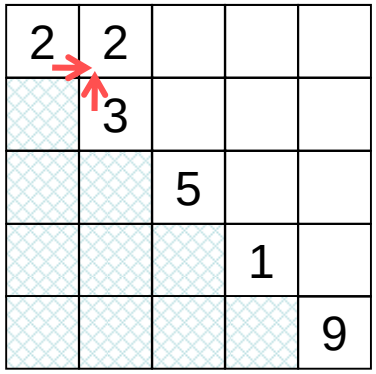
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



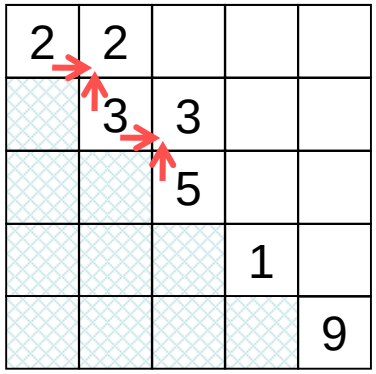
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



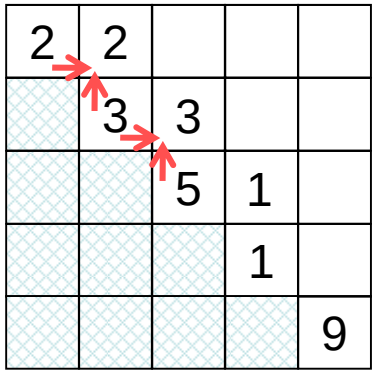
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



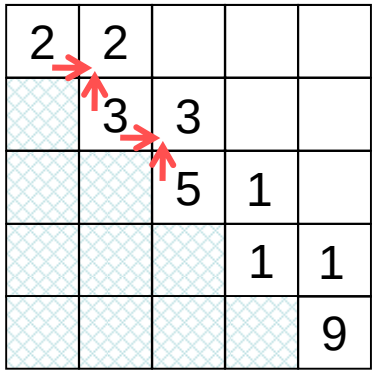
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



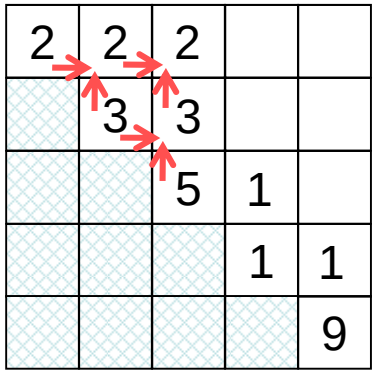
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



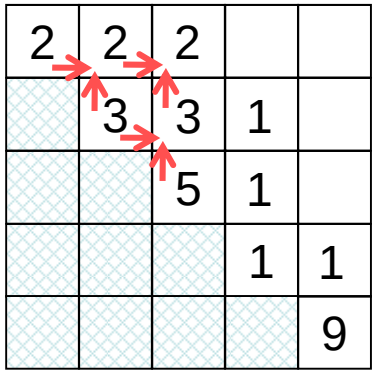
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



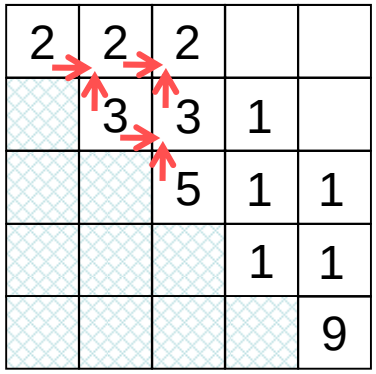
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



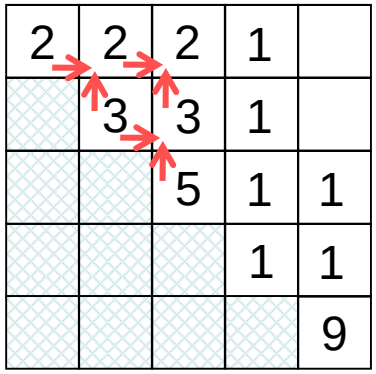
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



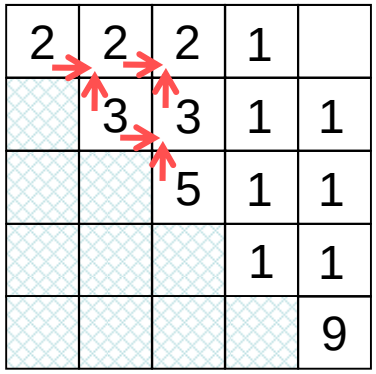
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



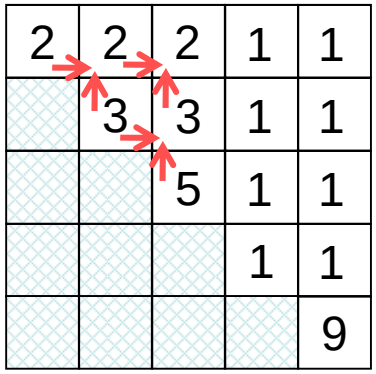
另一種 $O(n^2)$ 的算法

- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



另一種 $O(n^2)$ 的算法

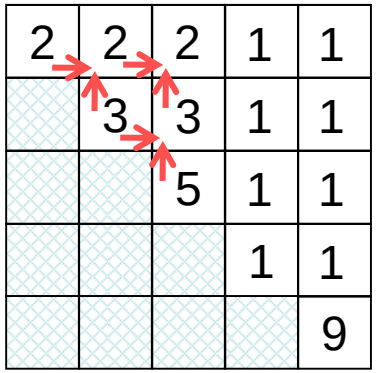
- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來



另一種 $O(n^2)$ 的算法

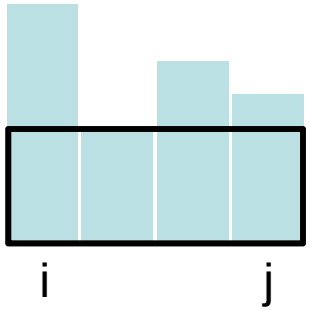
- a, b, c 的最小值可以用 min(min(a, b), min(b, c)) 計算出來

```
int i, data[100000], min[100000], max=0;
for (i=0; i<n; i++) {
    min[i] = data[i];
    if (min[i]>max) max = min[i];
}
for (i=1; i<n; i++)
    for (j=0; j<n-i; j++) {
        if (min[j]>min[j+1])
            min[j]=min[j+1],
            data[j]=data[j+1]+min[j+1];
        else
            data[j]+=min[j];
        if (data[j]>max)
            max=data[j];
    }
```



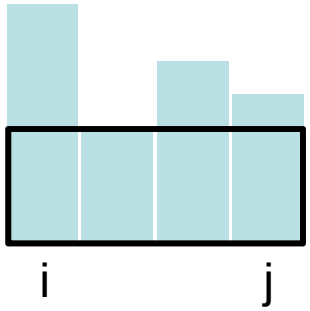
O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形



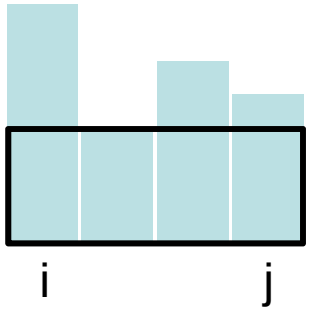
O(n logn) 的演算法

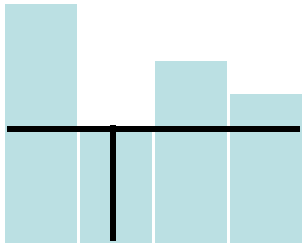
- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形



O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

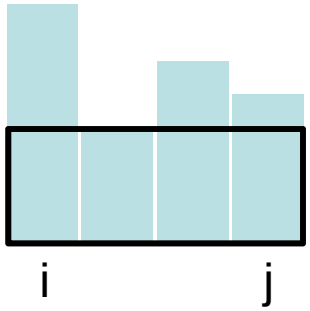


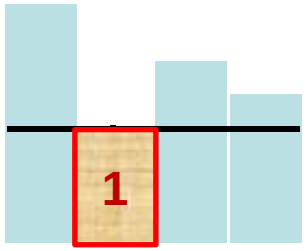


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

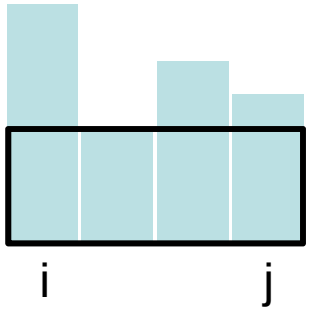


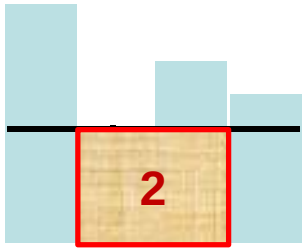


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

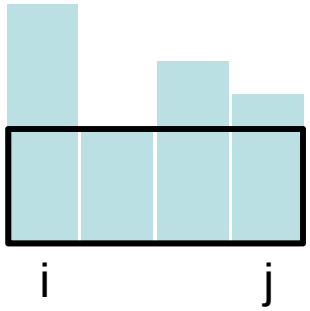


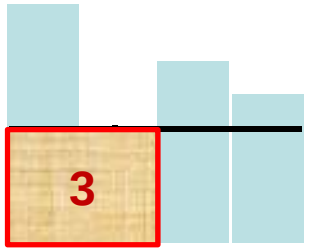


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

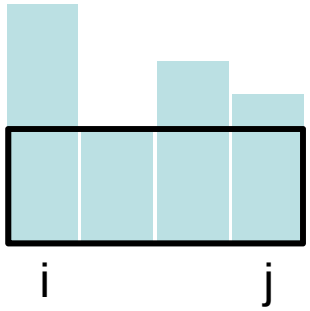




- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

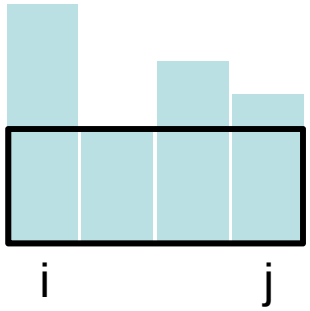
- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

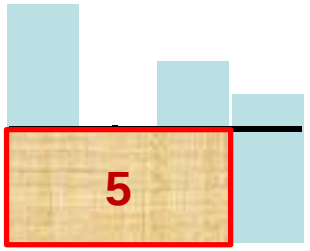


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

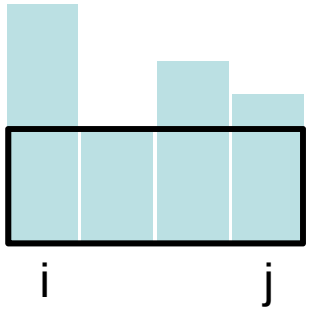


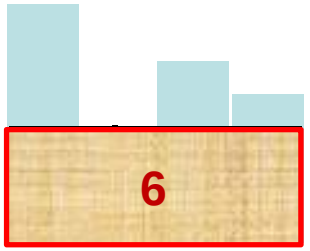


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形

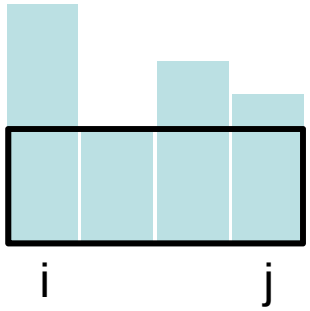


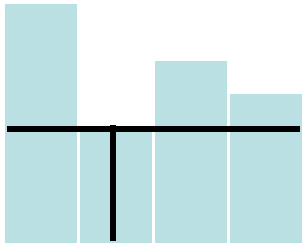


- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形

O(n logn) 的演算法

- 前面幾個方法中「矩形」是一個ㄇ字形由左邊邊界、右邊邊界、以及這個範圍中最小的高度組成, 共有 n(n+1)/2 個不同的矩形





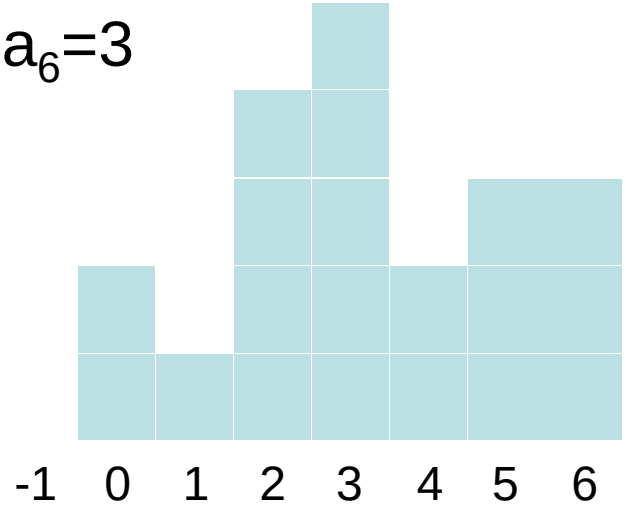
- 可是為什麼這個「矩形」不能看成像是一個 T 字形, 每一個高度紀錄左邊第一個小於它的位置, 以及右邊第一個小於它的位置, 這三個代表性的數值其實跟前面的想法是一致的, 可是在這裡看到的是 n 個矩形
- 矩形的個數減少了: 上例中高度相同的 6 個矩形我們只紀錄最大的那個矩形
- 如何有效率地找到每一個高度左邊第一個小於它的位置以及右邊第一個小於它的位置呢?

左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3

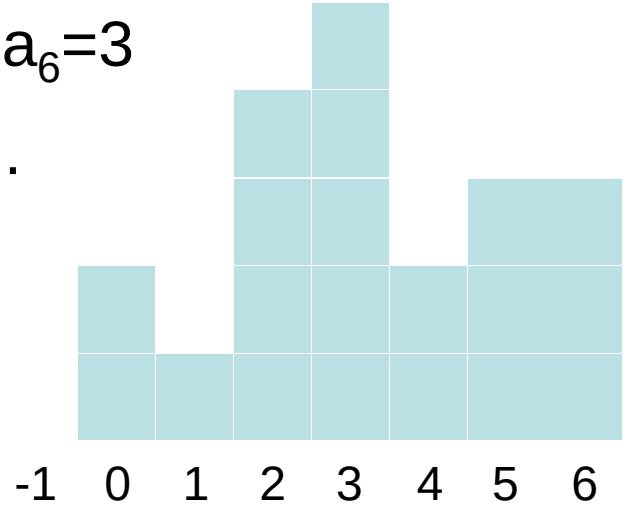
左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: $a_0=2$, $a_1=1$, $a_2=4$, $a_3=5$, $a_4=1$, $a_5=3$, $a_6=3$



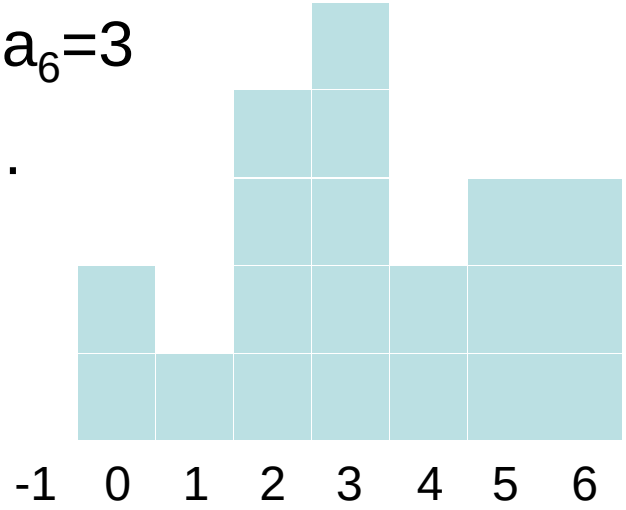
左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …



左邊第一個小於它的位置

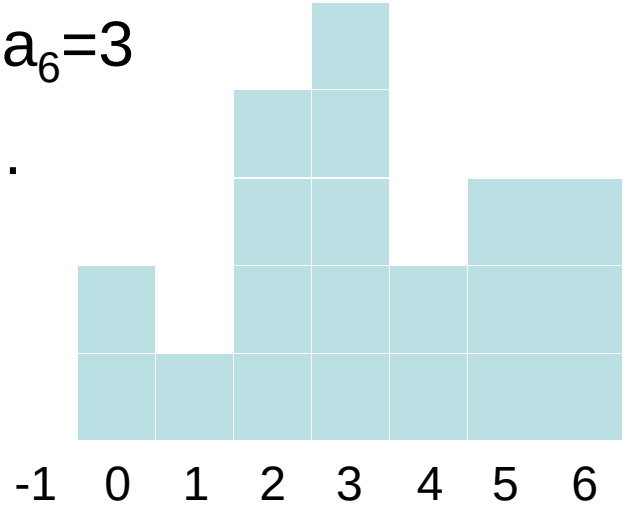
- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:



左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```

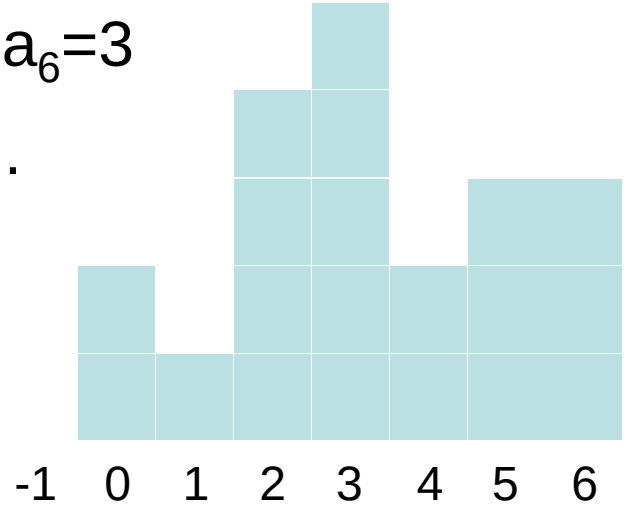


左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```

$left_0$ = -1



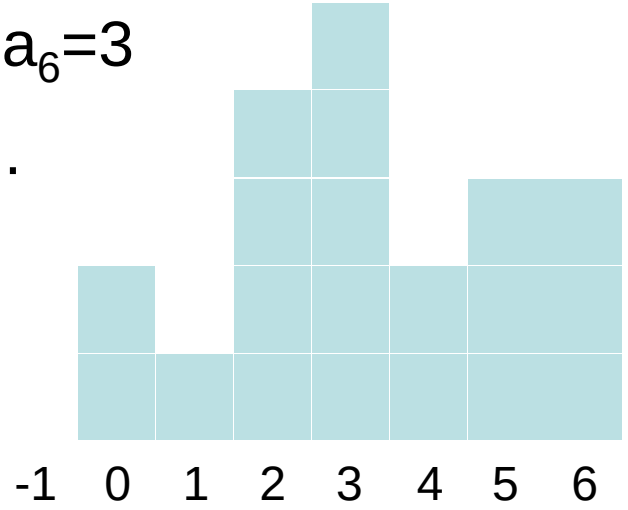
左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```

$left_0$ = -1

$left_1 \rightarrow left_0$ = -1



左邊第一個小於它的位置

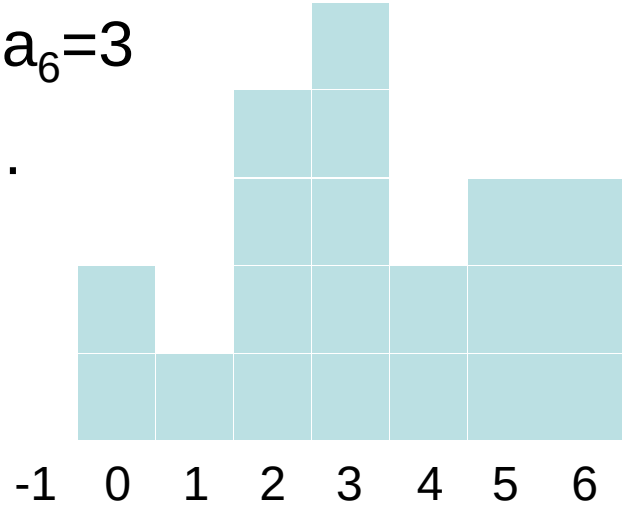
- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```

$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

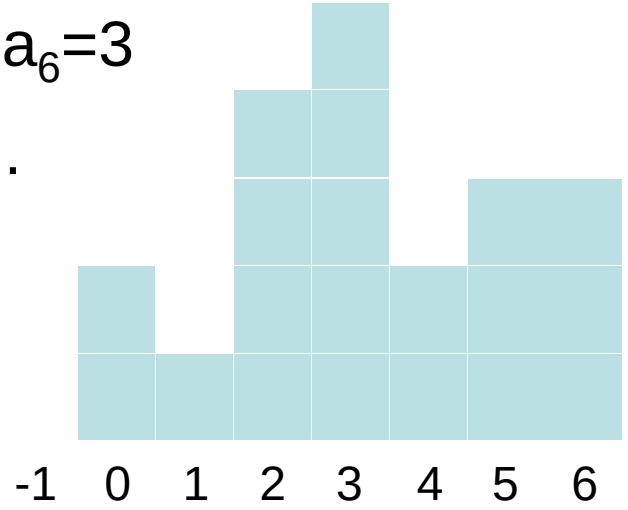
$left_2$ = 1



左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```



$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

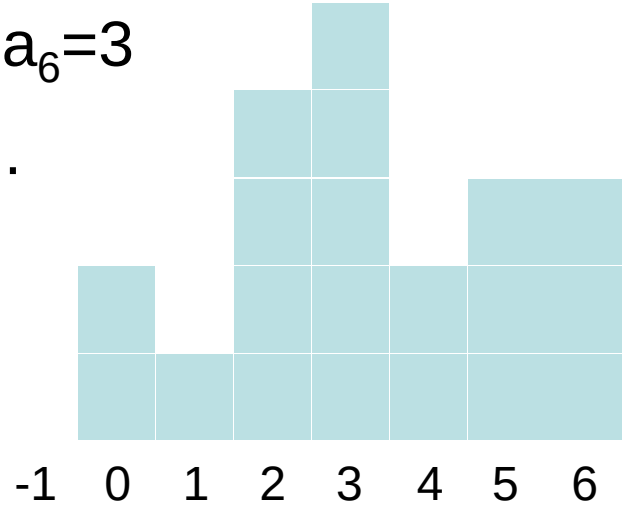
$left_2$ = 1

$left_3$ = 2

左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```



$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

$left_2$ = 1

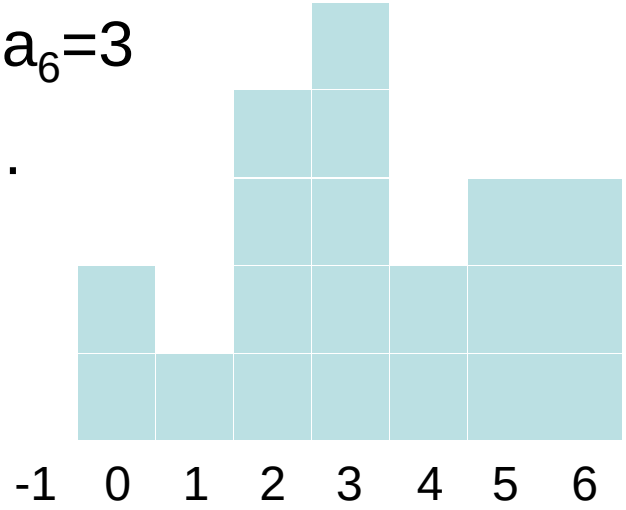
$left_3$ = 2

$left_4 \rightarrow left_3 \rightarrow left_2$ = 1

左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```



$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

$left_2$ = 1

$left_3$ = 2

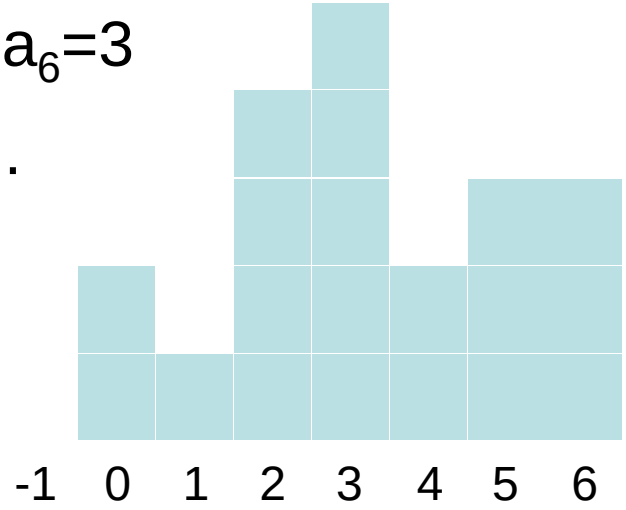
$left_4 \rightarrow left_3 \rightarrow left_2$ = 1

$left_5$ = 4

左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```



$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

$left_2$ = 1

$left_3$ = 2

$left_4 \rightarrow left_3 \rightarrow left_2$ = 1

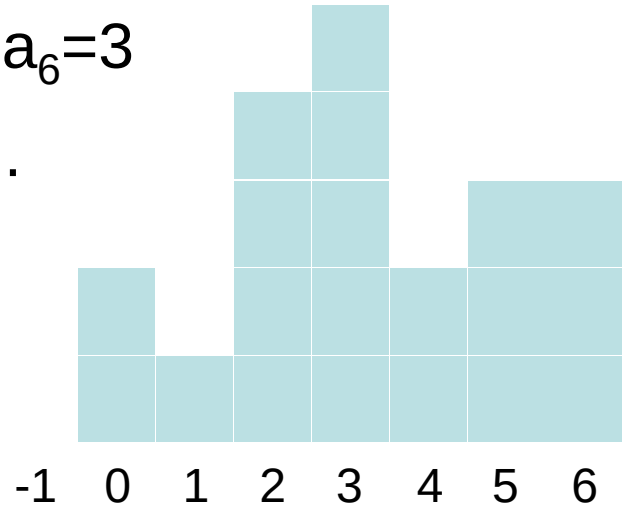
$left_5$ = 4

$left_6 \rightarrow left_5$ = 4

左邊第一個小於它的位置

- 找到每一個高度左邊第一個小於它的位置並且記錄下來
- 例: a_0=2, a_1=1, a_2=4, a_3=5, a_4=1, a_5=3, a_6=3
  - 從最左邊開始, $left_0$=-1, $left_1$, $left_2$, …
  - $left_i$:

```
x = i-1;
while ((a_x≥a_i)||(x<0)) x = left_x;
left_i = x;
```



$left_0$ = -1

$left_1 \rightarrow left_0$ = -1

$left_2$ = 1

$left_3$ = 2

$left_4 \rightarrow left_3 \rightarrow left_2$ = 1

$left_5$ = 4

$left_6 \rightarrow left_5$ = 4

- 平均每一個 $left_i$ 需要 O(log n) 運算量: 假設平均每一個 a_i 的左邊有 i/2 個數字比它大, 每次 $x = left_x$ 可以跳過 x/2 個數字, 每次看一個數字有 1/2 的機會比它小…

O(n logn) 的演算法

- 用相同的 n log n 演算法由右邊開始找, 可以找出每一個數字右邊第一個小於它的位置 $right_i$
- 運用前面步驟的結果 $left_i$, 可以很快算出最大矩形的面積是 $(right_i-left_i+1)*a_i$, 很快地計算 n 個數值的最大值就可以求出最大的矩形面積
- 整個演算法需要額外 n 個記憶體存放 $left_i$, 需要時間 n log n

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 ‘1’ 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 ‘1’ 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 ‘1’ 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

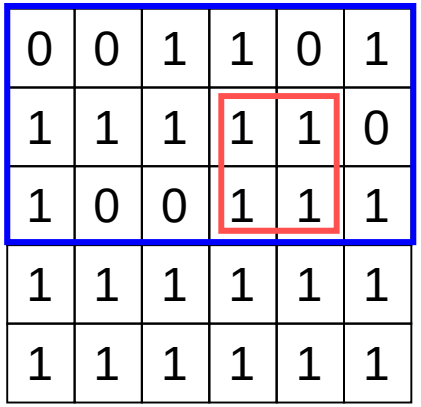
- 請寫一個程式在右圖中 ‘1’ 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形



- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

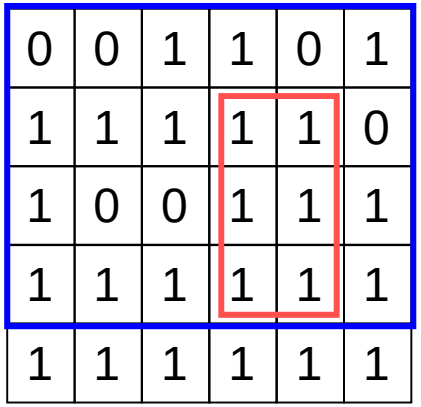
- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形



- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

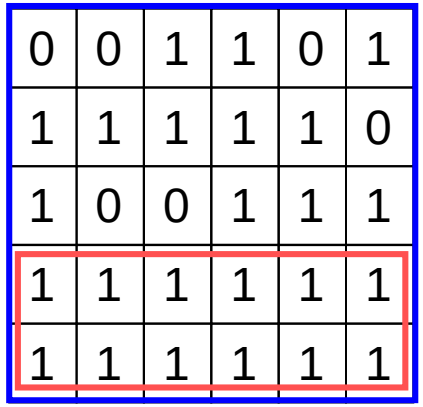
- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形

0	0	1	1	0	1
1	1	1	1	1	0
1	0	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

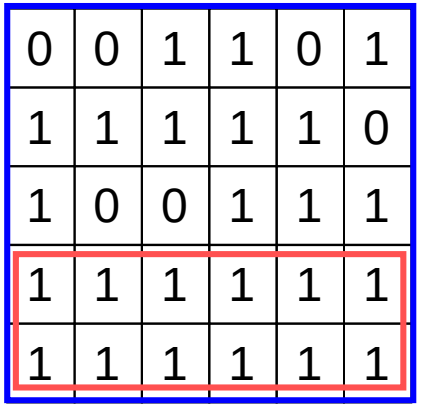
- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形



- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

ZOJ2180 (UVa 1330) City Game

- 請寫一個程式在右圖中 '1' 的區域中尋找最大面積的矩形



- 右圖是一個 nxn 的區域
  這個問題是前一個問題的延伸,
  基本上可以看成是 n 個最高高度為 i 的長方圖, 如果在這 n 個長方圖中尋找最大的矩形, 就會是這一題的答案

min{ 2, 5, 4, 6, 12 }

UVa 11389 The Bus Driver Problem

- 公車司機排班問題: 有 n 個司機, 每天需要排早晚兩班, 早上有 n 條路線長度為 $a_1, a_2, \ldots, a_n$, 晚上有 n 條路線長度為 $b_1, b_2, \ldots, b_n$, 如果每天走的路線長度超過 d 就需要發加班費 (每單位距離 r 元), 請寫一個程式找出一個加班費最少的路線安排方法, $1 \le n \le 100$, $1 \le d \le 10000$, $1 \le r \le 5$

範例輸入:

```
2 20 5          n  d  r
10 15           早班路線長度
10 15           晚班路線長度
2 20 5
10 10
10 10
16 8772 2
2640 1550 4569 8454 1619 1851 6227 7661 7358 4083 2809 4038 6815 7715 8287 2406
7073 1582 1852 5283 682 4079 5312 4624 9252 9885 358 7055 1611 9568 6662 7954
```

範例輸出:

```
50          最少的加班費
0
41124
```

- 暴力法: 如果長度都不相等, 每一個 a_i 配一個 b_j, 運算時間 $O(n!)$, $1 \le n \le 100$ 有限時間裡完全算不出來!

解法

- 早班所有長度由小到大排好 $a_1 \le a_2 \le a_3 \le \ldots \le a_n$
  晚班所有長度由大到小排好 $b_1 \ge b_2 \ge b_3 \ge \ldots \ge b_n$

解法

- 早班所有長度由小到大排好 $a_1 \le a_2 \le a_3 \le \ldots \le a_n$
  晚班所有長度由大到小排好 $b_1 \ge b_2 \ge b_3 \ge \ldots \ge b_n$
- $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ 就是加班費最少的路線安排方法

解法

- 早班所有長度由小到大排好 $a_1 \le a_2 \le a_3 \le \ldots \le a_n$
  晚班所有長度由大到小排好 $b_1 \ge b_2 \ge b_3 \ge \ldots \ge b_n$
- $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ 就是加班費最少的路線安排方法

```
#include <stdio.h>
#include <stdlib.h> // qsort
int comp1(const void *a, const void *b) { return *(int*)a - *(int*)b; }
int comp2(const void *a, const void *b) { return *(int*)b - *(int*)a; }
int main() {
   int n=16, d=8772, r=2, i;
   int a[] = {2640, 1550, 4569, 8454, 1619, 1851, 6227, 7661,
              7358, 4083, 2809, 4038, 6815, 7715, 8287, 2406};
   int b[] = {7073, 1582, 1852, 5283, 682, 4079, 5312, 4624,
              9252, 9885, 358, 7055, 1611, 9568, 6662, 7954};
   qsort(a, n, sizeof(int), comp1); qsort(b, n, sizeof(int), comp2);
   for (i=0; i<n; i++) printf("(%d,%d) ", a[i], b[i]); printf("\n");
   return 0;
}
```

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?
- 希望你體會到數學的訓練是絕對必要的, 如果你說反正我會寫程式, 最差就是暴力法也一定可以完成, 可是很多問題是暴力搜尋沒有辦法在這個宇宙毀滅以前完成的喔!

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?
- 希望你體會到數學的訓練是絕對必要的, 如果你說反正我會寫程式, 最差就是暴力法也一定可以完成, 可是很多問題是暴力搜尋沒有辦法在這個宇宙毀滅以前完成的喔!
- 若 $a_1<a_2$ 且 $b_1\geq b_2$ 則 $C((a_1,b_1),(a_2,b_2)) \leq C((a_1,b_2),(a_2,b_1))$

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?
- 希望你體會到數學的訓練是絕對必要的, 如果你說反正我會寫程式, 最差就是暴力法也一定可以完成, 可是很多問題是暴力搜尋沒有辦法在這個宇宙毀滅以前完成的喔!
- 若 $a_1<a_2$ 且 $b_1 \geq b_2$ 則 $C((a_1,b_1),(a_2,b_2)) \leq C((a_1,b_2),(a_2,b_1))$
  - $C((a_1,b_1),(a_2,b_2)) = f(a_1,b_1)+f(a_2,b_2)$

    where $f(a,b)=a+b-d$ if $a+b>d$, 0 otherwise

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?
- 希望你體會到數學的訓練是絕對必要的, 如果你說反正我會寫程式, 最差就是暴力法也一定可以完成, 可是很多問題是暴力搜尋沒有辦法在這個宇宙毀滅以前完成的喔!
- 若 $a_1<a_2$ 且 $b_1\geq b_2$ 則 $C((a_1,b_1),(a_2,b_2)) \leq C((a_1,b_2),(a_2,b_1))$
  - $C((a_1,b_1),(a_2,b_2)) = f(a_1,b_1)+f(a_2,b_2)$
    where $f(a,b)=a+b-d$ if $a+b>d$, 0 otherwise
  - 分成四個狀況來討論

正確性證明

- 上面的方法很簡單, 但是你有信心它找到的就是答案嗎? 如果老闆說萬一他找到更短的安排方法的話, 加班費都給你出…你敢說沒問題嗎?
- 希望你體會到數學的訓練是絕對必要的, 如果你說反正我會寫程式, 最差就是暴力法也一定可以完成, 可是很多問題是暴力搜尋沒有辦法在這個宇宙毀滅以前完成的喔!
- 若 $a_1<a_2$ 且 $b_1\geq b_2$ 則 $C((a_1,b_1),(a_2,b_2)) \leq C((a_1,b_2),(a_2,b_1))$
  - $C((a_1,b_1),(a_2,b_2)) = f(a_1,b_1)+f(a_2,b_2)$

    where $f(a,b)=a+b-d$ if $a+b>d$, 0 otherwise
  - 分成四個狀況來討論
    1. $a_1+b_1\geq d$ 且 $a_2+b_2\geq d$
    2. $a_1+b_1\geq d$ 且 $a_2+b_2<d$
    3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$
    4. $a_1+b_1<d$ 且 $a_2+b_2<d$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$

1. $a_1+b_1\geq d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$

  $=r(a_1+b_2-d)+r(a_2+b_1-d)$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $a_2+b_2 \geq d \Rightarrow a_2+b_1 \geq a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$
  $=r(a_1+b_2-d)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(a_1+b_2-d) = f(a_1,b_2)$ if $a_1+b_2 \geq d$
  - $r(a_1+b_2-d) < 0 = f(a_1,b_2)$ if $a_1+b_2<d$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $a_2+b_2 \geq d \Rightarrow a_2+b_1 \geq a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$
  $=r(a_1+b_2-d)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(a_1+b_2-d) = f(a_1,b_2)$ if $a_1+b_2 \geq d$
  - $r(a_1+b_2-d) < 0 = f(a_1,b_2)$ if $a_1+b_2<d$

2. $a_1+b_1 \geq d$ 且 $a_2+b_2<d$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $a_2+b_2 \geq d \Rightarrow a_2+b_1 \geq a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$
  $=r(a_1+b_2-d)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(a_1+b_2-d) = f(a_1,b_2)$ if $a_1+b_2 \geq d$
  - $r(a_1+b_2-d) < 0 = f(a_1,b_2)$ if $a_1+b_2<d$

2. $a_1+b_1 \geq d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+0$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $a_2+b_2 \geq d \Rightarrow a_2+b_1 \geq a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$
  $=r(a_1+b_2-d)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(a_1+b_2-d) = f(a_1,b_2)$ if $a_1+b_2 \geq d$
  - $r(a_1+b_2-d) < 0 = f(a_1,b_2)$ if $a_1+b_2<d$

2. $a_1+b_1 \geq d$ 且 $a_2+b_2<d$

- $a_1+b_1 \geq d \Rightarrow a_2+b_1 \geq a_1+b_1 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+0$
  $=r(a_2+b_1-d)+r(a_1-a_2)$

1. $a_1+b_1 \geq d$ 且 $a_2+b_2 \geq d$

- $a_2+b_2 \geq d \Rightarrow a_2+b_1 \geq a_2+b_2 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+r(a_2+b_2-d)$
  $=r(a_1+b_2-d)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(a_1+b_2-d) = f(a_1,b_2)$ if $a_1+b_2 \geq d$
  - $r(a_1+b_2-d) < 0 = f(a_1,b_2)$ if $a_1+b_2<d$

2. $a_1+b_1 \geq d$ 且 $a_2+b_2<d$

- $a_1+b_1 \geq d \Rightarrow a_2+b_1 \geq a_1+b_1 \geq d$

- $f(a_1,b_1)+f(a_2,b_2)=r(a_1+b_1-d)+0$
  $=r(a_2+b_1-d)+r(a_1-a_2)$
  $\leq f(a_2,b_1)+f(a_2,b_1)$
  - $r(a_1-a_2) < 0 \leq f(a_2,b_1)$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$
- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$
- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$

  $=r(b_2-b_1)+r(a_2+b_1-d)$

  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$
- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0 \leq f(a_1,b_2)+f(a_2,b_1)$

  - $0 \leq f(a_1,b_2)$
  - $0 \leq f(a_2,b_1)$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0 \leq f(a_1,b_2)+f(a_2,b_1)$

  - $0 \leq f(a_1,b_2)$
  - $0 \leq f(a_2,b_1)$

➢ 假設 **$a_1 < a_2 < a_3 < \ldots < a_n$, $b_1 \geq b_2 \geq b_3 \geq \ldots \geq b_n$**

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0 \leq f(a_1,b_2)+f(a_2,b_1)$

  - $0 \leq f(a_1,b_2)$
  - $0 \leq f(a_2,b_1)$

➢ 假設 $a_1 < a_2 < a_3 < \ldots < a_n$, $b_1 \geq b_2 \geq b_3 \geq \ldots \geq b_n$

➢ 一開始依序對每一個 a_i 安排任意一個 b_j, b_j 並不依照順序

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$

- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$

  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0 \leq f(a_1,b_2)+f(a_2,b_1)$

  - $0 \leq f(a_1,b_2)$
  - $0 \leq f(a_2,b_1)$

- 假設 $a_1 < a_2 < a_3 < \ldots < a_n$, $b_1 \geq b_2 \geq b_3 \geq \ldots \geq b_n$
- 一開始依序對每一個 a_i 安排任意一個 b_j, b_j 並不依照順序
- 比對相鄰配對中的 b_j, 例如 $(a_i,b_j),(a_{i+1},b_k)$, 如果 $j>k$ 就交換成為$(a_i,b_k),(a_{i+1},b_j)$, 此時 $C((a_i,b_k),(a_{i+1},b_j))\leq C((a_i,b_k),(a_{i+1},b_j))$

3. $a_1+b_1<d$ 且 $a_2+b_2\geq d$

- $a_2+b_2\geq d \Rightarrow a_2+b_1\geq a_2+b_2\geq d$
- $f(a_1,b_1)+f(a_2,b_2)=0+r(a_2+b_2-d)$
  $=r(b_2-b_1)+r(a_2+b_1-d)$
  $\leq f(a_1,b_2)+f(a_2,b_1)$
  - $r(b_2-b_1) < 0 \leq f(a_1,b_2)$

4. $a_1+b_1<d$ 且 $a_2+b_2<d$

- $f(a_1,b_1)+f(a_2,b_2)=0+0 \leq f(a_1,b_2)+f(a_2,b_1)$
  - $0 \leq f(a_1,b_2)$
  - $0 \leq f(a_2,b_1)$

- 假設 $a_1 < a_2 < a_3 < \ldots < a_n$, $b_1 \geq b_2 \geq b_3 \geq \ldots \geq b_n$
- 一開始依序對每一個 a_i 安排任意一個 b_j, b_j 並不依照順序
- 比對相鄰配對中的 b_j, 例如 $(a_i,b_j),(a_{i+1},b_k)$, 如果 $j>k$ 就交換成為$(a_i,b_k),(a_{i+1},b_j)$, 此時 $C((a_i,b_k),(a_{i+1},b_j))\leq C((a_i,b_k),(a_{i+1},b_j))$
- 運用類似 **Bubble Sort** 方法來依序比對相鄰的配對, 最後可以得到 $(a_1,b_1),(a_2,b_2),\ldots,(a_n,b_n)$, 而且加班費一直不斷地降低

Code[VS]3286 火柴排隊

- 兩盒火柴各有 n 支火柴(1≤n≤100)，長度分別為 $a_1, a_2, \ldots, a_n$ 以及$b_1, b_2, \ldots, b_n$。現在將每盒中的火柴各自排成一列，兩列火柴之間的距離定義為： $\Sigma (a_i-b_i)^2$，請問最小的距離是多少？如果每列火柴中相鄰兩根火柴的位置都可以交換，請你通過交換使得兩列火柴之間的距離最小，請問最少需要交換多少次？如果這個數字太大，請輸出這個最小交換次數對 99,999,997 取模的結果。

Code[VS]3286 火柴排隊

- 兩盒火柴各有 n 支火柴(1≤n≤100)，長度分別為 $a_1, a_2, \ldots, a_n$ 以及$b_1, b_2, \ldots, b_n$。現在將每盒中的火柴各自排成一列，兩列火柴之間的距離定義為： $\Sigma (a_i-b_i)^2$，請問最小的距離是多少？如果每列火柴中相鄰兩根火柴的位置都可以交換，請你通過交換使得兩列火柴之間的距離最小，請問最少需要交換多少次？如果這個數字太大，請輸出這個最小交換次數對 99,999,997 取模的結果。

範例輸入:	範例輸出:
4 2 3 1 4 3 2 1 4 4 1 3 4 2 1 7 2 4	0 1 10 2

Code[VS]3286 火柴排隊

- 兩盒火柴各有 n 支火柴(1≤n≤100)，長度分別為 $a_1, a_2, \ldots, a_n$ 以及$b_1, b_2, \ldots, b_n$。現在將每盒中的火柴各自排成一列，兩列火柴之間的距離定義為： $\Sigma (a_i-b_i)^2$，請問最小的距離是多少？如果每列火柴中相鄰兩根火柴的位置都可以交換，請你通過交換使得兩列火柴之間的距離最小，請問最少需要交換多少次？如果這個數字太大，請輸出這個最小交換次數對 99,999,997 取模的結果。

範例輸入:

```
4
2 3 1 4
3 2 1 4
4
1 3 4 2
1 7 2 4
```

範例輸出:

```
0 1
10 2
```

- 暴力列舉：如果長度都不相等，每一個 a_i 配一個 b_j, 運算時間 O(n!), 1≤n≤100

Code[VS]3286 火柴排隊

- 兩盒火柴各有 n 支火柴(1≤n≤100)，長度分別為 $a_1, a_2, \ldots, a_n$ 以及$b_1, b_2, \ldots, b_n$。現在將每盒中的火柴各自排成一列，兩列火柴之間的距離定義為： $\Sigma (a_i-b_i)^2$，請問最小的距離是多少？如果每列火柴中相鄰兩根火柴的位置都可以交換，請你通過交換使得兩列火柴之間的距離最小，請問最少需要交換多少次？如果這個數字太大，請輸出這個最小交換次數對 99,999,997 取模的結果。

範例輸入:
```
4
2 3 1 4
3 2 1 4
4
1 3 4 2
1 7 2 4
```

範例輸出:
```
0 1
10 2
```

- 暴力列舉：如果長度都不相等，每一個 a_i 配一個 b_j, 運算時間 O(n!), 1≤n≤100
- 有限時間裡完全算不出來!

解法

- 讀入 $(a_1, b_1),(a_2, b_2), \ldots, (a_n, b_n)$

解法

- 讀入 $(a_1, b_1),(a_2, b_2), \ldots, (a_n, b_n)$
- 根據 a_i 由小到大排序 $(a_{f(1)}, b_{f(1)}),(a_{f(2)}, b_{f(2)}), \ldots, (a_{f(n)}, b_{f(n)})$

解法

- 讀入 $(a_1, b_1),(a_2, b_2), \ldots, (a_n, b_n)$
- 根據 a_i 由小到大排序 $(a_{f(1)}, b_{f(1)}),(a_{f(2)}, b_{f(2)}), \ldots, (a_{f(n)}, b_{f(n)})$
- 計算 $b_{f(i)}$ 序列的逆序數 ($O(n^2)$ 或是 $O(n \log n)$)

解法

- 讀入 $(a_1, b_1),(a_2, b_2), \ldots, (a_n, b_n)$
- 根據 a_i 由小到大排序 $(a_{f(1)}, b_{f(1)}),(a_{f(2)}, b_{f(2)}), \ldots, (a_{f(n)}, b_{f(n)})$
- 計算 $b_{f(i)}$ 序列的逆序數 ($O(n^2)$ 或是 $O(n \log n)$)
- 將 b_i 由小到大排序 $b_{g(1)}, b_{g(2)}, \ldots, b_{g(n)}$, 計算 $\Sigma (a_{f(i)}-b_{g(i)})^2$

解法

- 讀入 $(a_1, b_1),(a_2, b_2), \ldots, (a_n, b_n)$
- 根據 a_i 由小到大排序 $(a_{f(1)}, b_{f(1)}),(a_{f(2)}, b_{f(2)}), \ldots, (a_{f(n)}, b_{f(n)})$
- 計算 $b_{f(i)}$ 序列的逆序數 ($O(n^2)$ 或是 $O(n \log n)$)
- 將 b_i 由小到大排序 $b_{g(1)}, b_{g(2)}, \ldots, b_{g(n)}$, 計算 $\Sigma (a_{f(i)}-b_{g(i)})^2$

```
int comp(const void *a1, const void *a2) { return *(int *)a1-*(int *)a2; }
…
scanf("%d",&n);
for (i=0; i<n; i++) { scanf("%d",&a[i][0]); a[i][1] = i; }
qsort(a, n, sizeof(int[2]), comp);
for (i=0; i<n; i++) scanf("%d",&b[i]);
for (count=i=0; i<n; i++)
    for (j=i+1; i<n; j++)
        count += b[a[i][1]]>b[a[j][2]];
qsort(b, n, sizeof(int), comp);
for (dist=i=0; i<n; i++)
    dist += (a[i][0]-b[i])*(a[i][0]-b[i]);
```

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \leq (a_1-b_2)^2+(a_2-b_1)^2$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$

$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$

$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$

$$=(a_2-b_2)^2+(a_1-b_1)^2+2(a_2-a_1)\,(b_2-b_1)$$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$

$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$

$$=(a_2-b_2)^2+(a_1-b_1)^2+2(a_2-a_1)\,(b_2-b_1)$$

  - 若 $(a_1,b_1), (a_2,b_2)$ 配對, 但是 $a_1<a_2$ 且 $b_1\ge b_2$ 則交換 b_1 與 b_2 可以得到距離比較小的配對 $(a_1,b_2), (a_2,b_1)$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$

$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$

$$=(a_2-b_2)^2+(a_1-b_1)^2+2(a_2-a_1)\,(b_2-b_1)$$

  - 若 $(a_1,b_1),(a_2,b_2)$ 配對, 但是 $a_1<a_2$ 且 $b_1\ge b_2$ 則交換 b_1 與 b_2 可以得到距離比較小的配對 $(a_1,b_2),(a_2,b_1)$

- 讀入 (a_i,b_i) 根據 a_i 由小到大排序後 $b_{f(j)}$ 並非依照順序

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$
$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$
$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$
$$=(a_2-b_2)^2+(a_1-b_1)^2+2(a_2-a_1)\,(b_2-b_1)$$

  - 若 $(a_1,b_1),(a_2,b_2)$ 配對, 但是 $a_1<a_2$ 且 $b_1\ge b_2$ 則交換 b_1 與 b_2 可以得到距離比較小的配對 $(a_1,b_2),(a_2,b_1)$

- 讀入 (a_i,b_i) 根據 a_i 由小到大排序後 $b_{f(j)}$ 並非依照順序

- 比對相鄰配對中的 $b_{f(j)}$, 例如 $(a_i,b_j),(a_{i+1},b_k)$, 如果 $b_j>b_k$ 就交換成為$(a_i,b_k),(a_{i+1},b_j)$, 此時 $(a_i-b_k)^2+(a_{i+1}-b_j)^2\le(a_i-b_k)^2+(a_{i+1}-b_j)^2$

正確性證明

- 若 $a_1<a_2$ 且 $b_1<b_2$ 則 $(a_1-b_1)^2+(a_2-b_2)^2 \le (a_1-b_2)^2+(a_2-b_1)^2$

$$(a_2-b_1)^2+(a_1-b_2)^2 =(a_2-b_2+b_2-b_1)^2+(a_1-b_1+b_1-b_2)^2$$

$$=(a_2-b_2)^2+2(a_2-b_2)\,(b_2-b_1)+(b_2-b_1)^2+$$

$$(a_1-b_1)^2+2(a_1-b_1)\,(b_1-b_2)+(b_1-b_2)^2$$

$$=(a_2-b_2)^2+(a_1-b_1)^2+2(a_2-a_1)\,(b_2-b_1)$$

  - 若 $(a_1,b_1),(a_2,b_2)$ 配對, 但是 $a_1<a_2$ 且 $b_1\ge b_2$ 則交換 b_1 與 b_2 可以得到距離比較小的配對 $(a_1,b_2),(a_2,b_1)$

- 讀入 (a_i,b_i) 根據 a_i 由小到大排序後 $b_{f(j)}$ 並非依照順序
- 比對相鄰配對中的 $b_{f(j)}$, 例如 $(a_i,b_j),(a_{i+1},b_k)$, 如果 $b_j>b_k$ 就交換成為$(a_i,b_k),(a_{i+1},b_j)$, 此時 $(a_i-b_k)^2+(a_{i+1}-b_j)^2\le(a_i-b_k)^2+(a_{i+1}-b_j)^2$
- 運用 **Bubble Sort** 的順序來比對並且交換相鄰配對中的 b_j與b_k, 可以得到 依序排列的 (a_i,b_i)配對, 而且總距離不斷地降低

簡單易懂的現代都市

單調隊列

問題描述

- 隨著時代的變遷，有的國際大都市擺脫了從前都市帶給人的灰濛濛的印象，進而取代的是以華麗的大片玻璃，反射光線、映照出藍色天空的大廈，或是五彩繽紛的鮮豔色彩彩妝的大樓。

問題描述

- 隨著時代的變遷，有的國際大都市擺脫了從前都市帶給人的灰濛濛的印象，進而取代的是以華麗的大片玻璃，反射光線、映照出藍色天空的大廈，或是五彩繽紛的鮮豔色彩彩妝的大樓。
- Prinkskatcv 就是大家口中「現代都市」的最佳範本。在 Prinkskatcv 的區域範圍內，你看不到傳統帶給人壓迫感的水泥建築，取而代之的是整體感十足、縱然走在高樓大廈之間也不會感受到其壓力的現代建築。

- 有一位知名的攝影師前往 Prinkskatcv 中最著名的 Kosivli St. 拍照放到所屬的攝影雜誌上，由於 **Kosivli St. 的高樓大廈**太多了 (共有 ***N* 棟**)，無法容納在一張照片內，每棟大廈都有其特色，讓攝影師一時無法決定要使用哪張照片；此外作為封面的照片包含的景物不能太空曠也不能太稠密。因此，這位攝影師決定挑選出符合「照片中最高與最低的高樓大廈差正好為 ***K* 公尺**」的照片，但是還是有太多種可能的拍攝地點了，請寫一個程式幫他找出所有符合條件的拍攝位置。

- 有一位知名的攝影師前往 Prinkskatcv 中最著名的 Kosivli St. 拍照放到所屬的攝影雜誌上，由於 **Kosivli St. 的高樓大廈**太多了 (共有 ***N* 棟**)，無法容納在一張照片內，每棟大廈都有其特色，讓攝影師一時無法決定要使用哪張照片；此外作為封面的照片包含的景物不能太空曠也不能太稠密。因此，這位攝影師決定挑選出符合「照片中最高與最低的高樓大廈差正好為 ***K* 公尺**」的照片，但是還是有太多種可能的拍攝地點了，請寫一個程式幫他找出所有符合條件的拍攝位置。
- 假設 Prinkskatcv 市內的高樓大廈每棟的寬度都相同且距離都可以被忽略，這位攝影師的相機鏡頭一次可以容納 ***M* 棟**高樓大廈 (街道的兩端可能拍攝2≦*M*‘≦*M*棟建築)，而且Kosivli St.上的所有的建築只有單側且排成一直線。

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

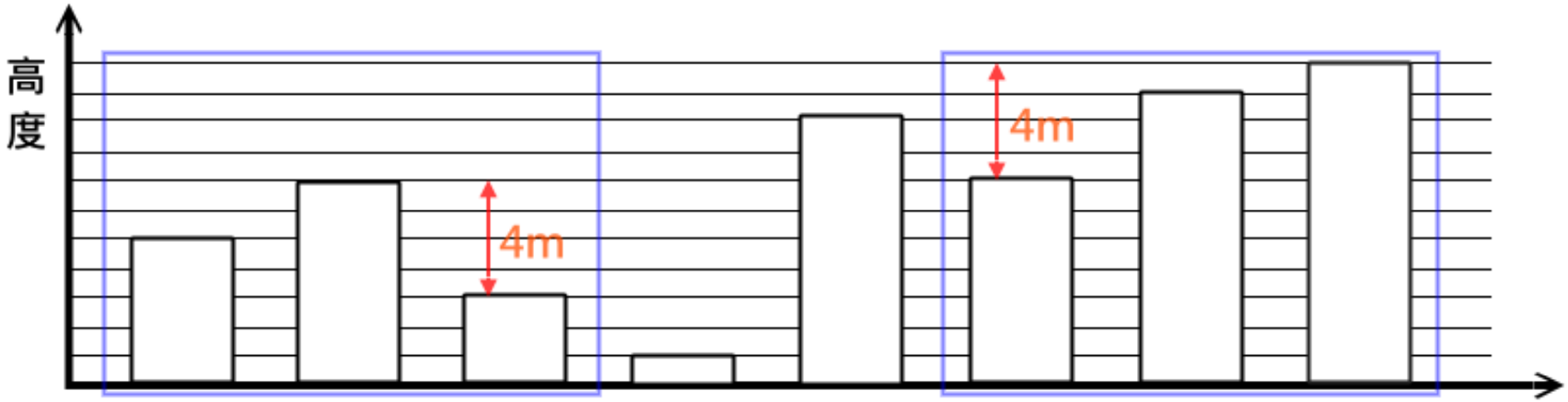


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

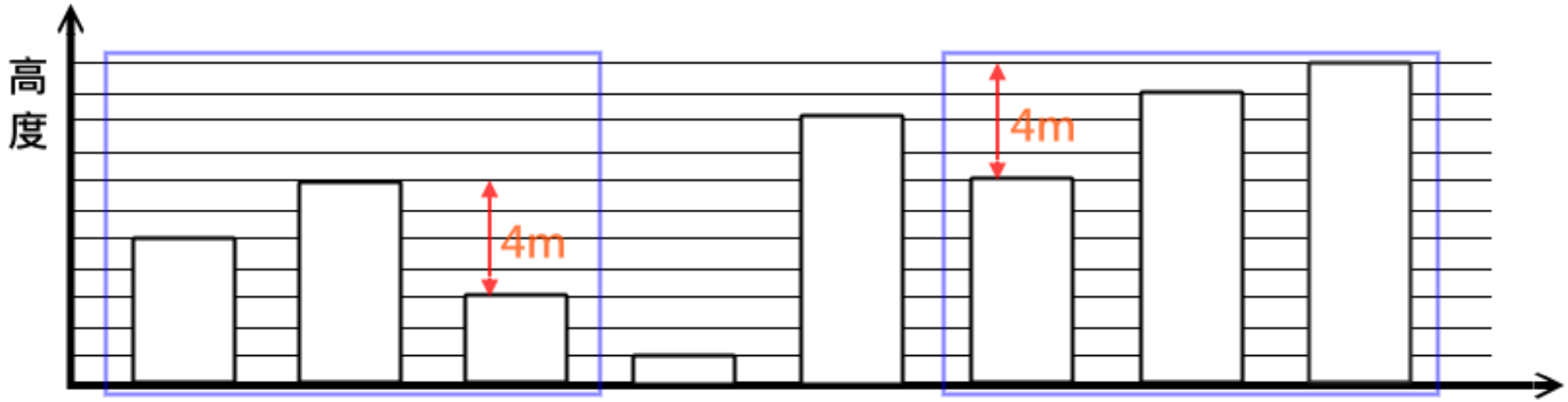


Figure 1. 範例示意圖

[0,2]

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

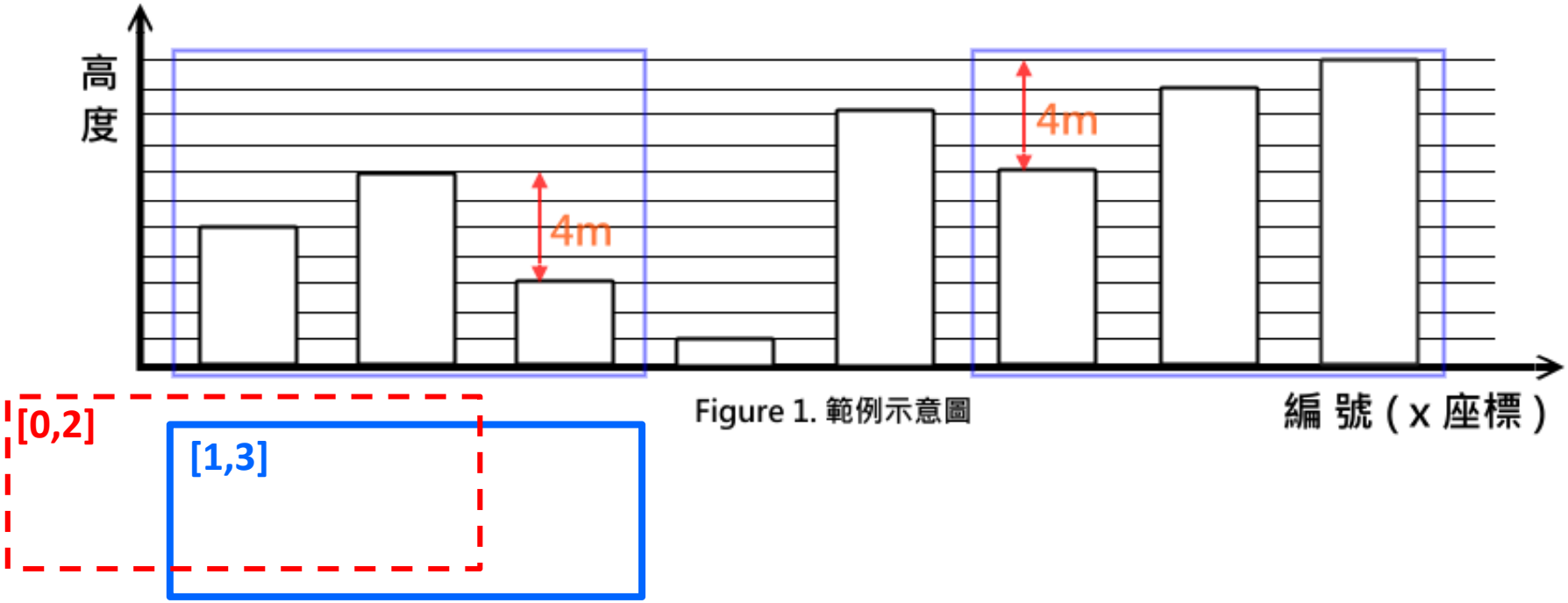


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

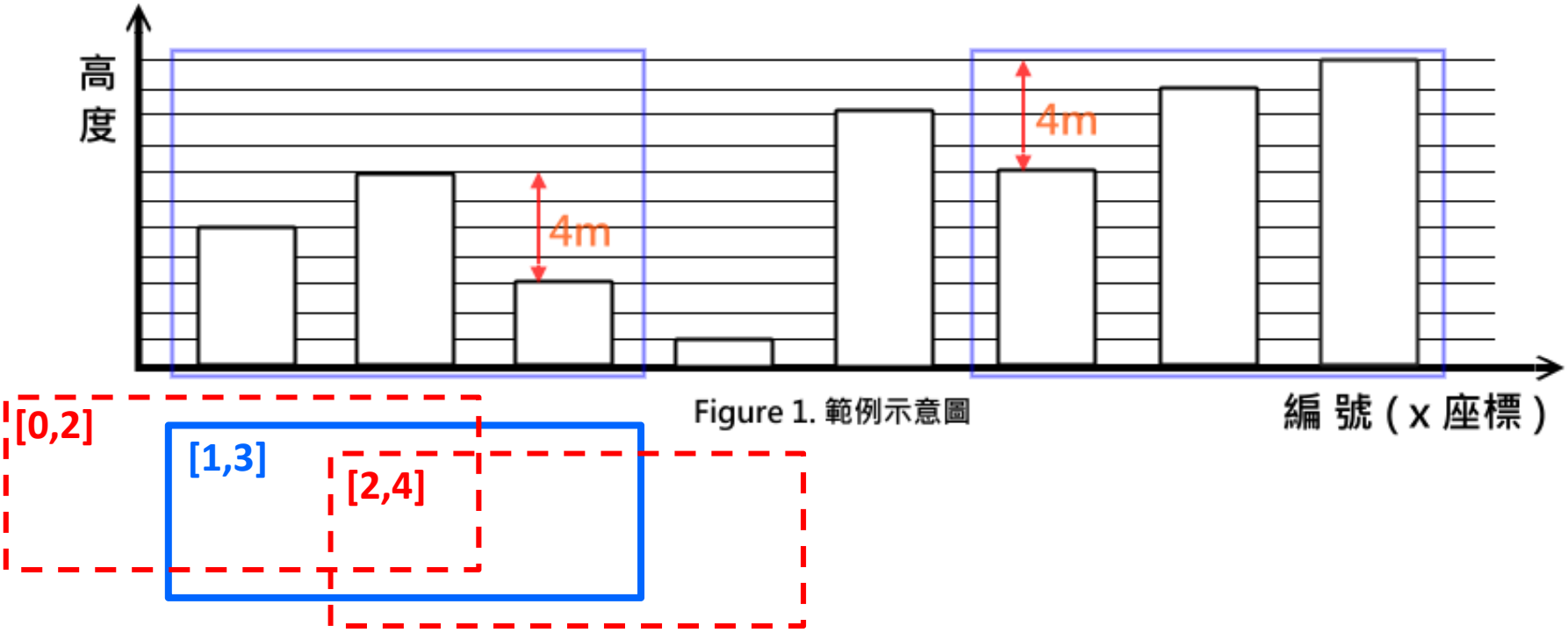


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

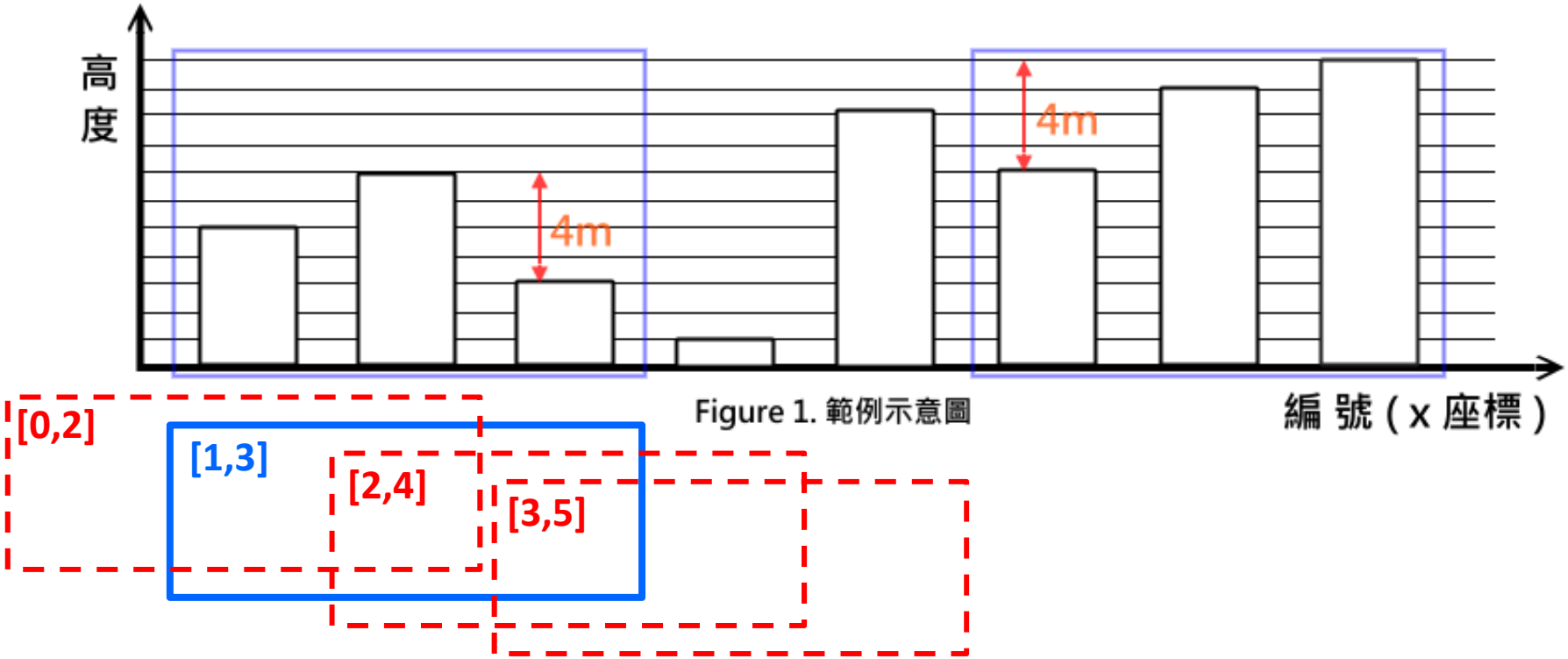


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

$N=8, M=3, K=4$

$H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11$

$2 \leqq N \leqq 10^7, 2 \leqq M \leqq 10^6, 1 \leqq K, H_i \leqq 2^{31}$

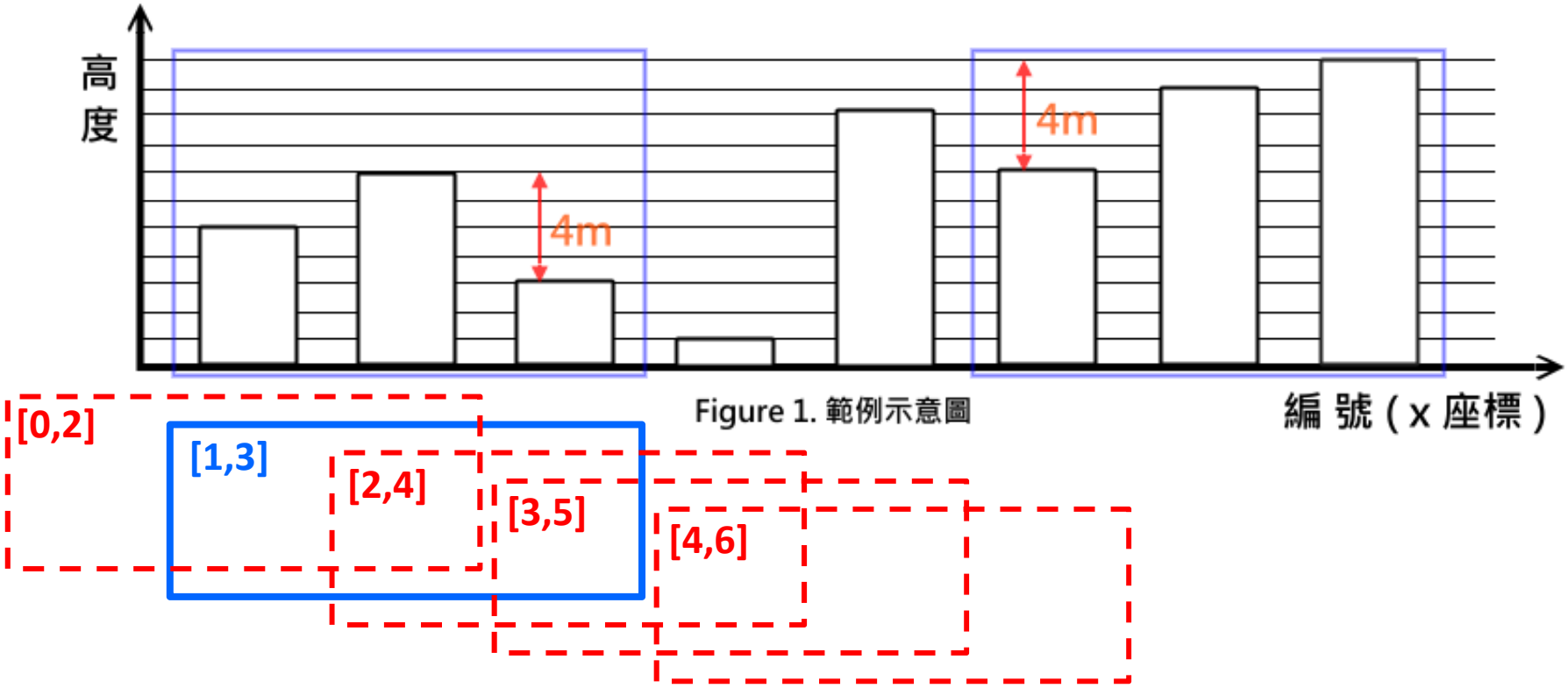


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

N=8, M=3, K=4

H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

$2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

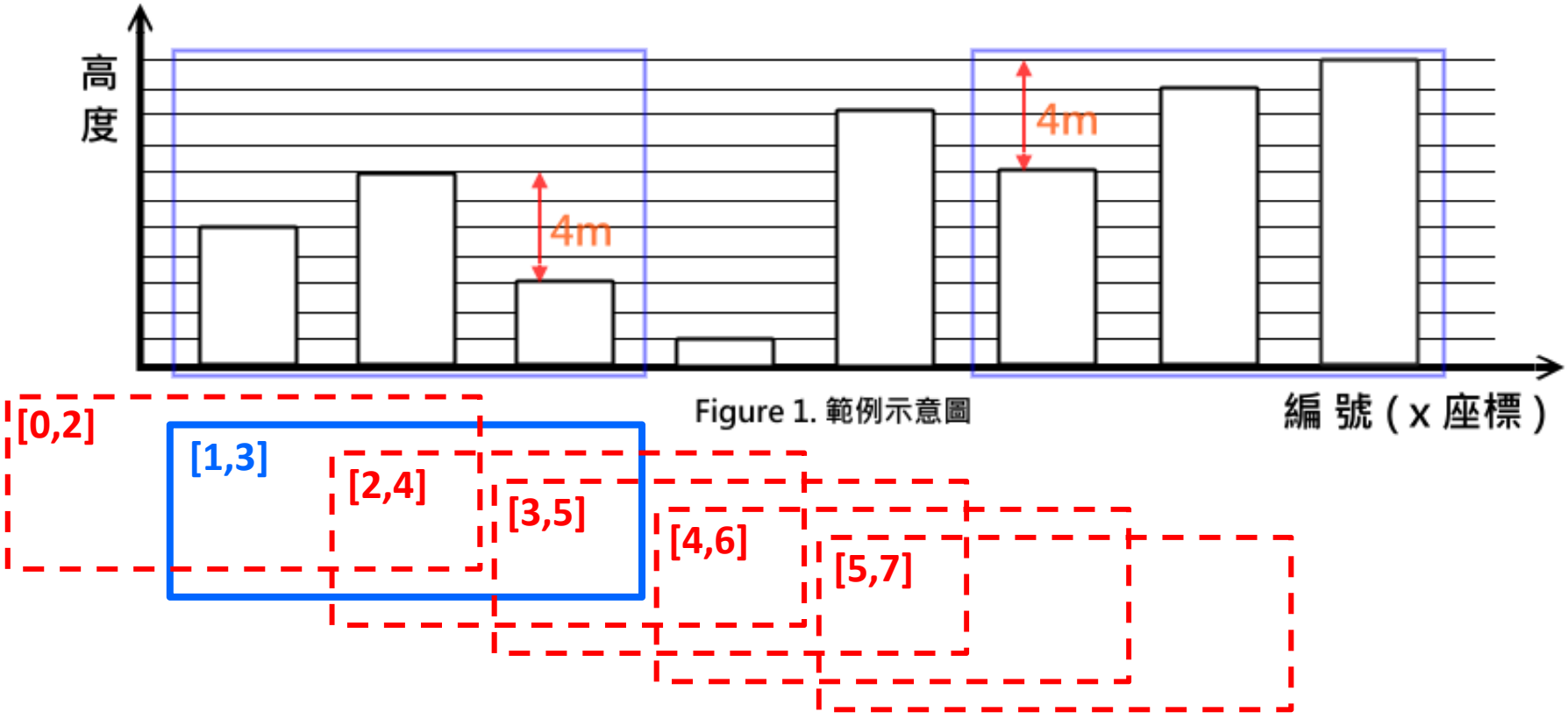


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

$N=8, M=3, K=4$

$H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11$

$2 \leqq N \leqq 10^7, 2 \leqq M \leqq 10^6, 1 \leqq K, H_i \leqq 2^{31}$

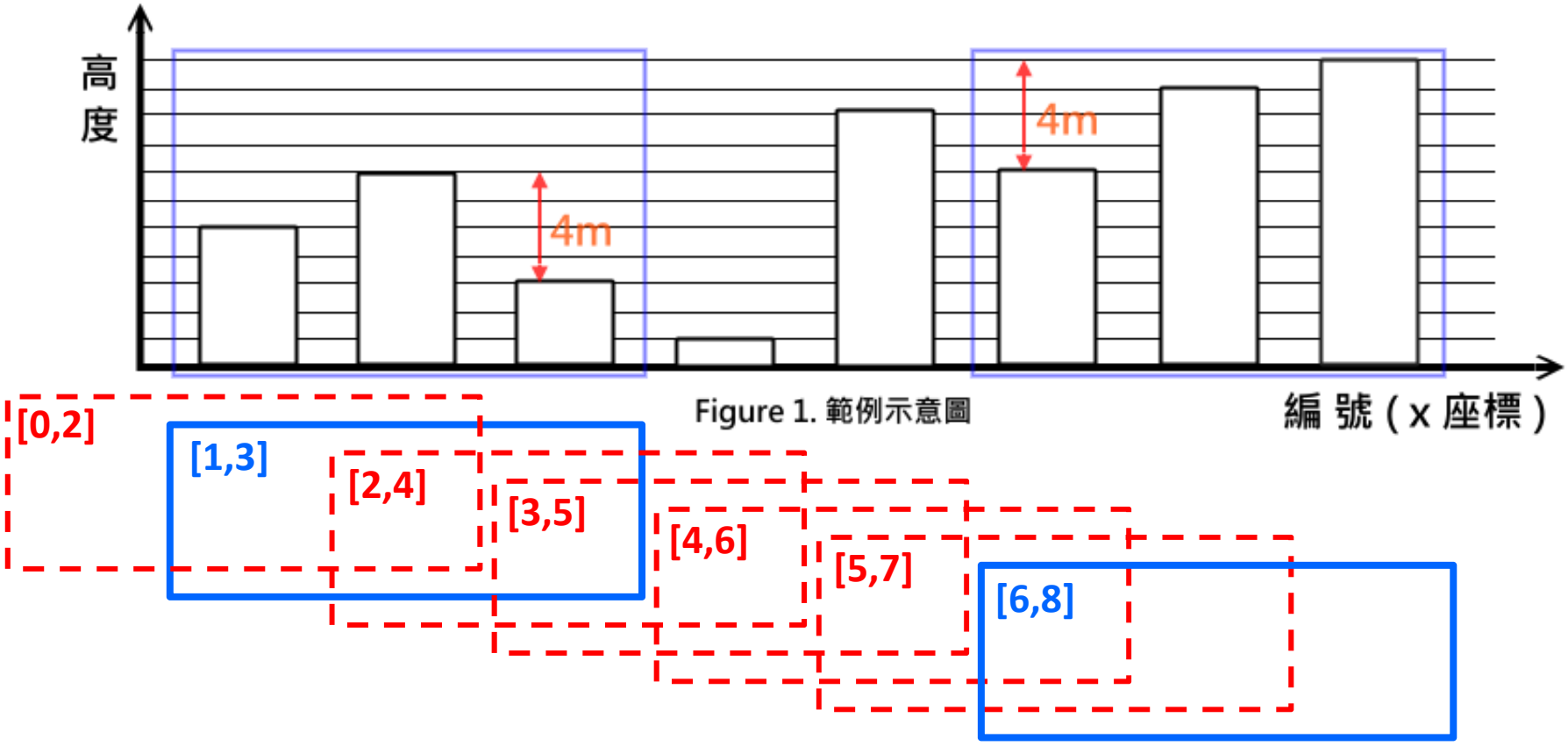


Figure 1. 範例示意圖

- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

$N=8, M=3, K=4$

$H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11$

$2 \leqq N \leqq 10^7, 2 \leqq M \leqq 10^6, 1 \leqq K, H_i \leqq 2^{31}$

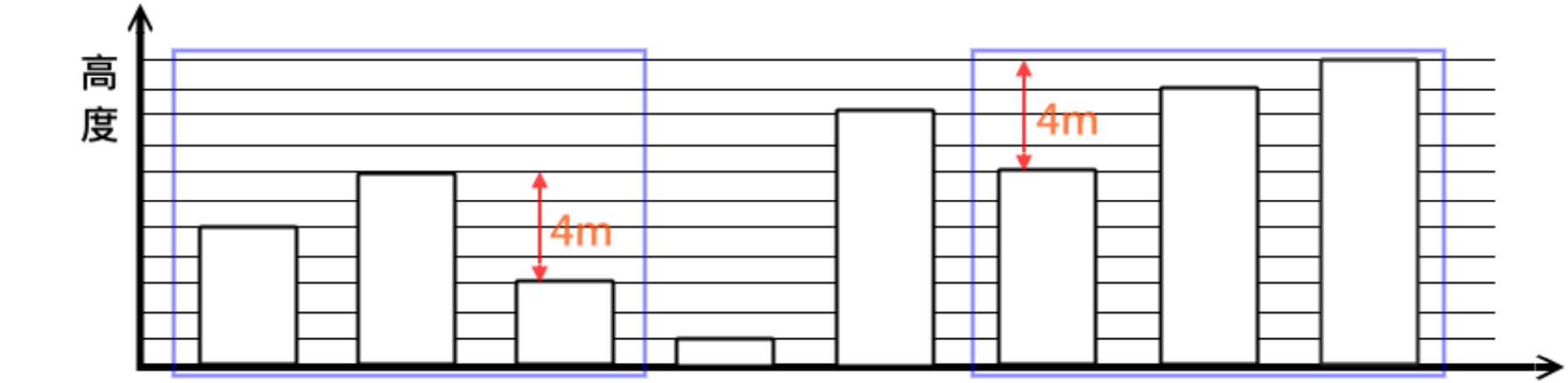
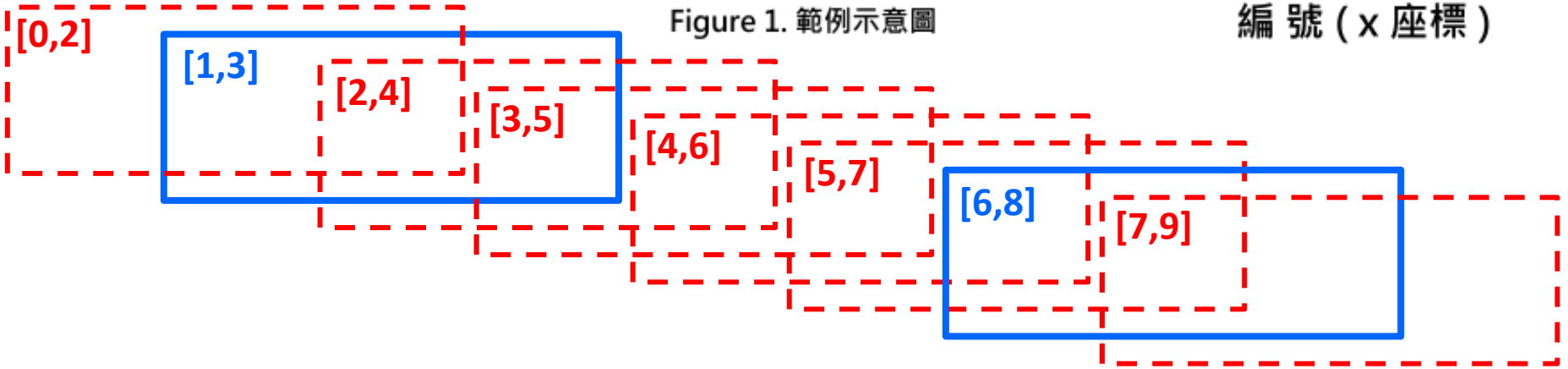


Figure 1. 範例示意圖



- 例如: 共有 **8** 棟大樓, 每張照片可以容納 **3** 棟,
  想要找高低差為 **4** 的取景地點

  N=8, M=3, K=4

  H_1=5, H_2=7, H_3=3, H_4=1, H_5=9, H_6=7, H_7=10, H_8=11

  $2 \leqq N \leqq 10^7$, $2 \leqq M \leqq 10^6$, $1 \leqq K$, $H_i \leqq 2^{31}$

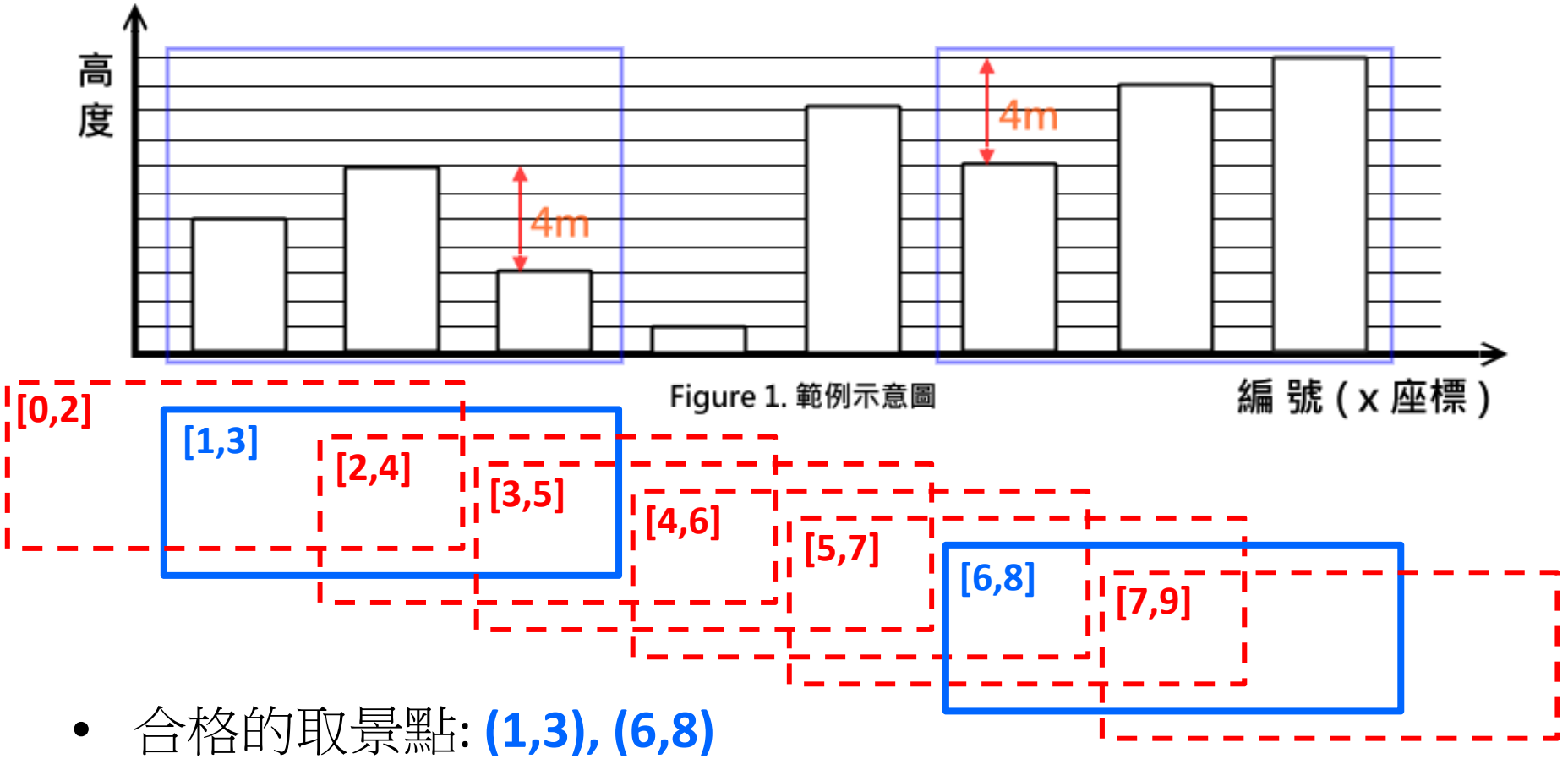


Figure 1. 範例示意圖

- 合格的取景點: **(1,3), (6,8)**

基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

```
for (left=0; left<N; left++) {
   for (max=0,min=MAX+1,i=0; i<M; i++) {
      if (height[left+i]>max) max = height[left+i];
      if (height[left+i]<min) min = height[left+i];
   }
   if (max-min==K) printf("%d %d\n", left, left+M-1);
}
```

基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

```
for (left=0; left<N; left++) {
   for (max=0,min=MAX+1,i=0; i<M; i++) {
      if (height[left+i]>max) max = height[left+i];
      if (height[left+i]<min) min = height[left+i];
   }
   if (max-min==K) printf("%d %d\n", left, left+M-1);
}
```

- 運算時間: O(NM), $N \sim 10^7$, $M \sim 10^6$

基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

```
for (left=0; left<N; left++) {
   for (max=0,min=MAX+1,i=0; i<M; i++) {
      if (height[left+i]>max) max = height[left+i];
      if (height[left+i]<min) min = height[left+i];
   }
   if (max-min==K) printf("%d %d\n", left, left+M-1);
}
```

- 運算時間: O(NM), $N \sim 10^7$, $M \sim 10^6$
- 每秒10^9運算 (~3hr)

基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

```
for (left=0; left<N; left++) {
   for (max=0,min=MAX+1,i=0; i<M; i++) {
      if (height[left+i]>max) max = height[left+i];
      if (height[left+i]<min) min = height[left+i];
   }
   if (max-min==K) printf("%d %d\n", left, left+M-1);
}
```

- 運算時間: O(NM), $N \sim 10^7$, $M \sim 10^6$
- 每秒10^9運算 (~3hr)
- **兩個相鄰的取景視窗有M-1棟大樓重疊, 需要重複檢查這些重疊的大樓嗎?**

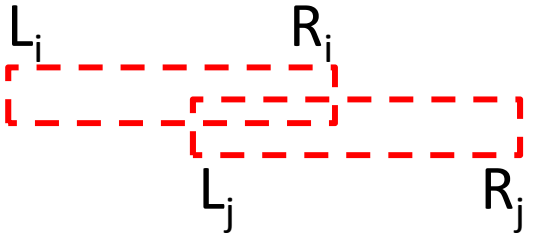
基本作法

- 枚舉所有可能的取景位置 0~N-1, 針對連續M棟大樓檢查是否最大值與最小值相差K

```
for (left=0; left<N; left++) {
   for (max=0,min=MAX+1,i=0; i<M; i++) {
      if (height[left+i]>max) max = height[left+i];
      if (height[left+i]<min) min = height[left+i];
   }
   if (max-min==K) printf("%d %d\n", left, left+M-1);
}
```

- 運算時間: O(NM), $N \sim 10^7$, $M \sim 10^6$
- 每秒10^9運算 (~3hr)
- **兩個相鄰的取景視窗有M-1棟大樓重疊, 需要重複檢查這些重疊的大樓嗎?**
- 希望有一個 O(N) 的計算方法, 可不可以用一些記憶體來換取時間呢?

單調隊列優化



- 主要用在取極值的動作：有時候我們必須針對一個序列的好幾個重疊區段 $[L_i,R_i]$ 取極值，而若這些區段是單調遞增的(即若 $L_i \le L_j$, 則 $R_i \le R_j$)，則我們可以不用每次都作 O(區段長度) 查詢。假設序列長度 N (查詢N個區段)、區段長度 M，我們可以在 O(N+M) 的時間內回答所有查詢。
- 此優化利用我們查詢的左界 L_i 與右界 R_i 為單調遞增之特性 - 我們可以有下列推論：
  1. 一元素從右界進入後必定只會從左界出去。
  2. 由於 1，越左邊之元素一定會越先出去。
  3. 由於 2，若區間中一較右元素之值比一較左元素之值極端，則較左元素再也不會是極值。

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque**前端**把離開區間的元素拿掉
2. 由deque**後端**把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

d.push_back(Elem(5,1))


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
d.push_back(Elem(5,1))

while (d.back().h<7) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque**前端**把離開區間的元素拿掉
2. 由deque**後端**把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
d.push_back(Elem(5,1))

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,2))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
d.push_back(Elem(5,1))

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,2))

while (d.back().h<3) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
d.push_back(Elem(5,1))

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,2))

while (d.back().h<3) d.pop_back()
d.push_back(Elem(3,3))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
d.push_back(Elem(5,1))

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,2))

while (d.back().h<3) d.pop_back()
d.push_back(Elem(3,3))

printf("%d\n", d.front().h) // max is 7
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 [7 3 1] 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque**前端**把離開區間的元素拿掉
2. 由deque**後端**把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
if (d.front().i<2) d.pop_front()
```


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 | **7 3 1** | **9 7 10 11**

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<2) d.pop_front()

while (d.back().h<1) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<2) d.pop_front()

while (d.back().h<1) d.pop_back()
d.push_back(Elem(1,4))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

7,2	3,3	1,4

```
if (d.front().i<2) d.pop_front()

while (d.back().h<1) d.pop_back()
d.push_back(Elem(1,4))

printf("%d\n", d.front()) // max is 7
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

if (d.front().i<3) d.pop_front()


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
if (d.front().i<3) d.pop_front()

while (d.back().h<9) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<3) d.pop_front()

while (d.back().h<9) d.pop_back()
d.push_back(Elem(9,5))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<3) d.pop_front()

while (d.back().h<9) d.pop_back()
d.push_back(Elem(9,5))

printf("%d\n", d.front()) // max is 9
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque**前端**把離開區間的元素拿掉
2. 由deque**後端**把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

if (d.front().i<4) d.pop_front()


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<4) d.pop_front()

while (d.back().h<7) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<4) d.pop_front()

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,6))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<4) d.pop_front()

while (d.back().h<7) d.pop_back()
d.push_back(Elem(7,6))

printf("%d\n", d.front()) // max is 9
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

if (d.front().i<5) d.pop_front()


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
if (d.front().i<5) d.pop_front()

while (d.back().h<10) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<5) d.pop_front()

while (d.back().h<10) d.pop_back()
d.push_back(Elem(10,7))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<5) d.pop_front()

while (d.back().h<10) d.pop_back()
d.push_back(Elem(10,7))

printf("%d\n", d.front()) // max is 10
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque**前端**把離開區間的元素拿掉
2. 由deque**後端**把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

if (d.front().i<6) d.pop_front()


```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素

```
if (d.front().i<6) d.pop_front()

while (d.back().h<11) d.pop_back()
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<6) d.pop_front()

while (d.back().h<11) d.pop_back()
d.push_back(Elem(11,8))
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入

- 例如求長度 M=3 的區段的最大值

5 7 3 1 9 7 10 11

- 利用一個 deque 實作, deque 裡面最多只放 M=3 個元素, deque 最前端維持最大的元素


```
if (d.front().i<6) d.pop_front()

while (d.back().h<11) d.pop_back()
d.push_back(Elem(11,8))

printf("%d\n", d.front()) // max is 11
```

```
struct Elem {
   Elem(long long h1, int i1):
      h(h1), i(i1) {}
   long long h;
   int i;
};
```

deque: 兩端可以 push/pop 的資料結構

1. 由deque前端把離開區間的元素拿掉
2. 由deque後端把小於進入區間元素的拿掉, 把進入區間的元素放入