CSE 431/531: Algorithm Analysis and Design	Spring 2022
Homework 1	
Instructor: Shi Li	**Deadline: 2/23/2022**

Your Name: RATIK DUBEY Your Student ID: 50363929

Problems	1	2	3	Total
Max. Score	20	25	35	80
Your Score				

Problem 1. For each pair of functions f and g in the following table, indicate whether $f = O(g)$, $f = \Omega(g)$ and $f = \Theta(g)$ respectively.

$f(n)$	$g(n)$	O	Ω	Θ
$\log_2 n$	$5\log_2(n^3)+3$			
$10n^2 - n$	$n^2 \log n$			
$n^3 - 4n^2 + 10$	n^2			

Prove $\lceil 10n\sqrt{n}\rceil + \lceil n\log n\rceil = O(n\sqrt{n})$.

Problem 2. Consider the following algorithm for sorting an array A of n numbers.

Algorithm 1 Sorting the integer array A, which is of size n

```
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        if A[i] > A[j] then t ← A[i], A[i] ← A[j], A[j] ← t
```

(2a) What does the pseudo-code "$t \leftarrow A[i], A[i] \leftarrow A[j], A[j] \leftarrow t$" do?

(2b) What is the running time of the algorithm? Briefly explain why. Your bound should be tight (that is, "the running time is $O(n^{10})$" is not considered as a correct answer).

(2c) Why is the algorithm correct? To answer the question, you just need to describe the property that the array A satisfies after each iteration i of the outer loop.

Problem 3. We are given a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, using the linked-list representation. You need to design an $O(n+m)$-time algorithm to decide between the following three cases:

(i) there is no topological-ordering for G, in which case your algorithm should output "none",

(ii) there is a unique topological-ordering for G, in which case your algorithm should output "unique", and

(iii) there are at least two different topological orderings for G, in which case your algorithm should output "multiple".

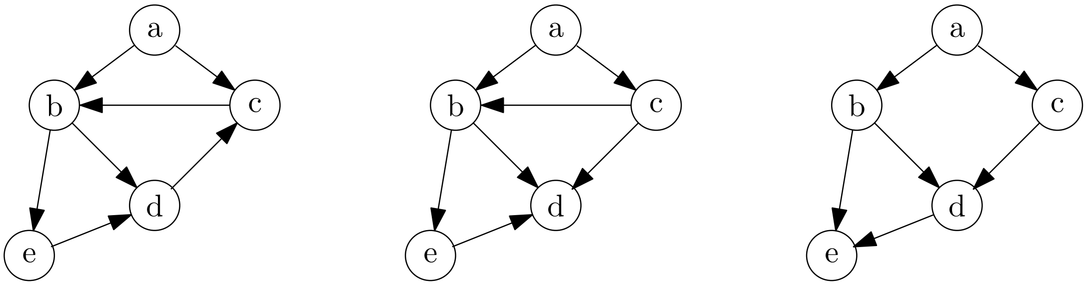

Figure 1: Example input graphs for Problem 3.

For example, consider the three graphs in Figure 1. The outputs for the left-side, middle and right- side graphs are respectively "none", "unique" and "multiple": There is no topological ordering for the left-side graph, there is a unique topological ordering (a, c, b, e, d) for the middle graph, and there are two different topological orderings (a, b, c, d, e) and (a, c, b, d, e) for the right-side graph.

Giving a pseudo-code for your algorithm is sufficient, if the correctness and running time can be easily seen.

1

$f(n)$	$g(n)$	O	Ω	Θ
$\log_2 n$	$5\log_2(n^3)+3$	YES	YES	YES
$10n^2 - n$	$n^2 \log n$	YES	No	No
$n^3 - 4n^2 + 10$	n^2	No	YES	No

$$[10n\sqrt{n}] + [n\log n] = O(n\sqrt{n})$$

$f(n) = 10n\sqrt{n} + n\log n \qquad g(n) = n\sqrt{n}$

$f(n) \le c\,g(n) \Rightarrow 10n\sqrt{n} + n\log n \le c(n\sqrt{n})$

Let $c = 11$ and $n_0 = 25$, for $n \ge 25$,

$10n\sqrt{n} + n\log n - c(n\sqrt{n}) \le 0$

$10n\sqrt{n} + n\log n - 11n\sqrt{n} \le 0$

$-n\sqrt{n} + n\log n \le 0$ (we know $\sqrt{n} \ge \log n$ for $n \ge 25$)

$\Rightarrow [10n\sqrt{n}] + [n\log n] \le n\sqrt{n}$

So, $[10n\sqrt{n}] + [n\log n] \in n\sqrt{n} \Rightarrow O(n\sqrt{n})$

② a> The pseudo-code does the following :

$t \leftarrow A[i]$ => stores the value of $A[i]$ in a temp. variable t

$A[i] \leftarrow A[j]$ => assigns the value of $A[j]$ to $A[i]$

$A[j] \leftarrow t$ => assigns the value of t i.e. the original value of $A[i]$ to $A[j]$

Effectively, the code interchanges the values of the i^{th} & j^{th} indices of array A.

② b> Running time of the algorithm : $O(n^2)$ and the bound is tight.

for $i \leftarrow 1$ to $n-1$ **do** $\rightarrow n$
    for $j \leftarrow i+1$ to n **do** $\rightarrow n$
        if $A[i] > A[j]$ **then** $t \leftarrow A[i], A[i] \leftarrow A[j], A[j] \leftarrow t$ $\rightarrow 1$

Outer for loop takes $O(n)$
Inner for loop also takes $O(n)$ for worst case
Innermost if condition takes $O(1)$ → constant time

2 c> The algorithm is correct because after each iteration of the outer loop, the value at the i^{th} index of the array is sorted.

Example → after the 1^{st} iteration the smallest value in A will be at index 0 (because indexing starts at 0 in python)

→ after the 2^{nd} iteration the second smallest value will be at index 1

→ and so on.... till the second largest value after which the loop stops, since only one value is left & that has to be the largest value.

Basically after every i^{th} iteration the array has been sorted upto the i^{th} index.

③

Pre-Requisites

- **Adjacency List**

```
adjList = [[] for _ in range(n)]
```

- **Incoming Edges Count List**

```
inCount = [0 for _ in range(n)]
for vertex in graph do
    for edge of vertex do
        adjList[vertex].append(edge)
        inCount[edge] = inCount[edge]+1
```

- Adjacency list used to get edges of a vertex in the graph
- inCount List used to know how many incoming edges does a vertex have

PSEUDO-CODE

```
L → empty List

S → empty stack
Append all vertices with no incoming edges to S
                                        (in Count used)
flag = False

while S not-empty do
    if len(S) > 1
        flag = True

    pop vertex n from S
    set L.next to n
    L = L.next

    for each vertex m with edge(e) from n to m do
        remove e from graph                  (adj List,
                                              incount
        if m has no incoming edges            used)
            append m to S

if graph has edges
    return None
else
    if flag == True
        return Multiple
    else
        return Unique
```

$\boxed{\text{Time: } O(n+m)}$