Name: ____________________ Roll #: ____________________ Section: ______________

National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Design and Analysis of Algorithms**	**Course Code:**	**CS302**
Program:	**BS(Computer Science)**	**Semester:**	**Spring 2019**
Duration:	**180 Minutes**	**Total Marks:**	**90**
Paper Date:	**27-May-19**	**Weight**	**45%**
Section:	**ALL**	**Page(s):**	**10**
Exam:	**Final Exam**		

Instruction/Notes: Attempt the examination on the question paper and write concise answers. You can use extra sheet for rough work. Do not attach extra sheets used for rough with the question paper. Don't fill the table titled Questions/Marks.

Question	1 – 7	8	9	10	Total
Marks	/45	/15	/15	/15	/90

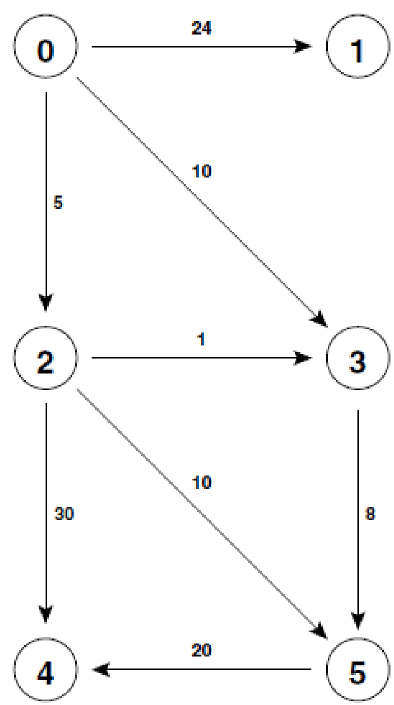
Q1) For each of the following statements, state true or false. **[5 Marks]**

a. If G is a simple graph with n vertices and n−1 or more edges, then G is connected. T/**F**
b. If a simple graph G has n vertices and n or more edges, then G contains a cycle. **T**/F
c. Dijkstra's algorithm can find shortest paths in a directed graph with negative weights, but no negative cycles. T/**F**
d. Prim's algorithm for computing the MST only work if the weights are positive. T/**F**
e. A graph where every edge weight is unique (there are no two edges with the same weight) has a unique MST. **T**/F

Q2) Simulate Dijkstra's algorithm on the edge-weighted graph below, starting from vertex 0. **[5 Marks]**

Fill in the following table:

Vertex	Shortest Path from vertex 0	Predecessor (parent) vertex in shortest path
1	24	0
2	5	0
3	6	2
4	34	5
5	14	3

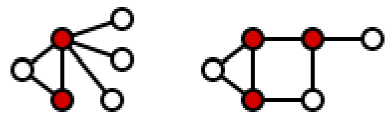


Name: ____________________ Roll #: ____________________ Section: ______________

Q3) How many times the partition function will be called in the worst case, during the recursive execution of quick sort if array to be sorted has n elements? Justify your answer. **[5 Marks]**

O(n) times because each element can be a pivot only once.

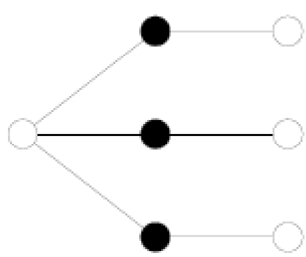
Q4) Let *G* = (*V*, E) be an undirected graph. A node cover of *G* is a subset *U* of the vertex set *V* such that every edge in *E* is incident to at least one vertex in *U.* A minimum node cover is one with the fewest number of vertices.



shaded vertices shows minimum node cover

Consider the following greedy algorithm for this problem: Select the vertex v of maximum degree (ties may be broken arbitrarily) and include it in the cover. Remove vertex v and all the edges that are incident on v from G repeat the same until G has 0 number of edges. Either prove that this greedy strategy always gives the optimal answer or disproof it using counter example. **[5 Marks]**

In the example below greedy will pick white vertices but optimal solution is black



Q5) You have been given an integer array *A* of size *N*. Each element of the array ranges between *1* and 10^5. You need to find the frequency of each distinct element of the array. The elements need to be present in the output in ascending order (sorted). You need to print the value and then frequency of each distinct element. Give high level description of algorithm to solve this problem in O(n) time. **[5 Marks]**

Use count sort

Q6) What is the best bound on the running time of the algorithm in the box below? **[5 Marks]**

```
count ← 0
For i ← 1 to n
    For j ← 1 to i
        For k ← 1 to i*i
            count ← count + 1
```

a) $O(n\lg n)$
b) $O(n\sqrt{n})$
c) $O(n^3)$
d) $\mathbf{O(n^4)}$
e) $O(n^5)$

Q7) Exams are over! You've rented a car and are ready to set out on a long drive on the Strange Highway. There are n tourist stores on the Strange Highway, numbered 1, 2, …, n and you would want to stop at these and buy some souvenirs (only one souvenir may be bought from a store and all souvenirs everywhere have the same price). You are a greedy shopper and are most interested in maximizing the total discount you get on your shoppings. You know that each store i offers a discount d_i on a souvenir. But it's a strange highway after all. It turns out that you can only buy a souvenir from a store i if you have not bought anything from the previous f_i stores. For example, if $f_6 = 3$ then you can only buy a souvenir from store 6, and get the discount d_6, if you haven't bought anything from stores 3, 4, and 5. All the d_i and f_i are known to you in advance. You have recently learnt the DP technique in your algorithms course and wish to apply it here in order to maximize your total discount under the given constraints. After some brain-storming, you think you can define the optimal sub-problem in one of the following two ways:

A. $D[i]$ = max total discount when store i is the last store where you buy a souvenir.
B. $D[i]$ = max total discount for the trip till store i whether buying at store i or not.

Which of the following recurrences correctly computes $D[i]$ for option (A) above and which one does it for (B). Write the correct option number in the table below. **[10 Marks]**

(A)	(v)
(B)	(iii)

(i) $D[i] = D[i - f_i - 1] + d_i$
(ii) $D[i] = \max(D[i-1] + d_i, D[i-1])$
(iii) $D[i] = \max(D[i - f_i - 1] + d_i, D[i-1])$
(iv) $D[i] = \max_{1<k<(i-f_i)} \{D[k] + d_i, D[i - f_i - 1]\}$
(v) $D[i] = \max_{1<k<(i-f_i)} \{D[k] + d_i\}$
(vi) $D[i] = \operatorname{sum}_{1<k<(i-f_i)} \{d_k\}$

Having identified the recurrences, which one do you think results in a better DP algorithm in terms of time complexity? **(A)** or **(B)**, Justify your answer. **[5 Marks]**

Answer: (B)

Q8) You are given a weighted undirected graph G(V, E) and its MST T(V, E'). Now suppose an edge (a, b) ϵ E' has been deleted from the graph. You need to devise an algorithm to update the MST after deletion of (a, b).

Describe in words an algorithm for updating the MST of a graph when an edge (a, b) is deleted from the MST (and the underlying graph). It's time complexity must be better than running an MST algorithm from scratch. State and explain the time complexity of your algorithm.

You can assume that all edge weights are distinct, that the graph has E edges and V vertices after the deletion, that the graph is still connected after the deletion of the edge, and that your graph and your MST are represented using adjacency lists. [**6 Marks**]

The following algorithm has time complexity O(E). Run DFS twice on what remains of the MST, once starting on a and once starting on b to determine the two connected components A and B, which you store in two HashSets. This takes O(E). Then iterate through all edges going out from the elements of A. If an edge leads to an element of B (can be check in O(1) using HashSet), check whether it is the cheapest edge so far. If it is cheaper than any other edge so far, store it. So, finding the cheapest edge can also be done in O(E). In the end, add the cheapest edge to the MST.

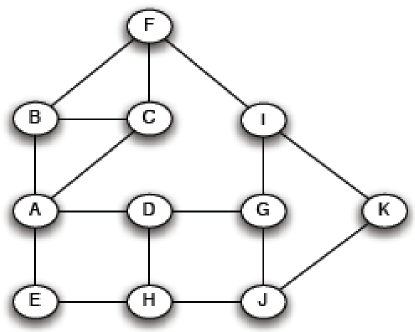
Name: ____________________ Roll #: ____________________ Section: ______________

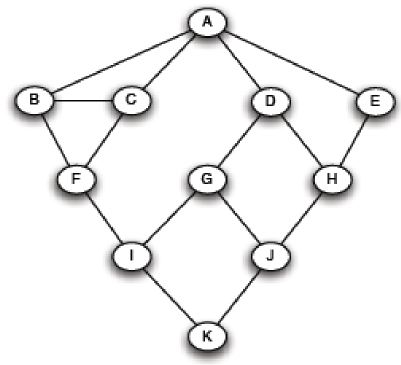
Give complete pseudo code of the algorithm **[7 Marks]**

What is the running time of your algorithms? **[2 Marks]**

Q9) Recall that BFS can be used to solve the problem of single source shortest path for unweighted graph G(V, E). Also BFS only finds one shortest paths between s and x, where s is the source vertex and x is any other vertex of the graph. But shortest path may not be unique. Suppose we want to count the number of distinct shortest paths between s and x for each x, where x∊V. For example in the graph given below if A is the source vertex then the number of distinct shortest paths for each pair are:

From A to A: 1
From A to B: 1
From A to C: 1
From A to D: 1
From A to E: 1
From A to F: 2
From A to G: 1
From A to H: 2
From A to I: 3
From A to J: 3
From A to K: 6





You are required to design a linear time algorithm that counts the number of distinct shortest paths between a given source vertex s and all the other vertices in the graph. You can assume that graph is represented using adjacency list representation.

Hint: you can use the tree structure of BFS to count the distinct shortest paths.

Give detailed explanation of your algorithm **[6 Marks]**

Use the following idea: Start BFS with the start node but instead of saving only one vertex as parent, store list of parents if more than one nodes will give the same distance. Number of unique shortest paths from s to s is 1. For all other vertices, number of distinct shortest paths are sum of distinct shortest paths of their parents.

Give pseudo code of your algorithm **[7 Marks]**

```
Modified_BFS(G(V,E), s)

        distinct_paths[1..v]
        color[1..v]
        pi[1..v] //each element is a list
        dist[1..v]
        for all v in V
                distinct_paths[v]=0
                color[v]=white
                dist[v]=inf
                pi[v]=null

        color[s]=grey
        dist[s]=0
        distinct_path[v]=1
        make FIFO queue Q
        Q.insert(s)
        while(Q is not empty)
                u = Q.remove()
                for each v adj to u
                        if(color[v]==white)
                                color[v]=grey
                                dist[v] = dist[u]+1
                                pi[v].insert(u)
                                distict_path[v] += distint_path[u]
                                Q.insert(v)
                        else
                                if(dist[v] == dist[u+1])
                                        pi[v].insert(u)
                                        distict_path[v] += distint_path[u]

                color[u]=black
```

Name: ____________________ Roll #: ____________________ Section: ______________

Argue why running time is linear **[2 Marks]**

Time complexity is same as of BFS, so Linear time

Q10) You're given an n x n matrix, M, of integers. The integers in each row and each column are unique and sorted. Following is an example of a 4x4 matrix with this property:

-1	6	8	11
9	13	17	18
11	16	19	21
15	17	20	23

Sample 4x4 matrix, M

Your goal is to write a search algorithm to find a key x, if it exists, in the matrix M. You need to write two versions of the algorithm.

(i) [5 points] Write a search algorithm with running time O(nlgn). Write your algorithm as a list of clearly specified pseudo-code steps.

//simple row by row binary search…

Write a search algorithm with running time O(n). Write your algorithm as a list of clearly specified pseudo-code steps. **[7 Marks]**
Hint: Use divide and conquer strategy

```
bool findKey(int M[][NUM_DIMS], int rtl, int ctl, int rbr, int cbr, int x, int & r, int & c){
    //performs binary search on rows
    if(rtl<=rbr){
        int rmid=(rtl+rbr)/2;

        //scan middle row, this is O(n)
        bool found=scanrowforkey(M, rmid, ctl, cbr, x, c);

        if(found){
            r=rmid;//(r, c) contain the key coordinates
            return true;
        }else{
            //split in O(1), students can give simple pseudo-code for this function
            int rtl1, ctl1, rbr1,cbr1, rtl2, ctl2, rbr2,cbr2;
            split(rtl,ctl,rbr,cbr,rtl1,ctl1,rbr1,cbr1,rtl2,ctl2,rbr2,cbr2,rmid,c);

            return (findKey(M,rtl2,ctl2,rbr2,cbr2,x,r,c) || findKey(M,rtl1,ctl1,rbr1,cbr1,x,r,c));
        }
    }else return false;
}
```

//basic idea

--- scan the middle row for x
--- if x is found return that (r, c)
--- if c is the column of the first element in the middle row bigger than x, split the matrix into two sub-matrices, (matrices to the north-east and south-west of c) and make two recursive calls
> The total size of these two recursive calls would be half the size of the original call

For an $N \times N$ matrix,
$T(N) = T\left(\frac{N}{a}\right) + T\left(\frac{N}{b}\right) + O(N), where, \frac{1}{a} + \frac{1}{b} = \frac{1}{2}$, this is $O(N)$

Argue why the running time is $O(n)$ **[3 Marks]**