1081 NTOUCSE 程式設計 1C 期中考

108/11/05 (二)

考試時間：**13:20 – 16:00**

考試規則：1. **請闔上課本**, 除了印給你的之外**不可**參考任何文件包括小考、作業、實習、或是其他參考資料
2. 你可以在題目卷上直接回答, 可以使用鉛筆, 但是**請在答案卷上註明題號**
3. 你覺得有需要的話, 可以使用沒有教過的語法, 但是**僅限於 C/C++ 語言**
4. 程式撰寫時請寫完整的程式碼, 寫 。。。 的分數很低 (一個程式裡有重複很多遍的敘述是扣分的)
5. **不可**使用電腦、平板、智慧手機、及工程型計算機
6. 請**不要**左顧右盼! 請勿討論! 請勿交換任何資料! 對於題目有任何疑問請舉手發問
7. 如果你提早交卷, 請迅速安靜地離開教室, 請勿在走廊喧嘩
8. 違反上述考試規則視為不誠實的行為, 由學校依學務規章處理
9. 請在**題目卷**及**答案卷**上都**寫下姓名及學號**, 交卷時請繳交**題目卷**及**答案卷**

1. [5] 請問C程式中什麼地方可以定義區域變數? 區域變數有什麼特性? 和全域變數有什麼差別?

Sol:

如果是 C90 標準的話，在每一個區塊 (左右大括號圍起來的區域) 的開始，還沒有任何可以執行的敘述之前都可以定義區域變數，區域變數就是在那個區塊裡面使用的變數，程式執行到那個區塊裡面才會新產生的變數，一離開那個區塊，變數就完全消失無法存取，再次進到同一區塊時，變數是全新的，和前一次離開時是沒有關係的，沒有保證裡面的資料會留存，也沒有保證每一次進到區塊時變數會存放在同一塊記憶體。全域變數從那個變數定義的敘述開始一直到檔案的結束之間，任何程式都可以直接使用，全域變數由程式一開始執行時就存在，一直到程式結束，變數才會消失。如果是C99標準的話，在區塊內任何位置都可以定義區域變數，但是區域變數的使用還是限制在區塊內，由定義的地方開始到區塊結束，其他使用方法和 C90 是一致的。

2. [5] 請問具有什麼特性的程式語言是所謂的 strongly typed 語言(weakly typed 語言)? C 語言是那一種? 請問 strongly typed 的語言有什麼好處?

Sol:

Strongly typed 語言是指所有的變數都需要指定變數的型態，每個變數只能存放規劃好存放的那種型態的資料，C 是 strongly typed 程式語言，相對地 weakly typed 程式語言中變數不需要事先指定其存放的資料型態，也可以在每一次使用時都放進不一樣型態的資料，strongly typed 語言雖然覺得彈性比較小，但是因此編譯器可以檢查所有使用變數的敘述，根據變數的型態來判斷是否程式設計者有以無法預期的方式使用變數，適時提出警告訊息，在撰寫大型程式時，可以避開很多問題。

3. [5] C 語言裡面資料的儲存格式主要分為整數與浮點數兩種，其中整數是二的補數格式，又根據存放資料大小的範圍分為哪幾種資料型態? 浮點數是 IEEE754 格式，分為哪幾種資料型態?

Sol:

char 為一個位元組的整數，可以存放的資料範圍在-128~127 之間

short 為兩個位元組的整數，可以存放的資料範圍在-32768~32767 之間

int 或是 long 為四個位元組的整數，可以存放的資料範圍在-2^{31}~2^{31}-1 之間

long long 為八個位元組的整數，可以存放的資料範圍在-2^{63}~2^{63}-1 之間

浮點數也依據存放資料大小的範圍分為三種：

float 為四個位元組的浮點數，可以存放的資料範圍大約在-10^{38}~10^{38}之間，絕對值最小的數值大約為$\pm10^{-38}$，有效位數大約為十進位 7 位數

double 為八個位元組的浮點數，可以存放的資料範圍大約在-10^{308}~10^{308}之間，絕對值最小的數值大約為$\pm10^{-308}$，有效位數大約為十進位 14 位數

long double 有些編譯器對應到 IEEE 754 80 位元的 extended precision floating，有些編譯器直接對應到 double，有些編譯器則對應到 128 位元的 IEEE 754 quadruple-presicion 浮點數格式。

4. [5] 請問下列 C 程式片段列印出來的資料為何?

```
01 int i, a[101];
02 a[0] = 0;
03 for (i=1; i<=100; i++)
04     a[i] = 2*i + a[i-1];
05 printf("%d\n", a[87]+a[31]);
```

Sol:

8648

$a_0 = 0, a_1=2, a_2=2(1+2)=6, a_3=2(1+2+3)=12, \ldots, a_n=n(n+1)$

$a_{87}+a_{31}=87*88+31*32=8648$

5. [5] 請完成下面計算陣列裡最大值和最小值的函式

```
01  void findMinMax(int n, int data[], _______min, _____max ) {
02      int i;
03      _______ = _______ = data[0];
04      for (i=1; i<n; i++)
05          if (data[i]<_______)
06              _____ = data[i];
07          else if (data[i]>_______)
08              _____ = data[i];
09  }
```

Sol:

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

6. [5] 請以 15 列以內的 switch 敘述改寫下列函式

```
01 char convertGrade(int score) {
02     if ((score >= 90)&&(score <= 100))
03         return 'A';
04     else if (score >= 80)
05         return 'B';
06     else if (score >= 70)
```

```
07              return 'C';
08      else
09              return 'F';
10  }
```

Sol:

```
char gradeConvert(int score)
{
    switch (score/10)
    {
    case 10:
    case 9:
        return 'A';
    case 8:
        return 'B';
    case 7:
        return 'C';
    default:
        return 'F';
    }
}
```

7. [10] 請撰寫一個程式讀取下圖左輸入串流中的測資，第一列是測資數量，每一筆測資有一列，其中有兩串 hh:mm:ss 的資料代表飛機起飛和降落的時間 t_1 與 t_2 (兩個 24 小時制)，請以 hh:mm:ss 的格式在螢幕上印出飛機的飛行時間 (請注意飛行時間小於 24 小時並特別注意輸出的格式)。

範例輸入
3 12:01:25 21:15:49 17:55:18 09:31:40 12:34:56 23:59:59

範例輸出
09:14:24 15:36:22 11:25:03

Sol:

```
#include <stdio.h>
int main()
{
    int n, time1, time2, diff, hh, mm, ss;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%d:%d:%d", &hh, &mm, &ss);
        time1 = hh*60*60+mm*60+ss;
        scanf("%d:%d:%d", &hh, &mm, &ss);
        time2 = hh*60*60+mm*60+ss;
        diff = time2 > time1 ? time2-time1 : 24*60*60-time1+time2;
        printf("%02d:%02d:%02d\n", diff/3600, diff/60%60, diff%60);
    }
    return 0;
```

```
}
====================
#include <stdio.h>
int main()
{
    int i, n, time[2], diff, hh, mm, ss;
    scanf("%d", &n);
    while (n--)
    {
        for (i=0; i<2; i++)
        {
            scanf("%d:%d:%d", &hh, &mm, &ss);
            time[i] = hh*60*60+mm*60+ss;
        }
        diff = time[1] - time[0];
        if (diff<0) diff += 24*60*60;
        printf("%02d:%02d:%02d\n", diff/3600, diff/60%60, diff%60);
    }
    return 0;
}
```

8. [10] 請撰寫一個程式，讀取下圖左輸入串流中的多筆輸入資料，串流結束代表所有資料結束，其中每一筆測資包括兩列資料，第一列是數字的個數 n，第二列包含 n 個正整數 a_1, a_2, …, a_n，請判斷在最多修改一個數字的情況下，是否可以得到一個單調遞增序列 b_1, b_2, …, b_n (亦即 b_i 序列滿足 $b_1 \le b_2 \le \ldots \le b_n$，且 n 對 (a_i, b_i) 中最多只能有一對不一樣)，是的話請輸出 True，反之輸出 False

範例輸入
3 4 2 3 3 4 2 1 6 3 7 10 4 8 9 7 5 7 9 8 9 10 10

範例輸出
True False False True

Sol:

基本想法是如果只有一對數字反序，例如 4 6 5，可以修改 6 成為 5，變成 4 5 5 就是一個單調遞增的序列了；仔細分析一下，只要一出現「兩對以上的數字反序」就表示無法只修改一個數字使得數列單調遞增了，這包括連續兩對反序，例如 7 6 5，不論把 6 改小消除後面的反序對，或是把 6 改大消除前面的反序對，都還是會留下一對反序對，把 7 改小或是把 5 改大也是會留下一對反序對；另外一種狀況是不連續的兩對數字反序，例如 5 7 6 8 8 10 9 11，其中 7 6 和 10 9 都反序，不管改那一個，整個序列都還不是單調遞增序列；另外當只有一對反序時，例如 4 6 5 8，一種是把 6 改小來消除，另一種是把 5 改大來消除，前者需要把 a 6 5 拿出來判斷，如果 a<=5，就可以把中間的 6 改小，如果 a>5 就沒有辦法了，後者則是把 6 5 b 拿出來判斷，如果 6<=b，就可以把中間的 5 改大，如果 6>b 就沒有辦法了，這兩種如果都不符合，就表示無法只修改一個數字使得數列單調遞增了。

```
#include <stdio.h>
```

```
int main()
{
	int i, n, x[3], ix, count;
	while (1==scanf("%d", &n))
	{
		for (count=x[0]=x[1]=x[2]=ix=i=0; i<n; i++,ix=(ix+1)%3)
		{
			scanf("%d", &x[ix]);
			if ((x[ix]<x[(ix+2)%3])&&((++count>1)||(x[ix]<x[(ix+1)%3])))
				i=n, scanf("%*[^\n]");
		}
		printf(i==n+1 ? "False\n" : "True\n");
	}
	return 0;
}
```

程式中的 count 就是紀錄先前有沒有出現過反序對，(ix+2)%3, ix 相當於 i-1, i，(ix+1)%3, ix 相當於 i-2, i，程式中檢查相鄰數字 x[ix]<x[(ix+2)%3] 看是不是反序，反序時檢查 count 看看先前有沒有出現過反序對，沒有的話再檢查更前一個數字 x[ix]<x[(ix+1)%3]看是不是反序；另外迴圈裡面用 i=n 來強制結束迴圈，如此提早離開迴圈時變數 i 應該是 n+1，正常離開時變數 i 應該是 n

==========

如果不用變數 ix，只用變數 i 來控制的話，程式寫起來比較清楚，雖然運算稍微多一點點，但是應該是划得來的選擇

```
#include <stdio.h>
int main()
{
	int i, n, x[3], count;
	while (1==scanf("%d", &n))
	{
		for (count=x[0]=x[1]=x[2]=i=0; i<n; i++)
		{
			scanf("%d", &x[i%3]);
			if ((x[i%3]<x[(i+2)%3])&&((++count>1)||(x[i%3]<x[(i+1)%3])))
				i=n, scanf("%*[^\n]");
		}
		printf(i==n+1 ? "False\n" : "True\n");
	}
	return 0;
}
```

==========

題目沒有指定數字 n 的範圍，上面這個作法的陣列只有三個元素，因為任何時候只需要知道前一個數字還有更前一個數字就夠了，所以不需要擔心 n 有多大，只是需要用%3 的方法來存取，如果用陣列把所有資料讀進來處理的話，就要知道數字 n 的範圍，以便規劃陣列的大小，但是可以直接用 x[i]<x[i-1]還有 x[i]<x[i-2]來檢查，邏輯上是最簡單的，例如

```
#include <stdio.h>
int main()
{
```

```
    int i, n, x[10002]={0}, count;
    while (1==scanf("%d", &n))
    {
        for (i=0; i<n; i++) scanf("%d", &x[i+2]);
        for (count=i=0; i<n; i++)
        {
            if ((x[i+2]<x[i+1])&&((++count>1)||(x[i+2]<x[i])))
                i=n, scanf("%*[^\n]");
        }
        printf(i==n+1 ? "False\n" : "True\n");
    }
    return 0;
}
```

由於需要檢查 x[0]<x[-1]還有 x[0]<x[-2]，在使用陣列的時候是不允許的，所以我們多宣告兩個元素，所有存取時的陣列註標都加 2 就可以省掉額外的條件判斷

==========

如果不喜歡所有的註標加 2，也可以運用類似平移的方法來得到 x[-2]..x[10000]的陣列，例如

```
#include <stdio.h>
int main()
{
    int i, n, x0[10002]={0}, *x=&x0[2], count;
    while (1==scanf("%d", &n))
    {
        for (i=0; i<n; i++) scanf("%d", &x[i]);
        for (count=i=0; i<n; i++)
        {
            if ((x[i]<x[i-1])&&((++count>1)||(x[i]<x[i-2])))
                i=n, scanf("%*[^\n]");
        }
        printf(i==n+1 ? "False\n" : "True\n");
    }
    return 0;
}
```

9. [10] 請撰寫一個程式，讀取下圖左輸入串流中的測資，第一列是測資數量，每一筆測資有兩個十進位正整數 a 和 b，請計算並輸出十進位 a + b 的加法運算中有幾次的進位? 例如第二筆測資 356 + 645，個位數 6+5=11 有進位，十位數 5+4+1=10 有進位，百位數 3+6+1=10 有進位，所以共有 3 次進位

範例輸入	範例輸出
3 123 456 356 645 92365 220839	0 3 4

Sol:

這個題目沒有說十進位正整數會有多少位數，但是這個題目一定需要這個資料，因為需要由個位數自開始處理，必須把整個數字都完了才會看到個位數，而數字讀進來以後需要存放在

變數裡，這裡可以用字元陣列來存放，所以需要位數的資訊，沒有給我們只好假設一個數字，例如 1000 好了，下面的程式運用 scanf("%1000s%1000s",a,b) 讀取這兩個數字陣列，使用%1000s 比使用%s 要好，萬一超過這個數字的時候，陣列保留的空間不夠，程式也不會發生未定的狀況，接向來用兩個 while 迴圈計算出兩個數字的位數 alen 及 blen，接下去由各位數字 a[alen-1]和 b[blen-1]開始一位數一位數相加，carry 為 0 或是 1，紀錄的是前一位數的進位值，carryCount 則紀錄總共有幾位數發生進位

```
#include <stdio.h>
int main()
{
	int n, alen, blen, carry, carryCount;
	char a[1001], b[1001];
	scanf("%d", &n);
	while (n--)
	{
		scanf("%1000s%1000s ", a, b);
		alen=0; while (a[alen]!=0) alen++;
		blen=0; while (b[blen]!=0) blen++;
		carryCount = carry = 0;
		while ((alen>0&&blen>0)||(alen>0&&carry>0)||(blen>0&&carry>0))
		{
			if (a[--alen]-'0' + b[--blen]-'0' + carry > 9)
			{
				carry = 1;
				carryCount++;
			}
			else
				carry = 0;
		}
		printf("%d\n", carryCount);
	}
	return 0;
}
```

==========

也可以用 scanf("%1000s%n", a, &alen) 直接取得數字字串 a 有幾位數 alen，但是要小心用另外的 scanf("□") 敘述把字串之前的空格先跳掉，如此 alen 才不會包括空格的個數，我們把這個跳過空格的命令直接放在前一個 scanf("%d□!", &n) 命令裡面，格式命令多了一個 ！字元其實一定會使得格式轉換命令出錯，但是也確定前面一個跳過空格的命令有執行到，基本上只有空格應該也可以運作，但是有函式庫會跳過尾巴的空格命令，所以才加上這個 ！，另外也用 carry = a[--alen]-'0' + b[--blen]-'0' + carry) > 9 的敘述來取代 if 的敘述

```
#include <stdio.h>
int main()
{
	int n, alen, blen, carry, carryCount;
	char a[101], b[101];
	scanf("%d□!", &n);
	while (n--)
	{
```

```
            scanf("%100[0-9]%n□!", a, &alen);
            scanf("%100[0-9]%n□!", b, &blen);
            carryCount = carry = 0;
            while ((alen>0&&blen>0)||(alen>0&&carry>0)||(blen>0&&carry>0))
                carryCount += (carry = (a[--alen]-'0' + b[--blen]-'0' + carry) > 9);
            printf("%d\n", carryCount);
        }
        return 0;
    }
```

10. [10] 請撰寫一個程式，讀取下圖左輸入串流中的多筆輸入資料，串流結束代表所有資料結束，其中每一筆測資包括兩列資料，第一列是學生人數 n，第二列包含由左至右排程一列的 n 個學生的身高資料 $a_1, a_2, \ldots, a_n$，a_i 為整數且以逗點分隔，$1 \le n \le 100$，如果希望同學們按照身高依序排好，左側最矮、右側最高，請計算出有多少位同學需要更換位置?

範例輸入	範例輸出
6 168, 168, 184, 172, 168, 173 11 168, 164, 162, 163, 163, 164, 161, 173, 170, 164, 172	3 8

Sol:

```
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b) { return *(int *)a - *(int *)b; }
int main()
{
    int heights[100], sorted[100], len, i, count;
    while (1==scanf("%d", &len))
    {
        for (i=0; i<len; i++)
            scanf("%d,", &heights[i]), sorted[i]=heights[i];
        qsort(sorted, len, sizeof(int), compare);
        for (count=i=0; i<len; i++)
            if (sorted[i]!=heights[i]) count++;
        printf("%d\n", count);
    }
    return 0;
}
```

==========

如果不使用 stdlib.h 裡面的 qsort()工具的話，也可以自己寫一個 bubbleSort 來取代，以這個題目最多 100 個元素來說，qsort()沒有比 bubbleSort 快很多

```
void swap(int *a, int *b) { int tmp=*a; *a=*b; *b=tmp; }
void bubbleSort(int len, int data[])
```

```
{
    int i, j;
    for (i=0; i<n-1; i++)
        for (j=n-2; j>=i; j--)
            if (data[j]>data[j+1]) swap(&data[j], &data[j+1]);
}
```

==========

另外計算不等個數的迴圈也可以寫成

```
        for (count=i=0; i<len; i++)
            count += (sorted[i]!=heights[i]);
```

11. [10] 請撰寫一個程式，讀取下圖左輸入串流中的多筆輸入資料，串流結束代表所有資料結束，其中每一筆測資包括兩列資料，第一列是物品的個數 n，第二列包含由左至右排成一列的 n 個物品的重量 $a_1, a_2, \ldots, a_n$，$1 \le n \le 1000000$，$1 \le a_i < 2^{31}$，物流人員處理時由最輕的物品開始一件一件依照重量來處理，為了方便起見每次由最左到最右才折返或是由最右到最左再折返，請問由最左邊開始向右處理，需要來回處理幾次才可以把所有物品處理完畢？例如下面第一筆測資，第一趟由左到右處理 76,80，第二趟由右到左處理 85, 210，第三趟由左到右處理 346。

範例輸入	範例輸出
5 76 210 85 346 80 2 100 152 1 7	3 1 1

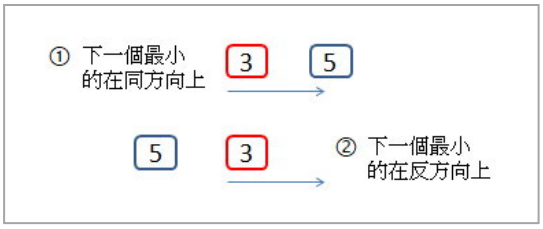
Sol:

(A) 如果題目有規定 a_i 都不相等，作法的概念很單純：

在任何時間點如果知道下一個最輕的物品在哪裡，是在移動的方向上(下圖中假設 3 處理完畢之後最小的是 5，而且處理 3 時是往右移動，往左的情況類似)，還是在反方向上，就知道需不需要多一趟折返來處理那個物品了，例如以第一筆測資為例，一開始由左向右，最輕的物品重量是 76，可以直接處理，移除此物品後，下一個最輕的物品重量是 80，還是在同一方向上，可以在這一趟就處理完畢，移除它以後，下一個最輕的物品重量是 85，但是在反方向上了，所以必須在折返以後下一趟時處理。用一個變數 count 來紀錄是第幾趟，所以可以用 count%2 這個數字來代表處理時移動的方向，count%2==0 代表由左向右，count%2==1 代表由右向左。是不是在移動的方向上可以看目前物品和上一個物品的相對位置來決定：count%2==0 時，如果目前物品在上一個物品的右邊，也就是檢查 x[i][1]>x[i-1][1]，就在同一方向上；count%2==1 時，如果目前物品在上一個物品的左邊，也就是檢查 x[i][1]<x[i-1][1]，就在同一方向上；下面的程式簡化成 count += count%2==0 ? x[i][1]<x[i-1][1] : x[i][1]>x[i-1][1]，也就是當 count%2==0 時反方向才加 1 同方向則加 0：count += x[i][1] < x[i-1][1];，最後因為我們需要由輕而重一個一個處理物品，所以如果讀入資料以後，先按照順序排好，就可以很快地由輕而重地處理所有的物品了

```
#include <stdio.h>
#include <stdlib.h>
int x[1000000][2];
int comp(const void *a, const void *b) { return *(int*)a-*(int*)b; }
```

```
int main()
{
    int i, n, count;
    while (1==scanf("%d", &n))
    {
        for (i=0; i<n; i++)
            scanf("%d", &x[i][0]), x[i][1]=i;
        qsort(x, n, sizeof(int[2]), comp);
        for (count=0,i=1; i<n; i++)
            count += count%2==0 ? x[i][1]<x[i-1][1] : x[i][1]>x[i-1][1];
        printf("%d\n", count+1);
    }
    return 0;
}
```



如果不使用 stdlib.h 裡面的 qsort()工具，自己寫一個 bubbleSort 來取代的話，可以寫出正確運作的程式，但是因為物品的個數可以到達 1000000，bubbleSort 的運算時間正比於 $O(n^2)$，需要 10^{12} 左右的動作，在現在的機器每秒鐘執行 10^9 個運算的話，大約要 1000 秒 (15 分鐘左右)，qsort 的運算時間正比於 O(n log n)，需要 10^7 左右的動作，還在 1 秒以內，前面的程式碼在計算總共折返幾次時只用下面這個迴圈把陣列看過一次 (n 個元素)

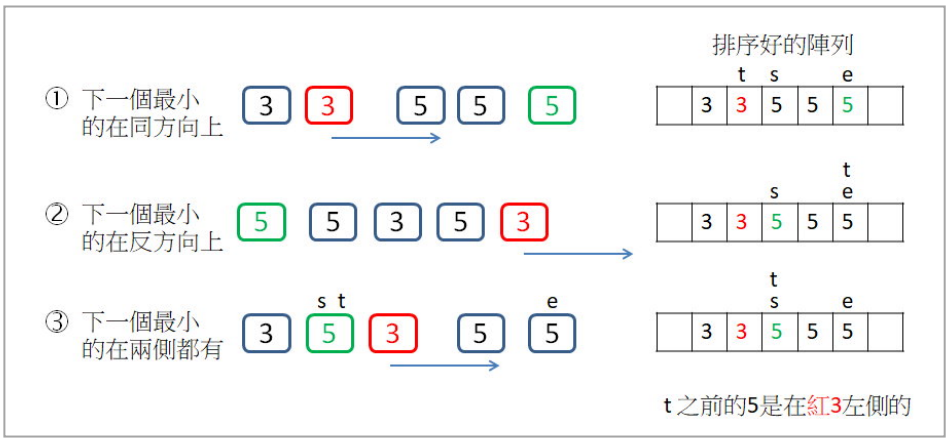
```
        for (count=0,i=1; i<n; i++)
            count += count%2==0 ? x[i][1]<x[i-1][1] : x[i][1]>x[i-1][1];
```

如果用下面這種完全模擬的方式，不需要有 x[i][1] 這個額外的欄位

```
int x[1000000], y[1000000];
int comp(const void *a, const void *b) { return *(int*)a-*(int*)b; }
int main()
{
    int i, j, n, count;
    while (1==scanf("%d", &n))
    {
        for (i=0; i<n; i++)
            scanf("%d", &x[i]), y[i]=x[i];
        qsort(y, n, sizeof(int), comp);
        for (count=i=0; i<n; i++)
        {
            for (count++,j=0; j<n; j++)   // 向右走
                if (x[j]==y[i]) i++, x[j]=-1;
            if (i>=n) break;
            for (count++,j=n-1; j>=0; j--)   // 向左走
                if (x[j]==y[i]) i++, x[j]=-1;
        }
        printf("%d\n", count+1);
    }
    return 0;
}
```

這個寫法使用的記憶體一樣多，很簡潔、邏輯清楚、而且還可以處理 a_i 相等的情況，不過這樣子做的時候如果運氣不好，遇見每一趟都只能處理一個物品的測資，兩層迴圈最多會做的動作次數正比於 n^2，物品的個數如果到達 10^6，需要 10^{12} 左右的動作。

(B) 題目沒有說清楚，所以如果允許 a_i 可以相等，需要多考慮一點：



首先 a_i 相同時物品還是有左右順序的，並不允許排序的時候隨意對調這些物品的順序，所以運用 qsort() 的時候，必須要避免在兩個物品有相同重量時回傳 0，需要維持原本的左右順序，這樣子才知道相同重量時每一個相對於前一個重量的最後一個的位置，所以在 comp() 函式中先後以兩個欄位的大小來排序，是否折返則需要考慮下圖中三種狀況，下途中假設紅色的 3 是目前處理的重量中最後一個物品，先考慮向右移動的狀況，向左移動的狀況類似，狀況①是所有下一個最小重量的物品 (假設是 5) 都在同一個方向上，此時不需要改變方向就可以處理完所有的 5，最後一個處理到的是同方向最靠邊綠色的 5；狀況②是所有下一個最小重量的物品都在反方向上，此時必須改變方向，最後一個處理到的是反方向最靠邊綠色的 5；狀況③是下一個最小重量的物品分在兩側，此時不需要改變方向可以處理同方向的 5，然後必須改變方向處理反方向的 5，最後一個處理到的是反方向最靠邊綠色的 5；程式裡面用 s 代表下一個最小重量的物品由左到右排序時最左邊的在排序好的陣列裡的位置，用 e 代表最右邊的下一個最小重量的物品在排序好的陣列裡的位置，t 則是代表在剛處理完的最後一個物品左邊下一個最小重量的物品在排序好的陣列裡的位置，程式裡在計算有幾趟的迴圈 for (count=0,i=1,x[n][0]=-1; i<n; i++)時首先檢查 x[i][0]>x[i-1][0]，在 x[i][0]==x[i-1][0]時不會有折返，所以不需要做任何事情，接下先配合 x[n][0]=-1 的初始值找出 s,t,e 三個位置，然後檢查 6 種不同的狀況，在折返的地方讓 count 數值加 1，由於檢查 x[i-1][0]代表前一個重量的最後一個物品的所在位置，所以當綠色的在最左邊時，用 x[e][1] = x[s][1]來修改最後一個處理的物品的位置。i = e; 那一列是當相同重量的物品很多時，用來直接跳過相同重量物品的，沒有這一列也會有相同的結果。這個寫法邏輯複雜很多，寫起來很容易有錯誤，目標只是為了讓計算的時間維持在 O(n log n)。

```
#include <stdio.h>
#include <stdlib.h>
int comp(const void *a, const void *b)
{
      int *pa = (int *)a;
      int *pb = (int *)b;
      if (pa[0]==pb[0])
            return pa[1]-pb[1];
      return pa[0]-pb[0];
}
int main()
{
      int i, s, t, e, n, count;
      int (*x)[2] = (int (*)[2]) malloc(sizeof(int[2])*1000000);

      while (1==scanf("%d", &n))
```

```
int comp(const void *a, const void *b)
{
      if (((int*)a)[0]==((int*)b)[0])
            return ((int*)a)[1]-((int*)b)[1];
      return ((int*)a)[0]-((int*)b)[0];
}
```

```
    {
        for (i=0; i<n; i++)
            scanf("%d", &x[i][0]), x[i][1]=i;
        qsort(x, n, sizeof(int[2]), comp);
        for (count=0,i=1,x[n][0]=-1; i<n; i++)
            if (x[i][0]>x[i-1][0])
            {
                for (t=i-1,s=e=i; x[e][0]==x[s][0]; e++)
                    if (x[e][1]<x[i-1][1]) t=e;
                e--;
                if ((count%2==0)&&(t<s)) ;
                else if (count%2==0) // && ((t==e)||((s<e)&&(t<e)))
                    count++, x[e][1] = x[s][1];
                else if ((count%2==1)&&(t==e))
                    x[e][1] = x[s][1];
                else // if ((count%2==1)&&((t<s)||((s<e)&&(t<e)))
                    count++;
                i = e;
            }
        printf("%d\n", count+1);
    }
    return 0;
}
```

12. 對於兩個字串 S 和 T，如果字串 S 由字串 T 串接而成(亦即 S = T + T + … + T)則我們定義為「字串 T 整除字串 S」，例如「字串"AB"整除字串"ABABAB"」，請寫一個程式讀取下圖左輸入串流中的多筆輸入資料，串流結束代表所有資料結束，其中每一筆測資包括兩列資料，第一列是字串 1，第二列是字串 2，每一字串包括最多 1000 個大寫英文字母，請找到並且輸出最長的字串整除字串 1 以及字串 2，如果不存在長度大於等於 1 的字串時，請輸出*，請完成下列要求：

範例輸入
ABCABC
ABC
ABABAB
ABAB
TEST
CODE
EEEEEEE
E

範例輸出
ABC
AB
*
E

a) [5] 請撰寫函式 int gcd(int a, int b); 計算兩個整數的最大公因數，其中 a≥b，並且回傳其值

Sol:

```
int gcd(int a, int b)
{
    if (a%b==0)
        return b;
    else
        return gcd(b, a%b);
```

```
}
```

b) [5] 請撰寫函式 int divides(int len1, char str1[], int len2, int str2[]); 測試 str1 陣列內由 str1[0] 到 str1[len1-1]這個長度 len1 的部份字串是不是整除 str2 陣列內長度 len2 的字串，請注意 str1 陣列中 str1[len1]不見得是 0，也就是說陣列 str1 裡面的字串的長度可能會大於 len1，但是這個函式只需要比對長度為 len1 的部份就夠了，len2 則是 str2 陣列裡面完整字串的長度，如果整除的話回傳 1 否則回傳 0

Sol:

```
int divides(int len1, char *str1, int len2, char *str2)
{
    int i, j;
    for (i=0; i<len1; i++)
        for (j=i; j<len2; j+=len1)
            if (str2[j]!=str1[i]) return 0;
    return 1;
}
```

c) [10] 請完成下列函式 char *gcdStrings(char *str1, char *str2); 計算整除兩個字串的最長字串，並且存放在 str2 陣列中回傳，如果不存在長度大於等於 1 的字串整除此兩個陣列中的字串，則把 '*' 放進 str2 陣列中做成一個字串並且回傳 str2

Sol:

```
char *gcdStrings(char *str1, char *str2)
{
    int len1=0, len2=0, tmpI, g, f;
    char *tmpS;
    while (str1[len1]!=0) len1++;
    while (str2[len2]!=0) len2++;
    if (len2>len1) {
        tmpS=str1; str1=str2; str2=tmpS;
        tmpI=len1; len1=len2; len2=tmpI;
    }
    if (len2>0)
    {
        g = gcd(len1, len2);
        for (f=1; f*f<=g; f++) // f 是所有小於等於根號 g 的因數
            if ((g%f==0)&&divides(g/f, str2, len2, str2)&&divides(g/f, str2, len1, str1))
            {
                str2[g/f] = 0;
                return str2;
            }
        f--; // 上述迴圈執行完畢時，f 一定大於根號 g
        if (f*f==g) f--;
        for (; f>=1; f--)
            if ((g%f==0)&&divides(f, str2, len2, str2)&&divides(f, str2, len1, str1))
            {
                str2[f] = 0;
```

```
                return str2;
            }
        }
        str2[0] = '*', str2[1] = 0;
        return str2;
}
```

```
int main() {
    char str1[1001], str2[1001];
    while (2==scanf("%1000s%1000s", str1, str2))
        printf("%s\n", gcdStrings(str1, str2));
    return 0;
}
```

上面程式裡 **divides**(g/f, str2, len2, str2) 可以換成 **divides**(g/f, str2, len2-g/f, &str2[g/f])，因為自己比對時不需要重複前面 g/f 個字元，同樣地 **divides**(f, str2, len2, str2) 可以換成 **divides**(f, str2, len2-f, str2+f)，str2+f 和&str2[f]是完全一樣的敘述