National University of Computer & Emerging Sciences, Karachi
Fall-2019 Department of Computer Science
Final Term
16th December 2019, 09:00 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 **09 questions** on **5 pages.**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 180 minutes. **Max Marks: 50**

Question # 1 **[(0. 25+0.25)*10=5 marks]**

Complete the following table.

Algorithm	**Worst Case**	**Write below whether the algorithm belongs to Dynamic Programming / Greedy Algorithms / None of them**
0/1 Knapsack using Dynamic Programming		
0/1 Knapsack using Brute Force		
Breadth First Search		
Depth First Search		
Kruskal's algorithm for finding MST		
Prim's algorithm for finding MST		
Shortest path by Dijkstra		
Shortest path by Bellman Ford		
Matrix Chain Multiplication using Dynamic Programming		
Matrix Chain Multiplication using Brute Force		

Question # 2 **[3+1+2=6 marks]**

(a) What is meant by P and NP Problems? Explain P = NP
(b) Why it is important to find approximate solutions for NP Complete Problems
(c) Explain the difference between greedy algorithms and dynamic programming

Question # 3 **[5 marks]**

What is convex Hull? Design $O(N^3)$, $O(Nh)$ and $O(NlogN)$ algorithms to solve convex hull problem.

Question # 4 **[0.5+5+0.5=6 marks]**

Floyd-Warshall can be used to determine whether or not a graph has *transitive closure*, i.e. whether or not there are paths between all vertices using the following procedure:

- Assign all edges in the graph to have weight = 1
- Run Floyd-Warshall
- Check if all $d_{ij} < n$ i.e. all values in matrix D is less than all values in matrix n.

Use the above algorithm to determine whether the graph in ***Figure 1*** has *transitive closure* or not.

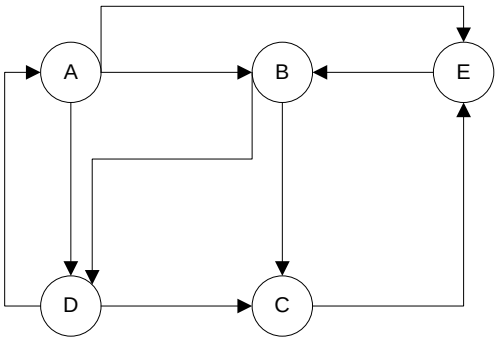

Figure 1

Question # 5 **[4 marks]**

Consider the following APPROX-VERTEX-COVER algorithm. Proof that this algorithm is 2-approximation method for VERTEX-COVER.

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E'=G.E;
    while(E' ≠ ∅ ){
        Randomly choose a edge (u,v) in E', put u and v into C;
        Remove all the edges that covered by u or v from E'
    }
    Return C;
```

Question # 6 **[4 marks]**

An Instance (X, F) of the set-covering problem consists of a finite set X and a family F of subset of X, such that every element of X belongs to at least one subset of F:

$$X = \bigcup_{S \in F} S$$

We say that a subset $S \in F$ covers all elements in X. Our goal is to find a minimum size subset $C \subseteq F$ whose members cover all of X.

$$X = \bigcup_{S \in C} S$$

Algorithm 1: GREEDY-SET-COVER (X, F)

```
1 U ← X
2 C ← ∅
3 While U ≠ 0
4    do select an S ∈ F that maximizes |S ∩ U|
5       U ← U − S
6       C ← C ∪ {S}
7 return C
```

Consider each of the following words as a set of letters: {arid, dash, drain, heard, lost, nose, shun, slate, snare, thread}. Show which set cover GREEDY-SET-COVER produces, when we break ties in favor of the word that appears first in the dictionary.

Question # 7 **[4+3=5 marks]**

Analyze the time complexity of the following algorithms and write in the most appropriate asymptotic notations:

A)

```
double Algo(Point points[], int i=0, int j=n)
{
   if (j < i+2)
      return 0;
   double res = MAX;
   for (int k=i+1; k<j; k++)
        res = min(res, (Algo(points, i, k) +
Algo(points, k, j) +  cost(points, i, k, j)));
   return  res;
}
```

Note: There are n points (i.e. points []) and the values of I and j are 0 and n-1 respectively.

B)

```
MatrixChainOrder(int dims[])
    n = dims.length - 1;
    for (i = 1; i <= n; i++)
        m[i, i] = 0;
    for (len = 2; len <= n; len++)
        for (i = 1; i <= n - len + 1; i++)
            j = i + len - 1;
            m[i, j] = MAXINT;
            for (k = i; k <= j - 1; k++)
                cost = m[i, k] + m[k+1, j] + dims[i-1]*dims[k]*dims[j];
                if (cost < m[i, j])
                    m[i, j] = cost;
                    s[i, j] = k;
```

Question # 8 **[3*3=9 marks]**

Which algorithm will you prefer to use in given scenarios, support your answer with some brief discussion?

A) A maximum bottleneck path, between a source *s* and a target *t* that allows the most flow to go through it. The amount of flow that can go through the bottleneck is equivalent to the weight of the minimum weight edge on that path. Because this edge effectively limits the amount of flow that can go from *s* to *t* in any path in the graph. Which algorithm will you apply to find the widest path and bottleneck edges?

B) You need to determine the total area of rainfall by analyzing a subset of all sensors that send the signal of the waterfall in a specific region to your computer, which algorithm you will apply to find out the total area of rainfall.

C) Suppose you are given a graph $G = (V, E)$ with edge weights $w(e)$ and a minimum spanning tree T of G. Now, suppose a new edge $\{u, v\}$ is added to G. Describe (in words) a method for determining if T is still a minimum spanning tree for G.

Question # 9 **[4 marks]**

Suppose that you've computed an MST T* for a graph $G = (V, E)$. Afterwards, you add a new edge $(u, v) \notin E$ to the graph with cost c(u, v). Depending on the shape of the graph, the initial edge weights, and the weight of (u, v), your old MST T* might not necessarily be an MST of the new graph $G' = (V, E\,\{(\cup\ u, v)\})$.

Assume that all edge weights in G' are distinct (which means that the edge weights in G are also distinct). Design an O(n)-time algorithm (pseudo code) that determines whether T* is an MST of G'.

BEST OF LUCK