**DSA HW2**

B03902048 林義聖

**2.1-(1)**

The function is trying to return a reference to a value that is going to go out of scope when the function returns. When the function returns, local variables will be recycled by the program. This would mean the caller receives a reference to garbage. This will result in a run-time error.

**2.1-(2)**

If we use integer variable to receive the return value from the function, the memory obtained by “new” in the function cannot be recycle by the program.

**2.2-(1)**

```
function findRepeatedIntegers(integer array B)
    integer array Counter
    for all element in B do
        Counter[element] ← Counter[element] + 1
    end for
    for all element in Counter do
        if Counter[element] > 1 then
            print element
        end if
    end for
end function
```

**2.2-(2)**

- Memory Layout: $[a_{00}]$ $[a_{10}]$ $[a_{11}]$ $[a_{20}]$ $[a_{21}]$ $[a_{22}]$ $[a_{30}]$ … $[a_{ij}]$ … $[a_{nn}]$
- Getting Value Function:

```
function getValue(integer array myarray)
    two-dimensional integer array ans
    integer ptr ← 0
    for i from 0 to n-1 do
        ptr ← ptr + i
        for j from 0 to i do
            ans[i][j] ← myarray[ptr + j]
        end for
        for k from i+1 to n-1 do
            ans[i][k] ← 0
        end for
    end for
    return ans
end function
```

- Putting Value Function:

```
function putValue(integer array myarray, two-dimensional integer array input)
        integer ptr ← 0
        for i from 0 to n-1 do
                ptr ← ptr + i
                for j from 0 to i do
                        myarray[ptr + j] ← input[i][j]
                end for
        end for
end function
```

**2.2-(3)**

```
function compare2CircularLinkedList(Node* startofL, Node* startofM)
        lengthofL ← lengthofLinkedList(startofL)
        lengthofM ← lengthofLinkedList(startofM)
        if lengthofL != lengthofM then
                return False
        end if
        Node* ptr ← startofM
        length ← lengthofL
        while length > 0 do
                if simpleCompare(startofL, ptr, lengthofL) then
                        return True
                end if
                ptr ← ptr.nextNode
                length ← length - 1
        end while
        return False
end function

function lengthofLinkedList(Node* startPoint)
        integer length ← 1
        Node* ptr ← startPoint.nextNode
        while ptr != startPoint do
                length ← length + 1
                ptr ← ptr.nextNode
        end while
        return length
end function
```

```
function simpleCompare(Node* startofA, Node* startofB, integer length)
        Node* ptr1 ← startofA
        Node* ptr2 ← startofB
        while length > 0 do
                if value of ptr1 not equal to value of ptr2 then
                        return False
                end if
                ptr1 ← ptr1.nextNode
                ptr2 ← ptr2.nextNode
                length ← length - 1
        end while
        return True
end function
```

**2.2-(4)**

```
void rearrangeIntArray(int* array, int current_pos, int end_pos){
        if(current_pos > end_pos)
                return;
        int tmp;
        if(array[current_pos] % 2 == 0)
                rearrangeIntArray(array, current_pos+1, end_pos);
        else{
                for(int i=current_pos; i<=end_pos; i++){
                        if(array[i] % 2 == 0){
                                tmp = array[current_pos];
                                array[current_pos] = array[i];
                                array[i] = tmp;
                                break;
                        }
                }
                rearrangeIntArray(array, current_pos+1, end_pos);
        }
        return;
}
```

**2.2-(5)**

```
int* rearrangeIntArray_oneloop(int* array, int length){
        int front_index = 0, tail_index = length - 1;
        int* answer = new int[length];
        for(int i=0; i<length; i++){
                if(array[i] % 2 == 0){
                        answer[front_index] = array[i];
                        front_index++;
                }
                else{
                        answer[tail_index] = array[i];
                        tail_index--;
                }
        }
        return answer;
}
```

**2.3-(1)**

show that if d(n) is O(f(n)) and e(n) is O(g(n)), then d(n) - e(n) is not necessarily O(f(n) - g(n)).
If d(n) = 2n - 1 and e(n) = n, then f(n) = n and g(n) = n.
d(n) - e(n) = n - 1 therefore d(n) - e(n) is O(n), but not O( f(n) - g(n) ) = O(0).
This shows that d(n) - e(n) is not necessarily O(f(n) - g(n)).

**2.3-(2)**

$(n+1)^5 = n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1$,
$| n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1 | < | n^5 + 5n^5 + 10n^5 + 10n^5 + 5n^5 + n^5 | = | 32n^5 |$
Since $(n+1)^5 \leq 32 \times n^5$ when $n \geq 1$, the function $(n+1)^5$ is $O(n^5)$.

**2.3-(3)**

Al's O(n log n)-time method comes from $f(n) \leq c_1 \times g(n) = c_1 \times (n \log n)$, and Bob's $O(n^2)$-time method comes from $d(n) \leq c_2 \times h(n) = c_2 \times (n^2)$.
In some cases, $c_1 \times (n \log n)$ may greater than $c_2 \times (n^2)$ when $n < 100$.
Let $n = 100$, $c_1 \times (n \log n) = c_1 \times (100 \times \log 100) = c_1 \times 200$, $c_2 \times (n^2) = c_2 \times (n^2) = c_2 \times 10000$.
If $c_1$ is 200 more times than $c_2$, then Bob's method will run faster than Al's method.
This shows that $O(n^2)$-time method may run faster than O(n logn)-time method when $n < n_0$ = some constants.

**2.4-(1)**

1. 實做方法
    a. 讀檔：

    我使用C語言的FILE I/O，將從command-line取得的參數作為檔名開啟檔案，並在迴圈中使用 fscanf() 將每一行的資料以11個 integer 和1個 unsigned long long integer 儲存起來。我認為C語言的基本輸入函式應該有略勝於C++的速度。

    b. 儲存結構：

    在程式一開始的部份，我就先用 new 取得兩個大小為24000000（兩千四百萬）的 vector array（類似動態二維陣列）。我將這兩個 vector array 中的一個以 UserID 作為 index，以自訂的 class 為內容存放 AdID、 QueryID、 Position、 Depth、 Click、Impression 這些資料；另一個 array 則是以 AdID 為 index，以自訂的 class 為內容存放 DisplayURL、AdvertiserID、KeywordID、TitleID、DescriptionID、UserID 這些資料。

    c. 儲存方式：

    我將讀取檔案過程中讀出的資料簡單地用 vector 的 push_back() 放進以 UserID 或是 AdID 為索引的兩個 vector array 裡面。

    d. get()：

    利用參數中所得到的 UserID 在 vector array 中查找，可以在 O(1) 的時間下找到我需要的 array。再來，遍歷 vector 比對其餘參數的資料，如果符合就將 Click 和 Impression 加總，最後就可以得到所要求的結果。

    e. clicked()：

    同樣地，利用 UserID 查找到所要的 vector，並遍歷它找到 Click 值大於一的 AdID 和 QueryID，將這兩個值以 integer 存進 set 裡，並以 AdID 和 QueryID 的漸增序為則，自訂 struct 中的比較函式，如此一來，根據 STL set 的性質，我們就可以得到不重複且排序過的結果。

    f. impressed()：

    遍歷以 u1 和 u2 為 index 建立的兩個 vector，並將 impression 大於一的資料的 AdID 以 UserID 為分別，放入兩個 set 之中。由於在 set 中的資料是排序過的，所以我們可以以兩個 iterator 同步遍歷兩個 set。在遍歷的過程中，每當找到相同的 AdID 之後，就去查找以 AdID 為 index 建立的 vector array，並遍歷那個 vector，找到符合的 UserID，並將所需要的 DisplayURL、AdvertiserID、KeywordID、TitleID、DescriptionID 等資料放進一個 map，都存放進 map 之後，再逐個輸出。輸出完單筆 AdID 的 properties 之後，再繼續比對下一個相同的 AdID。

    g. profit()：

查找以 AdID 為 index 建立的 vector array，遍歷那個 vector，並將 UserID 以 set 存放。而後遍歷 set 中排序過的 UserID，並以 UserID 為 index 查找 vector array，遍歷 vector，加總 Click 和 Impression後，計算兩個相除的結果是否大於或等於參數中的 $\theta$ 值，並額外地考慮 Impression 的結果，之後將 UserID 輸出。

2. 時間
    a. 讀檔過程加上建立資料庫大約要花 20 分鐘
    b. impressed() 函式處理較極端的測資時要花較久的時間，約以秒為單位。profit() 函式約以十分秒為單位。而 get() 和 clicked() 函式則是以毫秒為單位，執行速度快。
3. 實做過程

    我大部分的時間是花在構思儲存結構的設計與選擇上，另外，有花一點時間與同學討論他們所使用的結構，以及他們在時做過程中在記憶體與執行時間上的困難。真正實做時所使用的結構，即為三天下來所想出的成果，過程中也有嘗試使用 multimap，但在最後實做過程中並沒有採用。希望在接下來的作業裡，可以放寬執行時間的限制，或是降低題目的難度，使我們可以嘗試親手撰寫這些好用的資料結構，已達到對各種資料結構更深入地了解。