National University of Computer and Emerging Sciences, Lahore Campus

Course:	Design & Analysis of Algorithms	Course Code:	CS-2009
Program:	BS (Computer Science)	Semester:	Spring 2022
Duration:	180 Minutes	Total Marks:	61
Paper Date:	24-June-22	Section:	ALL
Exam:	Final Exam	Page(s):	10
Name		Roll Number	

Instruction/Notes: Ample space is provided for rough work; NO EXTRA sheets will be provided.

Question	1-5	6-8	9	10	11	Total
Marks	/16	/15	/10	/10	/10	/61

Q1) Devise the recurrence relation for the following recursive function and then solve it for calculating time complexity in Big O or Theta notation of this algorithm. Show complete working. **[5 Marks]**

```
Mystery(n){
        if(n==0)
                return n;
        else
                int x=0;
                for(i=1;i<n-1;i++)
                        x=x+i;
                return x+Mystery(n-1);
}
```

Solution:

Recurrence: $T(n) = T(n-1) + O(n)$

Time complexity = $O(n^2)$

Q2) Which of the following sorting algorithms in its typical implementation gives best performance (in terms of time complexity) when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced). **[2 Marks]**

a) QuickSort
b) MergeSort
c) Insertion sort

Solution: Insertions sort O(n)

Q3) In an unweighted, undirected connected graph, the shortest path from a node S to every other node is computed most efficiently, in terms of time complexity by: **[2 Marks]**

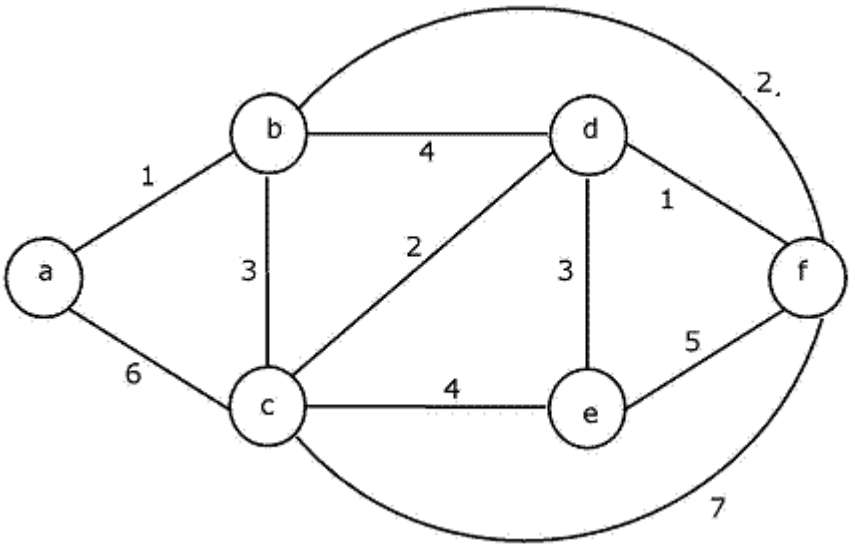
I. Dijkstra's algorithm starting from S
II. Floyd Warshall's algorithm

III. Performing a DFS starting from S
IV. Performing a BFS starting from S

Solution

Performing a BFS starting from S

Q4) Consider the following graph: [2 Marks]



Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm?

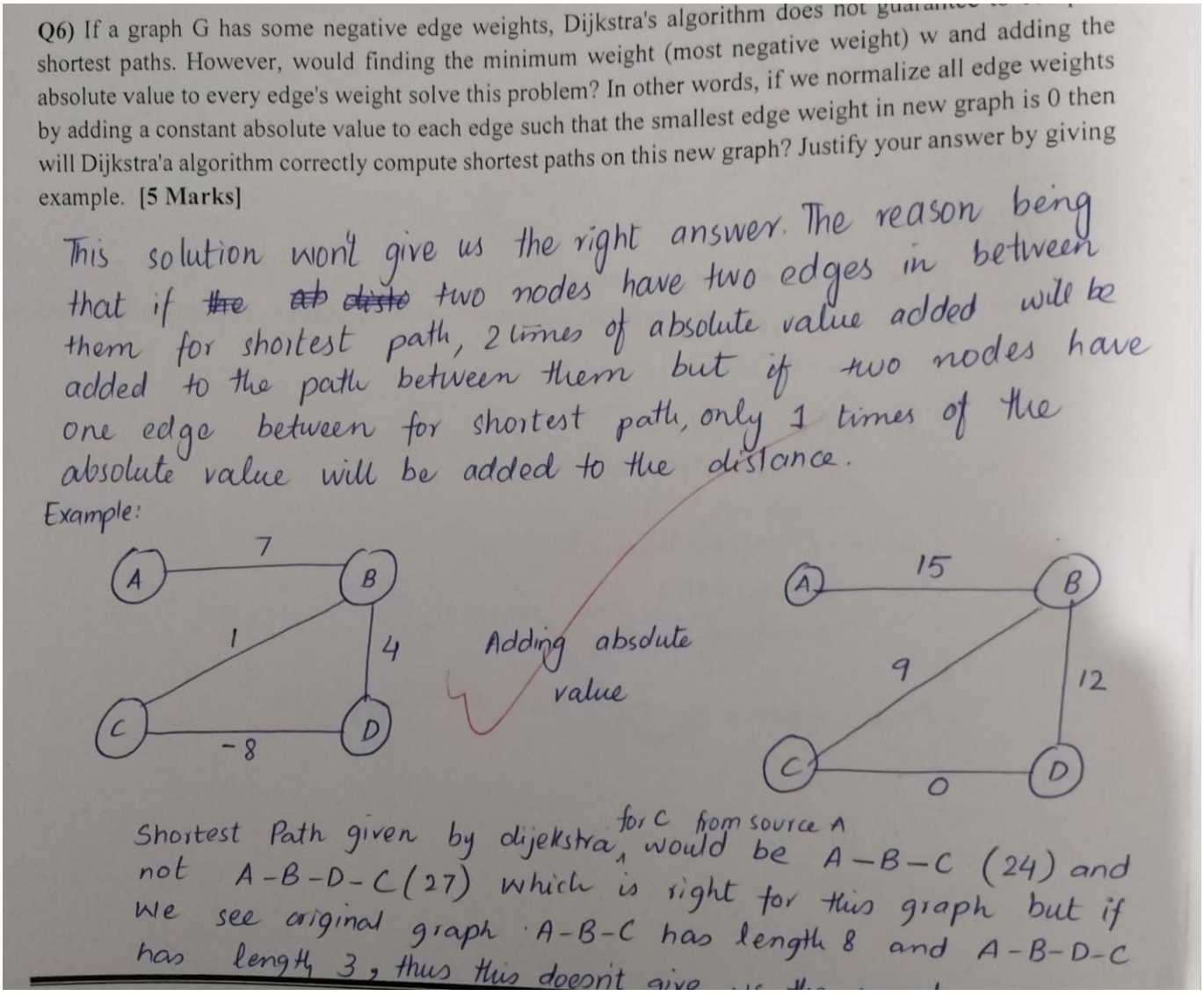
I. (a—b),(d—f),(b—f),(d—c),(d—e)
II. (a—b),(d—f),(d—c),(b—f),(d—e)
III. (d—f),(a—b),(d—c),(b—f),(d—e)
IV. (d—f),(a—b),(b—f),(d—e),(d—c)

Q5) Show how to sort n integers in the range 0 to $n^4 - 1$ in O(n) time. **[5 Marks]**
Solution: Use radix sort and use n as base of the numbers. Radix sort takes $O((n+b) * \log_b(k))$. Here $b = n$, $k = n^4$

Q6) If a graph G has some negative edge weights, Dijkstra's algorithm does not guarantee to compute shortest paths. However, would finding the minimum weight (most negative weight) w and adding the absolute value to every edge's weight solve this problem? In other words, if we normalize all edge weights by adding a constant absolute value to each edge such that the smallest edge weight in new graph is 0 then will Dijkstra'a algorithm correctly compute shortest paths on this new graph? Justify your answer by giving example. **[5 Marks]**

Solution:

No



Q7) You have an unsorted array, A, of unique numbers. You know its median number in advance. Now you wish to arrange the elements of A in such a way that the numbers $[i]$ and $A[i + 1]$ are successively in the increasing and decreasing order. That is, A[1]< A[2], A[2]> A[3], A[3]< A[4]… and so on. A friend of

yours claims that this is possible in O(n). Is her claim True / False. Justify your answer. If you are using an algorithm discussed in class then just name the algorithm and write the additional pseudocode required for to solve this problem. **[5 Marks]**

Solution 1:

The claim is **True**; following is linear time algorithm to solve the given problem:

```
Let B[1..n] be a new array
i = 1
j=2
for k = 1 to n
        if A[k] ≤ median
                B[i]=A[k]
                i=i+2
        else
                B[j]=A[k]
                j=j+2
return B
```

Solution 2:

Use partition function of Quick sort to partition the array around median in O(n) time.

```
x = 1
For (i = 1 to n/2)
   B[x] = A[i]
   B[x+1] = A[n/2+i]
   x= x+2
```

Q8) Given the alphabets from a text and their frequencies, we want to compress the text using Huffman's code. **[5 Marks]**

Alphabet	a	b	c	d	e	f
Frequency	40	10	8	15	25	2

Draw the binary tree for Huffman Code, and using that tree, write down the binary code of each alphabet.

Alphabet	**a**	**b**	**c**	**d**	**e**	**f**
Code						

Solution

Alphabet	**a**	**b**	**c**	**d**	**e**	**f**
Code	**0**	**1110**	**11111**	**110**	**10**	**11110**

Q9) Given a matrix M * N of integers where each cell has a cost associated with it, the cost can also be negative. Find the minimum cost to reach the first cell (0, 0) of the matrix from its last cell (M-1, N-1). We can only move one unit left (Column No - 1), one unit up (Row No - 1), and one unit in top diagonal (Row No - 1, Column No - 1). **[10 Marks]**

For example from index (i , j) you can move to (i , j-1), (i-1 , j), and (i-1, j-1)

where i = row no and j = column no.

Example

4	7	8	6	4
-6	7	3	9	2
3	8	1	-2	4
7	1	7	3	7
2	9	8	9	3

4	7	8	6	4
-6	**7**	3	9	2
3	8	**1**	**-2**	4
7	1	7	**3**	7
2	9	8	9	**3**

Path with minimum cost = 3 -> 3 -> -2 -> 1 -> 7-> -6 -> 4 = 10

Provide a Dynamic Programming solution for it.

a) Provide recurrence for sub-problem

Solution 1

(a) `C[i,j] = D[i,j] + min (C[i+1,j], C[i+1,j+1], C[i,j+1])`

(b)

Let C[0..M-1, 0..N-1] be a new matrix

C[M-1,N-1] = A[M-1,N-1]

for i=M-2 **dowto** 0

C[i,N-1] = A[i,N-1] + C[i+1,N-1]

for j=N-2 **dowto** 0

C[M-1,j] = A[M-1,j] + C[M-1,j+1]

for(i: m-1 to 0)

for (j: n-1 to 0)

C (i , j) = ***Min*** (**C**[i][j + 1] , **C**[i + 1][j] , C[i+1][j+1]) + **A**[i][j]

The final answer will be in C[0][0]

(c) Time Complexity = O(m*n)

Solution 2

for(0 to m)

for (0 to n)

C (i , j) = ***Min*** (**C**[i][j - 1] , **C**[i - 1][j] , C[i-1][j-1]) + **A**[i][j]

The final answer will be in C[M-1][N-1]

b) Provide pseudo code for DP solution

c) Provide time complexity of DP solution

Q10) Suppose you are given a weighted undirected graph G, and its subgraph T. The subgraph T and the graph G are represented in the form of adjacency lists. [10 Marks]

(a) Design an efficient algorithm that decides whether T is **spanning tree** of the given graph or not. Hint: Recall the definition of a spanning tree [7 Marks]

```
if |G.V| ≠ |T.V|
    return FALSE
if |T.E| ≠ |T.V| − 1
    return FALSE
//Run DFS (T) to check connectivit
for each vertex u ∈ T.V
        u.color = WHITE
        u.π = NIL
time = 0
visits = 0
for each vertex u ∈ T.V
        if u.color == WHITE
                DFS-VISIT (T, u)
                visits = visits + 1
        if visits > 1
                return FALSE
return TRUE
```

(b) Design an efficient algorithm that decides whether T is **minimum spanning tree (MST)** of the given graph G or not. You are given the total weight of MST of graph G. [3 Marks]

```
//Let M be the weight of MST
X = 0
for each vertex u ∈ T.V
        for each v ∈ T.Adj[u]

                X = X + w(u, v)
if X/2 == M
        return TRUE
else
        return FALSE
```

Q11) "$x - y$ **Logistics**" is a carrier company which has its own transportation network spread across major cities of the country. Most (i.e. around 95%) of the business of the company is related to the transportation of goods **from** city x **to** city y (hence, the name $x - y$ **Logistics**). Every now and then, the company also receives offers from other transportation companies to use their (i.e. other companies') transportation services between different cities on discounted rates. Sometimes these offers are so lucrative (i.e. the discounted price are very low) that $x - y$ **Logistics** prefer to avail some other company's transportation service instead of using their own.

For any such offers, the company has to respond affirmative *in a very short time* (otherwise, that offer may be taken by some other competitors). However, a hasty decision may also has its consequences. Hence, the company has decided to build a system which tells *in real time* whether a transportation offer from city p to city q is in the benefit of the company or not. For the time being, the company is **only** interested to get this real time information for transportation of goods **from city x to city y**. (x and y are names of actual cities not generic notations)

The above mentioned scenario can be modelled using a **weighted directed graph** $G = (V, E)$; the vertices in this graph represent cities where $x - y$ **Logistics** has its offices and an edge (u, v) indicates that the company has its own transportation available from city u to city v, whereas $w(u, v)$ represents the cost of using company's own transport to carry goods from city u to city v.

Any new transportation offer at discounted rate from city p to city q can be modelled as insertion of a new *temporary* edge from vertex p to vertex q, with $w(p, q)$ being the price of the offer.

(Recall, 95% of the company's business is related to the transportation of goods **from** city x **to** city y. Therefore, for the time being, the company is **only** interested to get this real time information for transportation of goods **from city x to city y**.)

Note1: All the edge weights are positive.

Note2: You must determine whether or not this new offer will reduce the cost of transportation of goods from city x to city y.

Hint: Recall that the shortest path problem exhibits optimal substructure.

(a) Suppose you have already computed the weights of all-pairs-shortest distances and have the information in array D. Using this information, how would you decide in $\Theta(1)$ time whether or not a ***new*** transportation offer from city p to city q will lower the cost of the transportation of goods **from** city x **to** city y. [3 Marks]

(b) Write an algorithm that takes $O((V+E)\lg V)$ preprocessing time and then decides in O(1) whether or not an ***new*** transportation offer from city p to city q will lower the cost of the transportation of goods **from** city x **to** city y. [7 Marks]

[10 Marks]

Solution

(a) Suppose the newly added edge is from u to v with weight w(u,v), then

If(D[x,u] + w(u,v) + D[v,y]) < D[x,y]) then use the new edge.

(b)

1) Run Dijkstra by keep x as source. Save distances from x to any other vertex v in d1[v] $O((V+E)\lg V)$
2) Calculate reverse of the graph G^R in linear time. O(V+E)
3) Run Dijkstra by keeping y as source in G^R. Save distances from y to any other vertex v for G^R in d2[v] $O((V+E)\lg V)$

Suppose the newly added edge is from u to v with weight w(u,v), then

If(d1[u] + w(u,v) + d2[v]) < d1[y]) then use the new edge.