排序 2

排序就是將資料由小到大或由大到小排列，常見排序演算法有氣泡排序、選擇排序、插入排序、合併排序與快速排序等，其中以合併排序與快速排序的演算法效率比較好，但程式也較複雜。

2-1 ▸▸ 氣泡排序（BubbleSort）

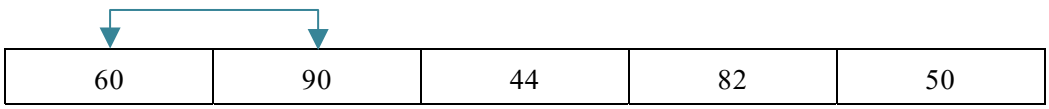
隨機產生一個陣列五個元素，以氣泡排序演算法將這五個元素由小到大排序。

氣泡排序舉例說明

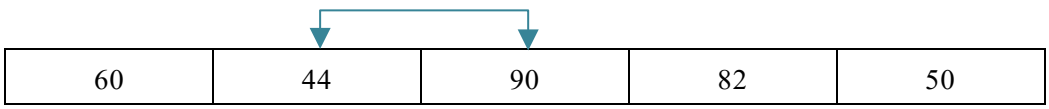
假設隨機產生五個陣列元素，如下圖。

60	90	44	82	50

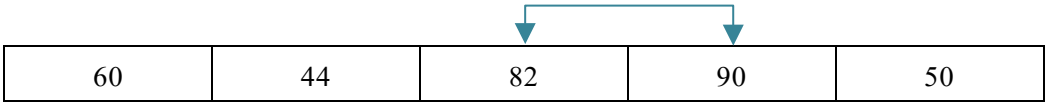
(1) 比較第一個元素 (60) 與第二個 (90) 元素，若第一個元素比第二個元素大，則第一個元素與第二個元素交換。



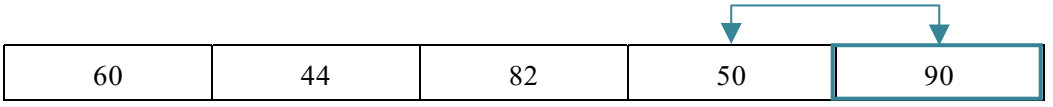
(2) 再比較第二個元素 (90) 與第三個元素 (44)，若第二個元素比第三個元素大，則第二個元素與第三個元素交換，目前第三個元素 (90) 為三個元素最大的。



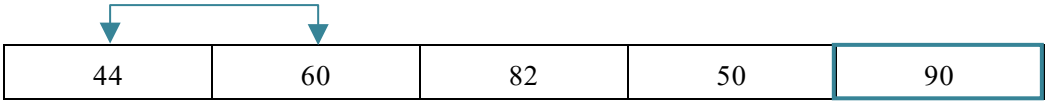
(3) 再比較第三個元素 (90) 與第四個元素 (82)，若第三個元素比第四個元素大，則第三個元素與第四個元素交換，目前第四個元素 (90) 為四個元素最大的。



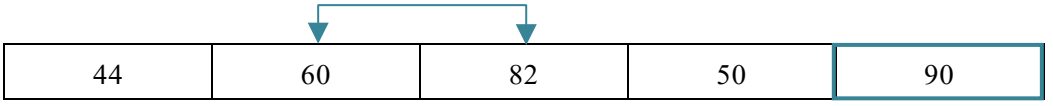
(4) 再比較第四個元素 (90) 與第五個元素 (50)，若第四個元素比第五個元素大，則第四個元素與第五個元素交換，目前第五個元素 (90) 為五個元素最大的，如此我們已經將最大元素放到陣列最後一個元素。



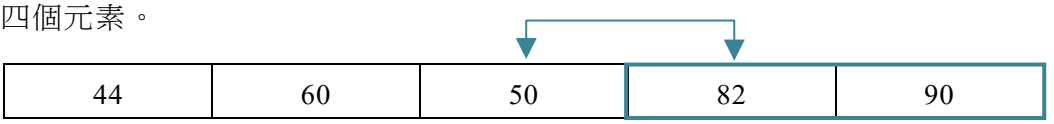
(5) 依此類推，將範圍改成第一到第四個元素，比較第一個元素 (60) 與第二個 (44) 元素，若第一個元素比第二個元素大，則第一個元素與第二個元素交換。



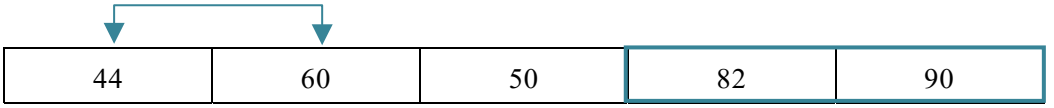
(6) 比較第二個元素 (60) 與第三個 (82) 元素，若第二個元素比第三個元素大，則第二個元素與第三個元素交換。



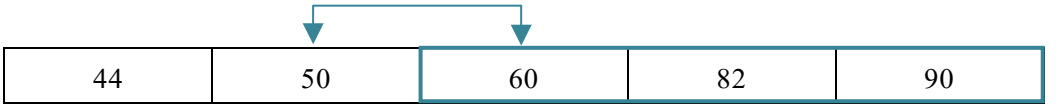
(7) 比較第三個元素 (82) 與第四個 (50) 元素，若第三個元素比第四個元素大，則第三個元素與第四個元素交換，照上述方式可以將四個中最大元素放到陣列第四個元素。



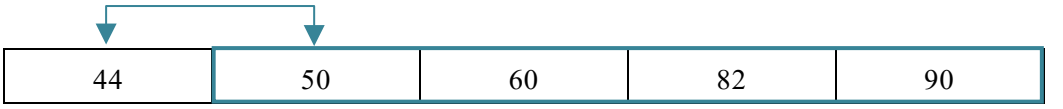
(8) 依此類推，將範圍改成第一到第三個元素，比較第一個元素 (44) 與第二個 (60) 元素，若第一個元素比第二個元素大，則第一個元素與第二個元素交換。



(9) 比較第二個元素 (60) 與第三個 (50) 元素，若第二個元素比第三個元素大，則第二個元素與第三個元素交換，照上述方式可以將三個中最大元素放到陣列第三個元素。



(10) 依此類推，將範圍改成第一到第二個元素，比較第一個元素 (44) 與第二個 (50) 元素，若第一個元素比第二個元素大，則第一個元素與第二個元素交換，照上述方式可以將兩個中最大元素放到陣列第二個元素，範圍內只有一個元素就不用排序了，到此已完成排序。



氣泡排序演算法程式碼【2-1-氣泡排序.cpp】

(a) 程式碼與解說

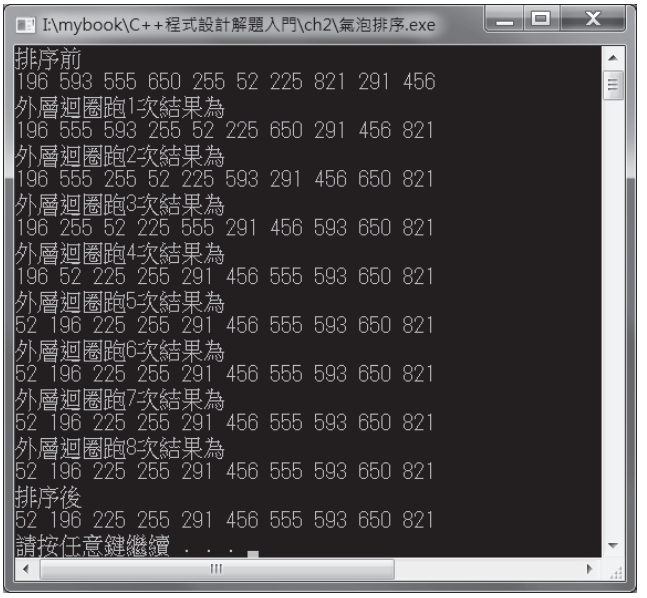
```
1     #include <iostream>
2     #include <ctime>
3     #include <cstdlib>
4     #define Size 10
5     using namespace std;
6     void printA(int *a,int s){
7       for(int i=0;i<s;i++){
8         cout << a[i] << " ";
9       }
10      cout << endl;
11    }
12    int main(){
13      int A[Size],tmp;
14      srand(time(NULL));
15      for (int i=0;i<Size;i++){
16        A[i]=rand()%1000+1;
17      }
18      cout << "排序前" << endl;
19      printA(A,Size);
20      for(int i=Size-1;i>1;i--){
21        for(int j=0;j<i;j++){
22          if (A[j]>A[j+1]){
23            tmp=A[j];
```

```
24              A[j]=A[j+1];
25              A[j+1]=tmp;
26            }
27          }
28          cout << "外層迴圈跑" << Size-i <<"次結果為" << endl;
29          printA(A,Size);
30        }
31        cout << "排序後" << endl;
32        printA(A,Size);
33      }
```

- 第 4 行：定義 Size 為 10。
- 第 6 行到第 11 行：定義函式 printA，使用迴圈印出陣列 a 的所有元素。
- 第 13 行：宣告 10 個元素的整數陣列 A，宣告 tmp 為整數變數。
- 第 14 行：初始化隨機函式。
- 第 15 到 17 行：使用 for 迴圈隨機產生陣列 A 元素的值，其值介於 1 到 1000 的整數。
- 第 18 行：顯示「排序前」。
- 第 19 行：顯示陣列 A 的元素。
- 第 20 到 30 行：氣泡排序演算法，外層迴圈變數 i，控制內層迴圈變數 j 的上限，迴圈變數 i 由 9 到 2，每次遞減 1，內層迴圈 j 由 0 到（i-1），每次遞增 1，第 22 到 26 行比較相鄰兩數，前面比後面大就交換，第 23 到 25 行表示交換兩數。
- 第 28 行：顯示「外層迴圈跑 x 次結果為」。
- 第 29 行：呼叫 printA 顯示陣列所有元素。
- 第 31 行：顯示「排序後」。
- 第 32 行：呼叫 printA 顯示排序後陣列所有元素。

(b) 預覽結果

按下「執行→編譯並執行」，螢幕顯示結果如下圖。



氣泡排序演算法效率

假設要排序 n 個資料，程式中第 20 行到第 30 行為氣泡排序演算法，外層迴圈 i 數值由 n-1 到 1，每次遞減 1，外層每執行一次內層執行 i 次，內層迴圈的執行總次數影響整個程式的執行效率，累加內層迴圈的執行次數為「(n-1)+(n-2)+(n-3)+…+2」，總次數接近「$\frac{n^2}{2}$」，演算法效率為 $O(n^2)$。

2-2 ▸▸ 插入排序（InsertionSort）

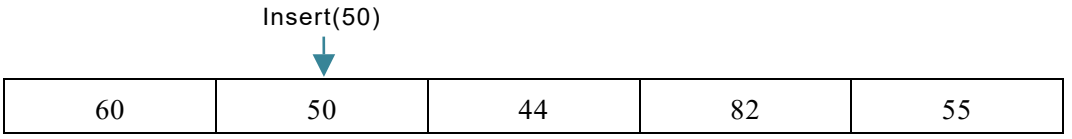
隨機產生一個陣列五個元素，以插入排序演算法將這五個元素由小到大排序。

插入排序演算法舉例說明

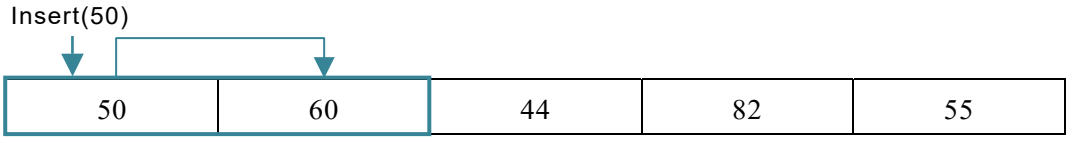
假設隨機產生五個陣列元素，如下圖。

60	50	44	82	55

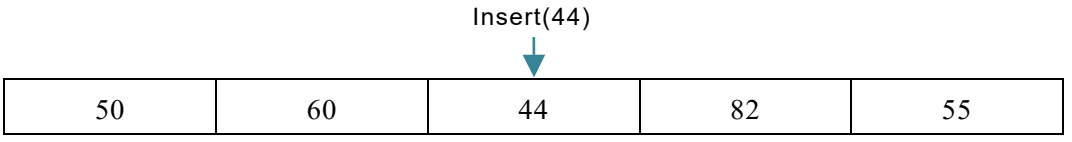
(1) 初始化變數 insert 為第二個元素 (50)，將變數 insert(50) 插入到陣列中，讓第一個元素與第二個元素成為已排序狀態。



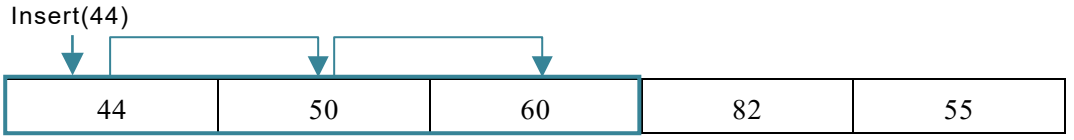
(2) 因為第一個元素 (60) 比變數 insert(50) 大，第一個元素 (60) 移到第二個元素，將變數 insert(50) 放到第一個元素，這樣第一個與第二個元素就變成已排序狀態。



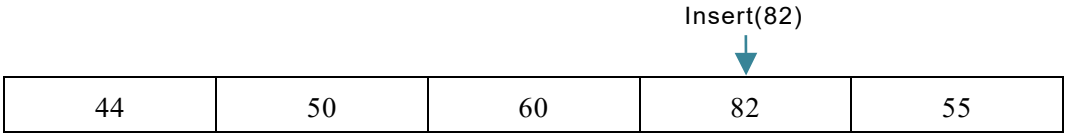
(3) 初始化變數 insert 為第三個元素 (44)，將變數 insert(44) 插入到陣列中，讓第一個元素到第三個元素成為已排序狀態。



(4) 因為第二個元素 (60) 比變數 insert(44) 大，第二個元素 (60) 移到第三個元素，因為第一個元素 (50) 比變數 insert(44) 大，第一個元素 (50) 移到第二個元素，將變數 insert(44) 放到第一個元素，這樣第一個元素到第三個元素就變成已排序狀態。

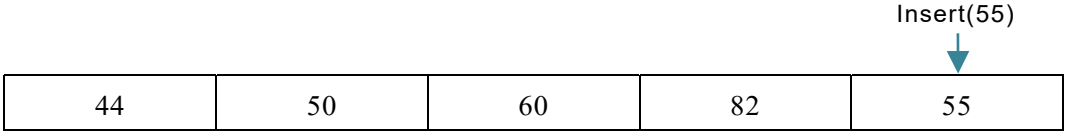


(5) 初始化變數 insert 為第四個元素 (82)，將變數 insert (82) 插入到陣列中，讓第一個元素到第四個元素成為已排序狀態。

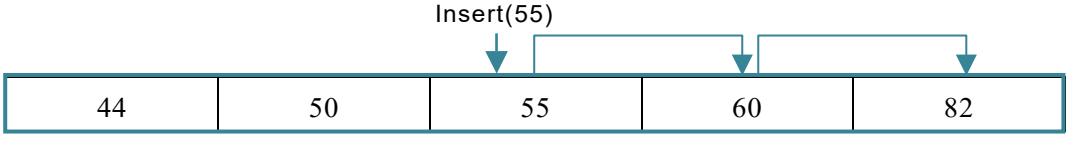


(6) 因為第三個元素 (60) 比變數 insert(82) 小，將變數 insert(82) 放到第四個元素，這樣第一個元素到第四個元素就變成已排序狀態。

(7) 初始化變數 insert 為第五個元素 (55)，將變數 insert (55) 插入到陣列中，讓第一個元素到第五個元素成為已排序狀態。



(8) 因為第四個元素 (82) 比變數 insert(55) 大，第四個元素 (82) 移到第五個元素，因為第三個元素 (60) 比變數 insert(55) 大，第三個元素 (60) 移到第四個元素，最後將變數 insert(55) 放到第三個元素，這樣第一個元素到第五個元素就變成已排序狀態。



插入排序演算法程式碼【2-2-插入排序.cpp】

(a) 程式碼與解說

```
1     #include <iostream>
2     #include <ctime>
3     #include <cstdlib>
4     #define Size 10
5     using namespace std;
6     void printA(int *a,int s){
7       for(int i=0;i<s;i++){
8         cout << a[i] << " ";
9       }
10      cout << endl;
11    }
12    int main() {
13      int A[Size],insert;
14      srand(time(NULL));
15      for (int i=0;i<Size;i++){
```

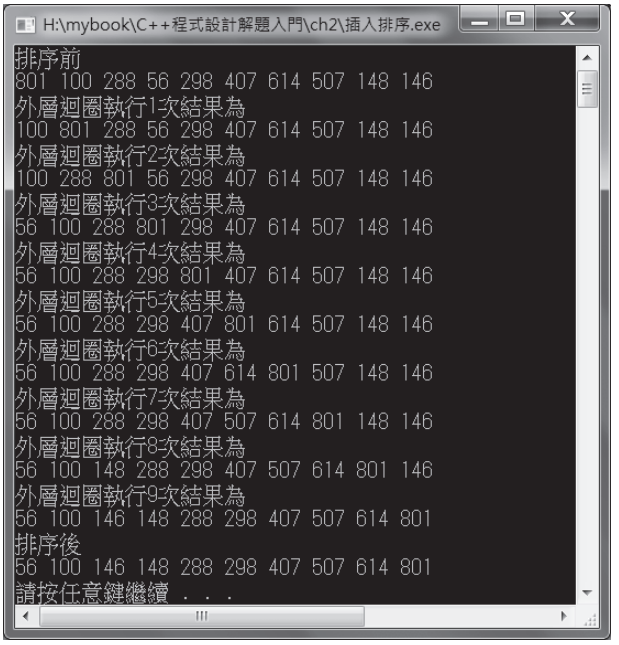
```
16        A[i]=rand()%1000+1;
17      }
18      cout << "排序前" << endl;
19      printA(A,Size);
20      int j;
21      for(int i=1;i<Size;i++){
22        insert = A[i];
23        for(j=i-1;j>=0;j--){
24          if(insert < A[j]) {
25            A[j+1]=A[j];
26          }else {
27            break;
28          }
29        }
30        A[j+1]=insert;
31        cout << "外層迴圈執行" << i <<"次結果為" << endl;
32        printA(A,Size);
33      }
34      cout << "排序後" << endl;
35      printA(A,Size);
36    }
```

- 第 4 行：定義 Size 為 10。
- 第 6 行到第 11 行：定義函式 printA，使用迴圈印出陣列 a 的所有元素。
- 第 12 行到第 36 行：定義 main 函式。
- 第 13 行：宣告 10 個元素的整數陣列 A，宣告 insert 為整數變數。
- 第 14 行：初始化隨機函式。
- 第 15 到 17 行：使用 for 迴圈隨機產生陣列 A 元素的值，其值介於 1 到 1000 的整數。
- 第 18 行：顯示「排序前」。
- 第 19 行：顯示陣列 A 的元素。
- 第 20 行：宣告變數 j 為整數變數。
- 第 20 到 33 行：插入排序演算法，外層迴圈變數 i，控制內層迴圈變數 j 的初始值，外層迴圈變數 i 由 1 到 Size-1，每次遞增 1，初始化變數 insert 為 A[i]，內層迴圈 j 由 i-1 到 0，每次遞減 1，若索引值 j 所指定的元素值較變數 insert 大，則將 A[j]複製到 A[j+1]；否則中斷內層迴圈（第 27 行）。
- 第 30 行：將變數 insert 儲存到 A[j+1]。
- 第 31 行：顯示「外層迴圈執行 x 次結果為」

- 第 32 行：呼叫 printA 顯示陣列所有元素。
- 第 34 行：顯示「排序後」。
- 第 35 行：呼叫 printA 顯示排序後陣列所有元素。

(b) 預覽結果

按下「執行→編譯並執行」，螢幕顯示結果如下圖。



插入排序演算法效率

假設要排序 n 個資料，程式中第 20 行到第 33 行為插入排序演算法，外層迴圈 i 數值由 1 到 n-1，每次遞增 1，外層每執行一次內層執行 i 次，內層迴圈的執行總次數影響整個程式的執行效率，累加內層迴圈的執行次數為「1+2+3+…+(n-2)+(n-1)」，總次數接近「$\frac{n^2}{2}$」，演算法效率為 $O(n^2)$。

2-3 ▸▸ 合併排序（MergeSort）

隨機產生一個陣列八個元素，以合併排序演算法將這八個元素由小到大排序。

合併排序演算法舉例說明

假設隨機產生八個數字放置於陣列中，如下圖。

60	50	44	82	55	24	99	33

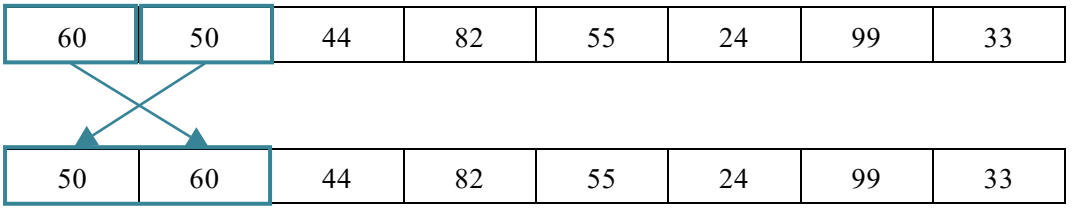
(1) 排序第 1 個元素到第 8 個元素，排序左半部（第 1 個元素到第 4 個元素）與右半部（第 5 個元素到第 8 個元素），最後將左右兩邊資料合併完成排序，首先執行排序左半部（第 1 個元素到第 4 個元素）。

60	50	44	82	55	24	99	33

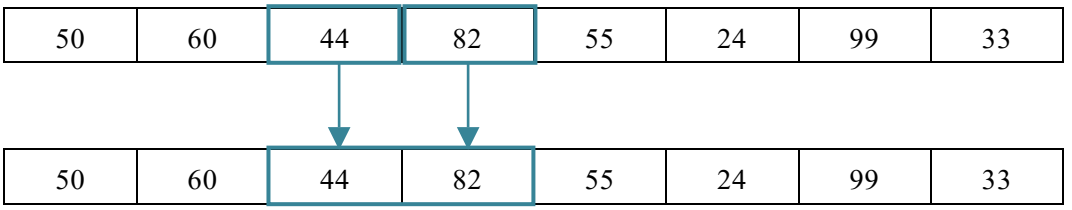
(2) 排序第 1 個元素到第 4 個元素，排序左半部（第 1 個元素到第 2 個元素）與右半部（第 3 個元素到第 4 個元素），最後將左右兩邊資料合併完成排序，首先執行排序左半部（第 1 個元素到第 2 個元素）。

60	50	44	82	55	24	99	33

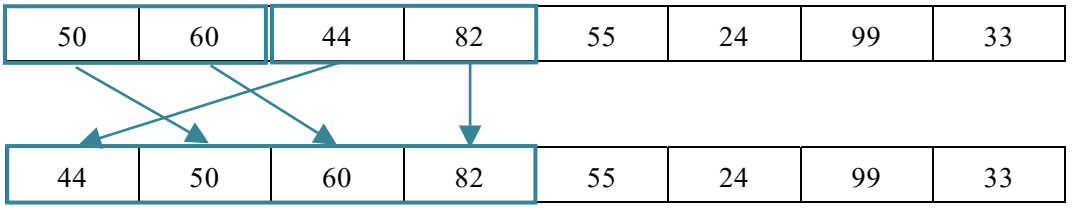
(3) 排序第 1 個元素到第 2 個元素，排序左半部（第 1 個元素到第 1 個元素）與右半部（第 2 個元素到第 2 個元素），因為左右兩邊都只剩下一個元素，左右兩邊都已經排序完成，將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 1 個元素到第 2 個元素的排序。



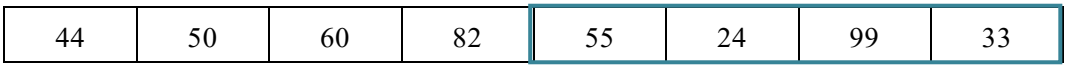
(4) 現在執行 (2) 排序第 3 個元素到第 4 個元素，排序左半部（第 3 個元素到第 3 個元素）與右半部（第 4 個元素到第 4 個元素），因為左右兩邊都只剩下一個元素，左右兩邊都已經排序完成，將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 3 個元素到第 4 個元素的排序。



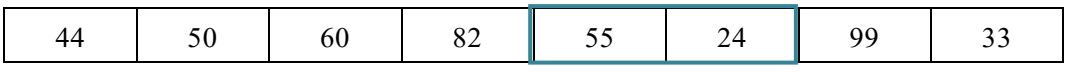
(5) 在 (2) 排序第 1 個元素到第 4 個元素，先排序左半部（第 1 個元素到第 2 個元素）與右半部（第 3 個元素到第 4 個元素），到此完成左右兩邊的排序，現在合併左右兩邊資料完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 1 個元素到第 4 個元素的排序。



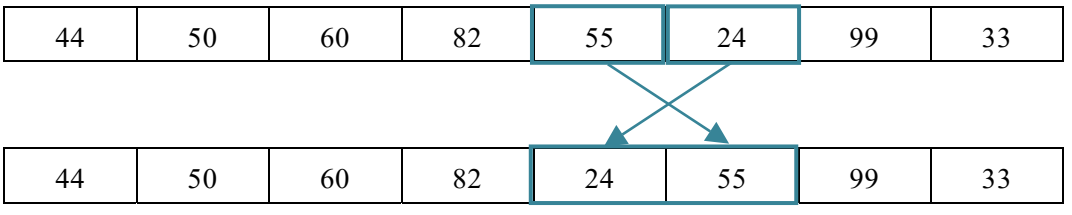
(6) 現在執行 (1) 排序右半部（第 5 個元素到第 8 個元素），最後將左右兩邊資料合併完成排序。



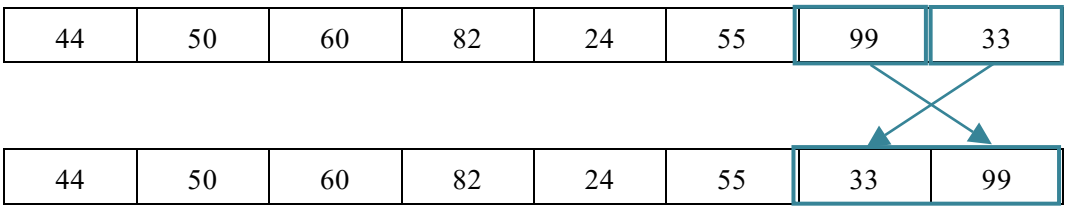
(7) 排序第 5 個元素到第 8 個元素，排序左半部（第 5 個元素到第 6 個元素）與右半部（第 7 個元素到第 8 個元素），最後將左右兩邊資料合併完成排序，首先執行排序左半部（第 5 個元素到第 6 個元素）。



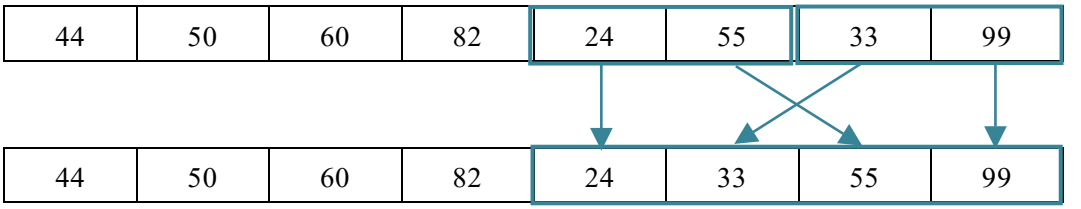
(8) 排序第 5 個元素到第 6 個元素，排序左半部（第 5 個元素到第 5 個元素）與右半部（第 6 個元素到第 6 個元素），因為左右兩邊都只剩下一個元素，左右兩邊都已經排序完成，將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 5 個元素到第 6 個元素的排序。



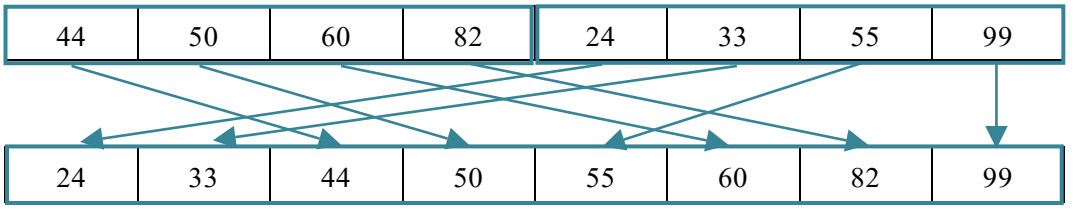
(9) 現在執行 (7) 的排序右半部第 7 個元素到第 8 個元素，排序左半部（第 7 個元素到第 7 個元素）與右半部（第 8 個元素到第 8 個元素），因為左右兩邊都只剩下一個元素，左右兩邊都已經排序完成，將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 7 個元素到第 8 個元素的排序。



(10) 在(7)排序第 5 個元素到第 8 個元素，現在完成排序左半部（第 5 個元素到第 6 個元素）與右半部（第 7 個元素到第 8 個元素），現在將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成第 5 個元素到第 8 個元素的排序。



(11) 在(1)排序第 1 個元素到第 8 個元素，現在完成排序左半部（第 1 個元素到第 4 個元素）與右半部（第 5 個元素到第 8 個元素），現在將左右兩邊資料合併完成排序，比較兩邊最小的元素，較小的放在前面，合併過程完成排序。



合併排序演算法程式碼【2-3-合併排序.cpp】

(a) 程式碼與解說

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#define Size 10
using namespace std;
void merge(int *,int,int,int);
void mergesort(int *,int,int);
void printA(int *a,int s){
  for(int i=0;i<s;i++){
    cout << a[i] << " ";
  }
  cout << endl;
}
void mergesort(int *a,int L,int R){
  int M;
  if(L<R){
    M=(L+R)/2;
    mergesort(a,L,M);
    mergesort(a,M+1,R);
    merge(a,L,M,R);
    cout <<"L="<<L<<" M="<<M<<" R="<<R<<endl;
    printA(a,Size);
  }
}
void merge(int *a,int L,int M,int R){
  int left,right,i,tmp[Size];
  left=L;
  right=M+1;
  i=L;
  while((left<=M)&&(right<=R)){
    if(a[left]<a[right]){
      tmp[i]=a[left];
      i++,left++;
    }else{
      tmp[i]=a[right];
      i++,right++;
    }
  }
  while(left<=M){
    tmp[i]=a[left];
    i++,left++;
  }
  while(right<=R){
```

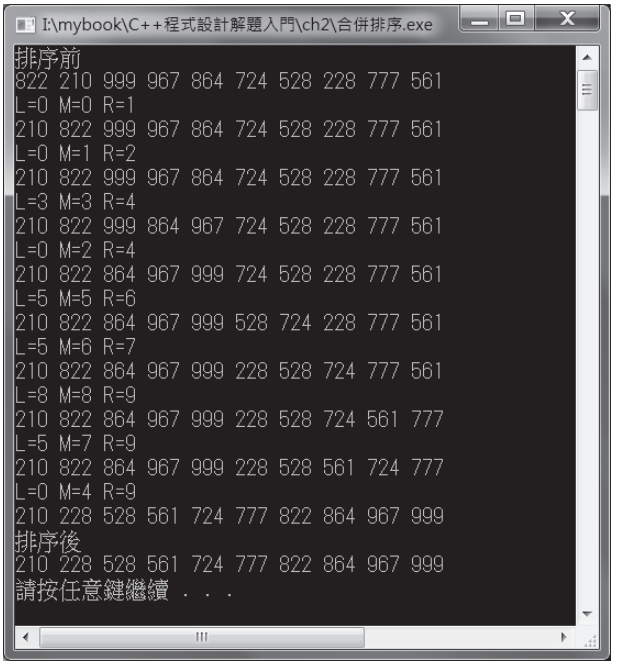
```
44          tmp[i]=a[right];
45          i++,right++;
46        }
47        for(i=L;i<=R;i++){
48          a[i]=tmp[i];
49        }
50      }
51      int main() {
52        int A[Size];
53        srand(time(NULL));
54        for (int i=0;i<Size;i++){
55          A[i]=rand()%1000+1;
56        }
57        cout << "排序前" << endl;
58        printA(A,Size);
59        mergesort(A,0,Size-1);
60        cout << "排序後" << endl;
61        printA(A,Size);
62      }
```

- 第 4 行：定義 Size 為 10。
- 第 6 行：宣告 merge 函式，用於合併左右兩邊的已排序陣列。
- 第 7 行：宣告 mergesort 函式，用於將陣列切割成左右兩半分別排序。
- 第 8 行到第 13 行：定義函式 printA，使用迴圈印出陣列 a 的所有元素。
- 第 14 行到第 24 行：定義 mergesort，輸入參數有陣列 a、左半部索引值 L 與右半部索引值 R。
- 第 15 行：宣告整數變數 M。
- 第 16 行到第 23 行：若左半部索引值 L 小於右半部索引值 R，則令索引值 M 為 L 與 R 除以 2。呼叫函式 mergesort(a,L,M) 切割成左半部先排序，呼叫函式 mergesort(a,M+1,R) 切割成右半部先排序，最後經由呼叫 merge(a,L,M,R) 將左右兩邊已排序陣列元素，合併成一個已排序陣列。顯示變數 L、變數 M 與變數 R 的數值，列印陣列所有元素，用於顯示執行合併排序的過程(第 21 到 22 行）。
- 第 25 到 50 行：定義 merge 函式，輸入的參數有要排序的陣列 a，左邊界索引值 L、中間索引值 M 與右邊界索引值 R。
- 第 26 行：宣告整數變數 left（目前左半部執行到第幾個元素）、right（目前右半部執行到第幾個元素）、i（合併後的暫存陣列 tmp 的索引值）與合併過程的暫存陣列 tmp，有 Size 個元素。

- 第 27 行：變數 left 初始化為 L，因為左半部的目前合併元素 left 從左邊界索引值 L 開始。
- 第 28 行：變數 right 初始化為 M+1，因為右半部的目前合併元素 right 從右邊界索引值 M+1 開始。
- 第 29 行：變數 i 初始化為 L，因為暫存陣列 tmp 的索引值 i 從左邊界索引值 L 開始。
- 第 30 行到第 38 行：當 left 小於等於 M 且 right 小於等於 R，表示左右兩半部都還有元素可以合併，繼續執行合併動作。合併動作為當陣列 a 的索引值 left 元素小於陣列 a 的索引值 right，將左半部元素放入陣列 tmp 第 i 個元素位置（第 32 行），變數 i 與 left 都遞增 1（第 33 行）；否則將右半部元素放入陣列 tmp 第 i 個元素位置（第 35 行），變數 i 與 right 都遞增 1（第 36 行）
- 第 39 行到第 42 行：若 left 小於等於 M，表示左半部還有元素需要放入陣列 tmp，而右半部已經全部放入，將左半部剩餘元素依序放入陣列 tmp。
- 第 43 行到第 46 行：若 right 小於等於 R，表示右半部還有元素需要放入陣列 tmp，而左半部已經全部放入，將右半部剩餘元素依序放入陣列 tmp。
- 第 47 行到第 49 行：將陣列 tmp 的第 L 個元素到第 R 個元素複製給陣列 a 的第 L 個元素到第 R 個元素。
- 第 51 行到第 62 行：定義 main 函式。
- 第 52 行：宣告 Size 個元素的整數陣列 A。
- 第 53 行：初始化隨機函式。
- 第 54 到 55 行：使用 for 迴圈隨機產生陣列 A 元素的值，其值介於 1 到 1000 的整數。
- 第 57 行：顯示「排序前」。
- 第 58 行：顯示陣列 A 的所有元素。
- 第 59 行：呼叫 mergesort 函式，排序陣列 A 所有元素。
- 第 60 行：顯示「排序後」。
- 第 61 行：呼叫 printA 顯示排序後陣列所有元素。

(b) 預覽結果

按下「執行→編譯並執行」，螢幕顯示結果如下圖。



合併排序演算法效率

假設要排序 n 個資料，程式中第 14 行到第 50 行為合併排序演算法，第 18 行到第 19 行的 mergesort 函式每次將資料拆成一半，所以合併排序的 mergesort 的遞迴深度為 O(log(n))，第 20 行的 merge 動作每一層都需要 O(n)，所以合併演算法效率為 O(n log(n))，相較於氣泡排序與插入排序演算法效能較佳，但排序過程須額外使用 O(n)的暫存記憶體空間，所以記憶體空間的使用較氣泡排序與插入排序演算法來的多。

2-8 APCS 排序相關實作題詳解

2-8-1 成績指標（10503 第 1 題）【2-8-1-成績指標.cpp】

問題描述

一次考試中，於所有及格學生中獲取最低分數者最為幸運，反之，於所有不及格同學中，獲取最高分數者，可以說是最為不幸，而此二種分數，可以視為成績指標。

請你設計一支程式，讀入全班成績（人數不固定），對所有分數進行排序，並分別找出不及格中最高分數，以及及格中最低分數。

當找不到最低及格分數時，表示此次考試全班皆不及格，此時請你印出：「worst case」；反之，當找不到最高不及格分數時，請你印出「best case」。

註：假設及格分數為 60，每筆測資皆為 0~100 間整數，且筆數未定。

輸入格式

第一行輸入學生人數，第二行為各學生分數（0~100 間），分數與分數之間以一個空白間格。每一筆測資的學生人數為 1~20 的整數。

輸出格式

- 每筆測資輸出三行。
- 第一行由小而大印出所有成績，兩數字之間以一個空白間格，最後一個數字後無空白。
- 第二行印出最高不及格分數，如果全數及格時，於此行印出 best case。
- 第三行印出最低及格分數，當全數不及格時，於此行印出 worst case。

範例一：輸入

```
10
0 11 22 33 55 66 77 99 88 44
```

範例一：正確輸出

```
0 11 22 33 44 55 66 77 88 99
55
66
```

（說明）不及格分數最高為 55，及格分數最低為 66。

範例二：輸入

```
1
13
```

範例二：正確輸出

```
13
13
worst case
```

（說明）由於找不到最低及格分數，因此第三行須印出「worst case」。

範例三：輸入

```
2
73 65
```

範例三：正確輸出

```
65 73
best case
65
```

（說明）由於找不到不及格分數，因此第二行須印出「best case」。

評分說明

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制（time limit）均為 2 秒，依正確通過測資筆數給分。

(a) 解題想法

使用函式庫 algorithm 的函式 sort 進行成績排序，可以自行定義比較函式 cmp 輸入到函式 sort，讓函式 sort 由小到大排序資料，接著由前往後找尋不及格成績，

就可以找到最高的不及格成績；由後往前找尋及格成績，就可以找到最低的及格成績。

(b) 程式碼與解說

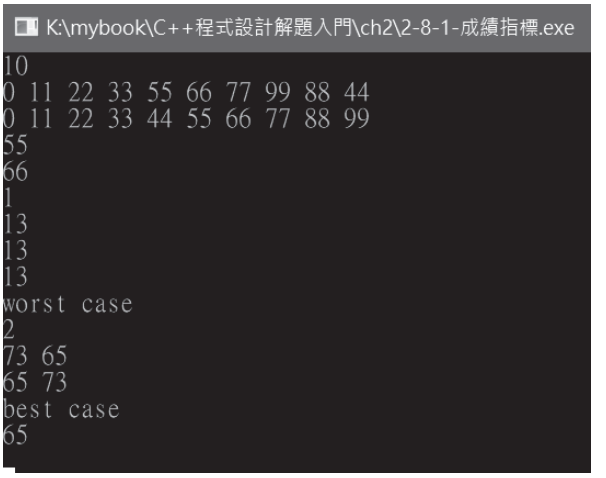
```
1     #include <iostream>
2     #include <algorithm>
3     using namespace std;
4     int cmp(int a, int b){
5       return a<b;
6     }
7     int main(){
8       int n,score[20],a60,b60;
9       while (cin >> n){
10        a60=-1,b60=-1;
11        for(int i=0;i<n;i++){
12          cin >> score[i];
13        }
14        sort(score,score+n,cmp);
15        for(int i=0;i<n;i++){
16          if (score[i]<60) b60=score[i];
17        }
18        for(int i=n-1;i>=0;i--){
19          if (score[i]>=60) a60=score[i];
20        }
21        cout << score[0];
22        for(int i=1;i<n;i++){
23          cout << " " << score[i];
24        }
25        cout << endl;
26        if (b60 != -1) cout << b60 << endl;
27        else cout << "best case" << endl;
28        if (a60 != -1) cout << a60 << endl;
29        else cout << "worst case" << endl;
30      }
31    }
```

- 第 4 到 6 行：自行定義函式 cmp，將此函式輸入函式 sort，會造成數值由小到大排序。
- 第 7 到 31 行：定義函式 main。
- 第 8 行：宣告 n、a60 與 b60 為整數變數，score 為 20 個元素的整數陣列。

- 第 9 到 30 行：不斷輸入成績個數到變數 n，設定變數 a60 初始值為-1，變數 b60 初始值為-1（第 10 行）。
- 第 11 到 13 行：使用迴圈輸入 n 個成績到陣列 score。
- 第 14 行：使用函式 sort 以函式 cmp 為輸入，由小到大排序陣列 score。
- 第 15 到 17 行：使用迴圈由前往後找小於 60 的數值到變數 b60，變數 b60 就是最高的不及格成績。
- 第 18 到 20 行：使用迴圈由後往前找大於等於 60 的數值到變數 a60，變數 a60 就是最低的及格成績。
- 第 21 到 25 行：輸出由小到大已經排序的所有成績。
- 第 26 到 27 行：若 b60 不等於-1，表示 b60 為最高的不及格成績，輸出變數 b60；否則表示找不到最高的不及格成績，則輸出「best case」。
- 第 28 到 29 行：若 a60 不等於-1，表示 a60 為最低的及格成績，輸出變數 a60；否則表示找不到最低的及格成績，則輸出「worst case」。

(c) 預覽結果

按下「執行→編譯並執行」，輸入題目所規定的三組測資後，螢幕顯示結果如下圖。



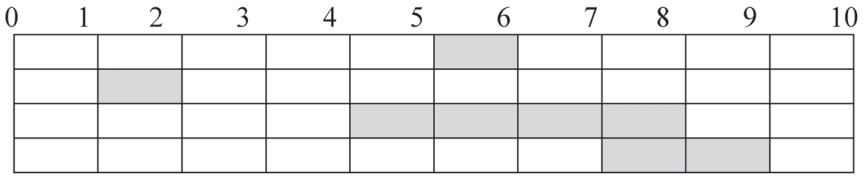
(d) 演算法效率分析

第 14 行排序演算法效率為 O(N*log(N))，N 為成績個數，第 15 到 20 行找尋最高的不及格成績與最低的及格成績演算法效率為 O(N)，本程式演算法效率為 O(N*log(N))。

2-8-2 線段覆蓋長度（10503 第 3 題）【2-8-2-線段覆蓋長度.cpp】

問題描述

給定一維座標上一些線段，求這些線段所覆蓋的長度，注意，重疊的部分只能算一次。例如給定三個線段：(5, 6)、(1, 2)、(4, 8)、和(7, 9)，如下圖，線段覆蓋長度為 6。



輸入格式

第一列是一個正整數 N，表示此測試案例有 N 個線段。

接著的 N 列每一列是一個線段的開始端點座標和結束端點座標整數值，開始端點座標值小於等於結束端點座標值，兩者之間以一個空格區隔。

輸出格式

輸出其總覆蓋的長度。

範例一：輸入

輸入	說明
5	此測試案例有 5 個線段
160 180	開始端點座標值與結束端點座標
150 200	開始端點座標值與結束端點座標
280 300	開始端點座標值與結束端點座標
300 330	開始端點座標值與結束端點座標
190 210	開始端點座標值與結束端點座標

範例一：輸出

輸出	說明
110	測試案例的結果

範例二：輸入

輸入	說明
1 120 120	此測試案例有 1 個線段 開始端點座標值與結束端點座標值

範例二：輸出

輸出	說明
0	測試案例的結果

評分說明

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制（time limit）均為 2 秒，依正確通過測資筆數給分。每一個端點座標是一個介於 0~M 之間的整數，每筆測試案例線段個數上限為 N。其中：

第一子題組共 30 分，M<1000，N<100，線段沒有重疊。

第二子題組共 40 分，M<1000，N<100，線段可能重疊。

第三子題組共 30 分，M<10000000，N<10000，線段可能重疊。

(a) 解題想法

使用多鍵值排序，線段依照開始端點座標由小到大排序，若開始端點座標相同，則依照結束端點座標由大到小排序。從第一個線段開始，若第一個線段的結束座標包含第二個線段的開始座標，接著若第一個線段的結束座標包含第二個線段的結束座標，則忽略此線段，繼續找尋下一個線段；若第一個線段的結束座標不包含第二個線段的結束座標，則更新第一個線段的結束座標為第二個線段的結束座標，如此下去不斷地串接可以連接的線段，直到結束座標無法包含下一個線段的開始座標，則累加此區段長度到解答，繼續求解下一個線段，最後就可以獲得總共的線段長度。

(b) 程式碼與解說

```
1     #include <cstdio>
2     #include <algorithm>
3     using namespace std;
4     typedef struct _node {
5       int a;
6       int b;
```

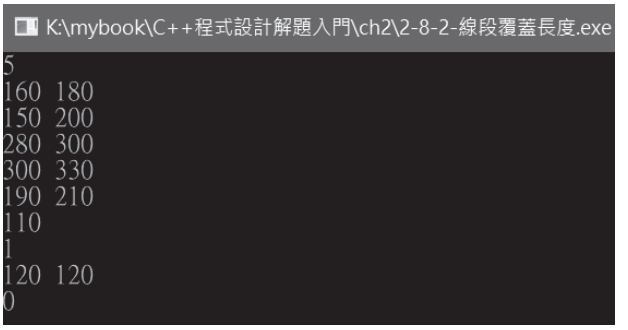
```
7     }Node;
8     bool cmp(Node p, Node q) {
9       if (p.a == q.a) return (p.b > q.b);
10      else return(p.a < q.a);
11    }
12    Node d[10000];
13    int main() {
14      int n, cnt,s,e;
15      while (scanf("%d", &n) != EOF) {
16        cnt = 0;
17        for (int i = 0; i<n; i++) {
18          scanf("%d %d", &(d[i].a), &(d[i].b));
19        }
20        sort(d, d + n, cmp);
21        for (int i = 0; i < n;i++) {
22          s = d[i].a;
23          e = d[i].b;
24          while ((i + 1 < n) && d[i + 1].a < e) {
25            if (d[i + 1].b <= e) i++;
26            else {
27              e = d[i + 1].b;
28              i++;
29            }
30          }
31          cnt = cnt + e - s;
32        }
33        printf("%d\n", cnt);
34      }
35    }
```

- 第 4 到 7 行：結構_node 包含兩個變數 a 與 b，轉換成自訂型別 Node。
- 第 8 到 11 行：自行定義函式 cmp，線段依照開始端點座標由小到大排序，若開始端點座標相同，則依照結束端點座標由大到小排序。
- 第 12 行：宣告陣列 d 有 10000 個元素，每個元素都是資料型別 Node，用於儲存線段的兩個端點座標。
- 第 13 到 35 行：定義函式 main。
- 第 14 行：宣告 n、cnt、s 與 e 為整數變數。
- 第 15 到 34 行：不斷輸入線段個數到變數 n，設定變數 cnt 初始值為 0（第 16 行）。
- 第 17 到 19 行：使用迴圈輸入 n 個線段的兩個端點座標到陣列 d。

- 第 20 行：使用函式 sort 以函式 cmp 為輸入，使用多鍵值排序來排序陣列 d。
- 第 21 到 32 行：使用迴圈變數 i，變數 i 由 0 到 n-1，每次遞增 1，設定變數 s 為 d[i] 的開始座標（第 22 行），設定變數 e 為 d[i] 的結束座標（第 23 行）。
- 第 24 到 30 行：當 i + 1 < n（下一個線段存在）且 d[i + 1].a（下一個線段的開始座標）小於變數 e，若 d[i + 1].b（下一個線段的結束座標）小於等於變數 e，則 i 遞增 1（忽略此線段）；否則設定變數 e 為 d[i + 1].b（下一個線段的結束座標），i 遞增 1（考慮下一個線段）。
- 第 31 行：累加線段的長度到變數 cnt。
- 第 33 行：輸出變數 cnt，變數 cnt 就是所有線段的覆蓋長度。

(c) 預覽結果

按下「執行→編譯並執行」，輸入題目所規定的兩組測資後，螢幕顯示結果如下圖。



(d) 演算法效率分析

本題的第三組測資輸入大量資料，使用函式 scanf 相較指令 cin 有效率，所以本程式融合 C 語言的輸入與輸出與 C++的 algorithm 函式庫。第 20 行排序演算法效率為 O(N*log(N))，N 為線段的個數，第 21 到 32 行找尋線段覆蓋長度演算法效率為 O(N)，本程式演算法效率為 O(N*log(N))。如果使用暴力方式解題，N 個線段，每個線段的座標範圍為 0 到 M，則演算法效率為 O(N*M)，可能無法通過第三組測資。

UVa Online Judge 網路解題資源

排序	
分類	**UVa 題目**
基礎題	UVa 10107 What is the Median?
基礎題	UVa 11462 Age Sort
基礎題	UVa 10474 Where is the Marble?
進階題	UVa 11369 Shopaholic
進階題	UVa 10763 Foreign Exchange
多鍵值排序	
分類	**UVa 題目**
基礎題	UVa 10062 Tell me the frequencies!
基礎題	UVa 10008 What's Cryptanalysis?
進階題	UVa 10194 Football (aka Soccer)

註：使用題目編號與名稱為關鍵字進行搜尋，可以在網路上找到許多相關的線上解題資源。