# DSA作業二手寫

**B00104009 生傳四 李奕儒**

- (2.1)

  1. Because variable c in sub1 fuction is stored in memory location of **stack**, when sub1 fuction is over , momory will be deallocated. But, at the end of sub1 function still return the reference of the variable c, the caller fuction can’ t access the memory.
  2. In sub1 fuction, pointer pc, which is stored in the memory location of **stack**, points to the momory location of **heap**. When it comes to the end of sub1 function, it returns the value of memory. The caller function do receive the value, but the caller fuction can’ t access memory location of heap anymore, because the pointer pc has been deallocated. There will be **memory leak** problem.

- (2.2)

  1. Use bubble sort first, make array sorted in increasing order.
  pseudo code:

  ```
  counting=1, number=0
  for i=0 to i=size of arrays
   if array[i]==array[i+1]
    counting++
   if counting==5
    return number=array[i]
  ```

  2. Only need to store a[0][0], a[1][0], a[1][1], a[2][0], a[2][1], a[2][2]... in one-dimentional array. The pseudo code presented as below:

```
//datalist為Circular linked list
//datalist的back()會回傳目前cursor的element
//datalist的next()會讓cursor指向下一個node
//轉換方式
ary[ i(i+1)/2 + j ] = A[i][j];
//get從metrix拿element放到array裡
int get(int a[][], int b[] , int i, int j)
{
 b[ i(i+1)/2 + j ] = a[i][j];
 return;
}
//put將array[i] assign給對應的metrix's element
//void put(int a[][], int b[], int i)
{
 int counting =1;
 int sum=0;
 for(sum>=i+1)
 {
  sum += counting++;
 }
 a[counting-2][i-sum+counting-1] = b[i];
 return;
}
```

3. First, counting both numbers of nodes in L and M, if numbers of nodes are not same, L and M are not same. Second, If the numbers of nodes are same, then make sure the content of nodes in L is totally as orderly same as M. The code presented as below:

```
bool cmp(datalist* L_Head, datalist* M_Head)
{
int L_counting=0;
int M_counting=0;
datalist* L_Iterator = L_Head;
datalist* M_Iterator = M_Head;
while(L_Iterator->next() != L_Head)
L_counting++;
while(L_Iterator->next() != M_Head)
M_counting++;
if(L_counting != M_counting)
 return false;
else
{
 do
 {
  if(L_Iterator->back() !=M_Head->back())
   continue;
  else
  {
   datalist* L_tmp = L_Iterator;
   datalist* M_tmp= M_Iterator;
   while(L_tmp->next()!= L_Iterator)
   {
   if (L_tmp->back() != M_tmp->next().back())  break;
   else continue;
   }
  }
 }while(L_Iterator->next() != L_Head);
}
```

4. code presented below:

```
//evenpos defualt 輸入為0
//oddpos default 輸入最後一個元素的位置
void sort(int ary[], int evenpos, int oddpos)
{
 while(ary[evenpos]%2==0)
  evenpos++;
 while(ary[oddpos]%2==1)
  oddpos--;
 if(evenodd >= oddpos)
  return;
 else
 {
  int tmp = ary[evenpos];
  ary[evenpos] = ary[oddpos];
  ary[oddpos] = tmp;
  sort(ary, evenpos++, oddpos--);
 }
}
```

5. codes presented below:

```
void func(int ary[], int v)
{
 int i=0;
 while(i<v)
 {
  if(ary[i]%2==0 && ary[v]%2==0)
  {
   int tmp = ary[i+1];
   ary[i+1] = ary[v];
   ary[v] = tmp;
   i++;
  }
  else if(ary[i]%2==0 && ary[v]%2==1)
  {
   i++;
   v--;
  }
  else if(ary[i]%2==1 && ary[v]%2==0)
  {
   int tmp = ary[i];
   ary[i] = ary[v];
   ary[v] = tmp;
   i++;
   v--;
  }
  else if(ary[i]%2==1 && ary[v-1]%2==1)
  {
   int tmp = ary[i];
   ary[i] = ary[v-1];
   ary[v-1] = tmp;
   v-=2;
  }
 }
}
```

- (2.3)

    1. prove:

```
d(n) = O(g(n)) means there exists c1>0 and n1>1 s.t
d(n) <= c1(f(n))
------------------------
e(n) = O(g(n)) means there exists c2>0 and n2>1 s.t
e(n) <= c2(g(n))
------------------------
d(n)-e(n) <= c1(f(n))-c2(g(n))
if there exitst c3>=max(c1,c2) and n3>=max(n1,n2), the situation exists,
But, when n3<max(n1,n2), then d(n)-e(n) is necessarily O(f(n)-g(n)).
```

2. prove:

```
(n+1)^5=x^5+5x^4+10x^3+10x^2+5x+1  <  5*x^5
There exist c>5 and n0>1, so (n+1)^5 is O(n^5)
```

3. In the worst case for Al’s algorithm, Bob’s alogorithm could be performing better . Big O doesn’t guarantee the speed of the algorithm; it just estimates running time **roughly**. In this case, when n<100, Al’s algorithm running in O($n^2$) time is faster.

- (2.4)

  1. 在這次作業當中，單一log所有資訊由data這個class所儲存；儲存所有log資料的結構：std::map<unsigned long long int, std::map<unsigned long long int, std::map<unsigned int, std::vector<data> > > >。最外層的map key為userID，中層的map key為adID，內層map key為queryID，最內的vector，儲存所有userID、adID、queryID皆相同但其他性質卻不同的log。
  之所以最外層的map key為userID乃因為4個function都有使用userID作為參數，其次adID與queryID也出現多次，故使用這些properties作為key，可以有效快速篩選出目標log。
  補充：之所以使用stl map原因有二。第一，map查詢的時間複雜度為O(log n)，相較於O(n)快上許多。第二，map有提供find()函式，讓使用者能夠迅速檢查map中是否存在該key，方便實作click()與impressed()函式。