APCS 複習

2018/5/12 by Sean

一、時間複雜度

O(x)，x 越大跑越久，O 函數代表執行步驟數目上限
如果一個演算法執行的步驟是固定的，無關輸入的值而改變，那我們會記成 **O(1)**

而下面這個演算法則是依據輸入的 n 的數量會跑 n 次，所以是 O(n)：

```
1. function(int n){
2.     for(i=0;i<n;i++){
3.         print(i);
4.     }
5. }
```

這個演算法雖然跑了 n*(n-1) = n2-n 次，但我們還是會記做 O(n2)，也就是說，**只要找出最高次方，並且把係數拿掉即可**。

```
1. function(int n){
2.     for(i=0;i<n;i++){
3.         for(j=0;j<n-1;j++){
4.             print(i*j);
5.         }
6.     }
7. }
```

常見的時間複雜度還有：O(nlog(n))、$O(n^2)$、$O(2^n)$、$O(n^3)$... 等等，不用特別去記，只要大概的數一下迴圈數量，大致上**判斷一下丟進去的變數會讓程式執行幾次**即可。

二、排序

1. Insertion sort

分成未排序及已排序，將未排序中正在處理的值放到已排序中的適當位子

https://www.youtube.com/watch?v=DFG-XuyPYUQ

時間複雜度 (Time Complexity)

Best Case(omega)：O(1)

當資料的順序恰好為由小到大時，每回合只需比較 1 次

Worst Case：$O(n^2)$

當資料的順序恰好為由大到小時，第 i 回合需比 i 次

Average Case：$O(n^2)$

第 n 筆資料，平均比較 n/2 次

2.Selection sort

分成未排序及已排序，從未排序中找出最小的數，放入已排序的最尾端

https://www.youtube.com/watch?v=f8hXR_Hvybo

時間複雜度 (Time Complexity)

Best Case：$O(n^2)$

Worst Case：$O(n^2)$

Average Case：$O(n^2)$

無論資料順序如何，都會執行兩個迴圈

3.Bubble sort

https://www.youtube.com/watch?v=nmhjrI-aW5o

時間複雜度 (Time Complexity)

Best Case：O(n)

當資料的順序恰好為由小到大時

第一次執行後，未進行任何 **swap** ⇒ 提前結束

Worst Case：$O(n^2)$

當資料的順序恰好為由大到小時

每回合分別執行：n-1、n-2、...、1 次

(n-1) + (n-2) + ... + 1 = n(n-1)/2 ⇒ $O(n^2)$

Average Case：$O(n^2)$

第 n 筆資料，平均比較 (n-1)/2 次

4.Counting sort

5. Code of insertion/selection/bubble sort

https://gist.github.com/cilegann/2a5e66a213e21acdec282fc920e14450

6. STL lib

> #include<algorithm>

> sort(array_begin, array_end)

> sort(a,a+5)

> 取得變數佔用的記憶體量，把取得的值 / 變數型別就可以得到陣列長度了

> int end = sizeof(a)/sizeof(int)

> sort(a,a+end)

> 降冪 ：reverse(array_begin,array_end)

7.參考題目

a104 a233 d190

三、Struct

> 為何要用結構?

B899 中 我們把每個點的 X Y 都分開定義，但是他們其實是一個東西的很多屬性。如果我們定義一個叫做「點」的資料型態，不是很方便?

> 像是一台車，有 廠牌、顏色、馬力、長度....等等屬性，若我們要定義一台車，必須要逐行宣告這些散亂在四處的變數，會很難管理。這時我們可以定義一個叫做 car 的資料型態，也就是 struct。

> 基本結構

```
struct car{
    string brand;
    string color;
    double power;
    double length;
};
```

> 直接在 main 存取

```
#include<iostream>
using namespace std;
struct car{
    string brand;
    string color;
    double power;
    double length;
};
int main(){
    car mycar;
    mycar.brand="BMW";
    mycar.color="Black";
```

```
    mycar.power=120;
    mycar.length=250;
    car cars[10];
    cars[0].brand="TOYOTA";
    cars[0].color="Black";
    cars[0].power=100;
    cars[0].length=250;
    cout<<cars[0].name;
}
```

> constructer

但是這樣還不夠方便，我們可以定義一個屬於這個資料型態的 function，拿來初始化這個結構。這個 function 叫做 constructor 建構子。

```
#include<iostream>
using namespace std;
struct vector{
    double x;
    double y;
    vector(){x=0;y=0;}
    vector(double x,double y){this->x=x;this->y=y;}
};

int main(){
    vector vec=vector(10,2);
    cout<<vec.x<<" "<<vec.y;
}
```

> 外部 function (struct 1,struct 2)

既然他是一個型態，當然也可以拿來當作 function 的參數及回傳型態

```
#include<iostream>
using namespace std;
struct vector{
    double x;
    double y;
    vector(){x=0;y=0;}
    vector(double x,double y){this->x=x;this->y=y;}
};

void printVector(vector vec){
    cout<<"X="<<vec.x<<" ,Y="<<vec.y<<endl;
}

vector add(vector vec1,vector vec2){
    vector ans=vector(vec1.x+vec2.x,vec1.y+vec2.y);
    return ans;
}

int main(){
    vector vec1=vector(10,2);
    vector vec2=vector(7,3);
    vec1=add(vec1,vec2);
    printVector(vec1);
}
```

> 內部 function (struct 2)

1.定義在裡面，實作在外面 void vector::add

2.this->x todo.x

```
#include<iostream>
using namespace std;
struct vector{
    double x;
    double y;
    vector(){x=0;y=0;}
    vector(double x,double y){this->x=x;this->y=y;}
    void printVector();
    void add(vector todo);
};

void vector::printVector(){
    cout<<"X="<<this->x<<" ,Y="<<this->y<<endl;
}

void vector::add(vector todo){
    this->x=this->x+todo.x;
    this->y=this->y+todo.y;
}

int main(){
    vector vec1=vector(10,2);
    vector vec2=vector(7,3);
    vec1.add(vec2);
    vec1.printVector();
}
```

四、補充 function

1. 輸入數字的另一種方式

```
int a;
while(scanf(“%d”,&a)!=EOF)
```

Scanf 這個 function 會在 a 成功讀入後 return 1。若遇到錯誤或是輸入結束(EOF)，則會 return EOF。

2. 取出字串 str 中的某段

```
string str=”ABCDE”;
str.substr(開始取的位置,長度);
```