National University of Computer & Emerging Sciences, Karachi

Fall-2019 Department of Computer Science

Mid Term-2

5th November 2019, 11:00 AM – 12:00 PM

Course Code: CS302	**Course Name:** Design and Analysis of Algorithm
Instructor Name / Names: Dr. Muhammad Atif Tahir, Waqas Sheikh, Zeshan Khan	
Student Roll No:	**Section:**

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **4 questions** on **2 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 60 minutes. **Max Marks: 12.5**

Question # 1 [2+2=4 marks]

A) Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any sorting algorithm stable. How much additional time and space does your scheme entail?

B) Explain why the worst-case running time for bucket sort is $O(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \log n)$.

Solution:

A) Insertion sort and merge sort are stable. Heapsort and quicksort are not. To make any sorting algorithm stable we can preprocess, replacing each element of an array with an ordered pair. The first entry will be the value of the element, and the second value will be the index of the element. For example, the array [2; 1; 1; 3; 4; 4; 4] would become [(2; 1); (1; 2); (1; 3); (3; 4); (4; 5); (4; 6); (4; 7)]. We now interpret $(i; j) < (k; m)\ if\ i < k\ or\ i = k\ and\ j < m$. Under this definition of less-than, the algorithm is guaranteed to be stable because each of our new elements is distinct and the index comparison ensures that if a repeat element appeared later in the original array, it must appear later in the sorted array. This doubles the space requirement, but the running time will be asymptotically unchanged.

B) In the worst case, we could have a bucket which contains all n values of the array. Since insertion sort has worst case running time $O(n^2)$, so does Bucket sort. We can avoid this by using merge sort to sort each bucket instead, which has worst case running time $O(n \log n)$.

Question # 2 [3 marks]

As stated, in dynamic programming we first solve the sub problems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that we do not always need to solve all the sub problems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrix chain multiplication problem by always choosing the matrix A_k at which to split the sub product $A_i A_{i+1}$ ... A_j (by selecting k to minimize the quantity pi-1pkpj) before solving the subproblems. Proof that for dimensions $p_0;\ p_1;\ p_2;\ p_3;\ p_4 = 1000;\ 100;\ 20;\ 10;\ 1000$ is an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.

<u>Solution:</u>

Suppose that we are given matrices $A_1; A_2; A_3, and\ A_4$ with dimensions such that $p_0;\ p_1;\ p_2;\ p_3;\ p_4 = 1000;\ 100;\ 20;\ 10;\ 1000$. Then p0pkp4 is minimized when $k = 3$, so we need to solve the sub problem of multiplying $A_1A_2A_3$ and also A_4 which is solved automatically. By her algorithm, this is solved by splitting at $k = 2$. Thus, the full parenthesization is $(((A_1A_2)A_3)A_4)$. This requires 1000 x 100 x 20 + 1000 x 20 x 10 + 1000 x 10 x 1000 = 12; 200; 000 scalar multiplications. On the other hand, suppose we had fully parenthesized the matrices to multiply as $((A_1(A_2A_3))A_4)$. Then we would only require 100 x 20 x 10+1000 x 100 x 10+1000 x 10 x 1000 = 11; 020; 000 scalar multiplications, which is fewer than Professor Capulet's method. Therefore her greedy approach yields a suboptimal solution.

<u>Question # 3</u> **[3 marks]**

Given a rope of length n meters, the provided algorithm cuts the rope in different parts of integer lengths in a way that maximizes product of lengths of all parts. Execute the following algorithm to compute the optimal way of cutting the rope of length **12** meters.

$$\text{maxP}[n] = \begin{cases} 0 & if \quad n = 0 \\ 1 & if \quad n = 1 \\ \max\limits_{1 \le i \le n}(n, i * (n - i), i * maxP[n - i]) & if \quad n > 1 \end{cases}$$

<u>Solution:</u>

A 1-D array is filling as below

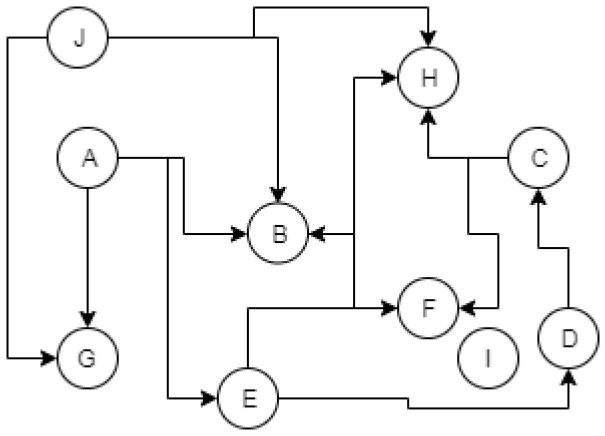
Loc	0	1	2	3	4	5	6	7	8	9	10	11	12
i0	0	1											
i1													
i2			2										
i3				NA=3									
i4					4								
i5						6							
i6							A3*3=9						
i7								12					
i8									18				
i9										A6*3=27			
i10											36		
i11												54	
i12													A9*3=81

<u>Question # 4</u> **[0.5+0.5+1.5=2.5 marks]**

A) Explain how one can check a graph's acyclicity by using breadth-first search.

B) Does either of the two traversals—DFS or BFS—always find a cycle faster than the other? If you answer yes, indicate which of them is better and explain why it is the case; if you answer no, give two examples supporting your answer.

C) Apply the topological sort to check if the provided graph (**Fig#1**) is a DAG (Directed Acyclic Graph) or not.



Fig#1

<u>Solution:</u>

A)

Pro/In	A	B	C	D	E	F	G	H	I	J
	0	3	1	1	1	2	2	3	0	0
A		2			0		1			
I										
J		1					0	2		
E		0		0		1		1		
G										
B										
D			0							

C						0		0		
F										
H										

$A, I, J, E, G, B, D, C, F, H$
$A, I, J, E, G, B, D, C, F, H \rightarrow total = 10 = |V|$
The Provided Graph is a **DAG**.

Or

DFS (Graph)

Start with A
DFS(A) is H, B, F, C, D, E, G, A
Then, DFS (I) is I
Then DFD(J) is B, G, J

Thus,
DFS (Graph) = H, B, F, C, D, E, G, I, J, A

Reverse is

A, J, I, G, E, D, C, F, B, H

Since there is no backward arrow, this is DAG

<u>BEST OF LUCK</u>