National University of Computer and Emerging Sciences, Lahore Campus

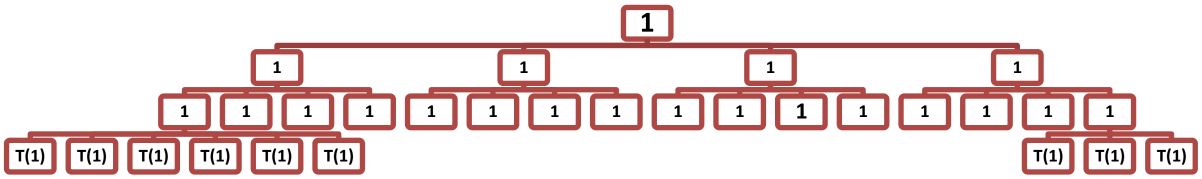
Course:	Design & Analysis of Algorithms	Course Code:	CS-2009
Program:	BS (Computer/Data Science)	Semester:	Fall 2023
Duration:	180 Minutes	Total Marks:	50
Paper Date:	3-Jan-24	Section:	ALL
Exam:	Final	Page(s):	10
Name		Roll Number	

Instruction/Notes: Solve it on this question paper. Work in the provided space and do not attach any sheets with the question paper.

Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total Marks
Marks	6	9	6	12	5	6	6	50
Obtained Marks								

Q1) Solve the following recurrences and find out asymptotic time complexity. Use recursion tree method. Show complete working.
[3+3 = 6]

a) $T(n) = 4T(n-1) + 1$



Work of each level increases as geometric progression. $1+4+4^2+\dots4^k$

Sum of geometric series common ration 4 >1 and k = n-1 total levels of tree are n

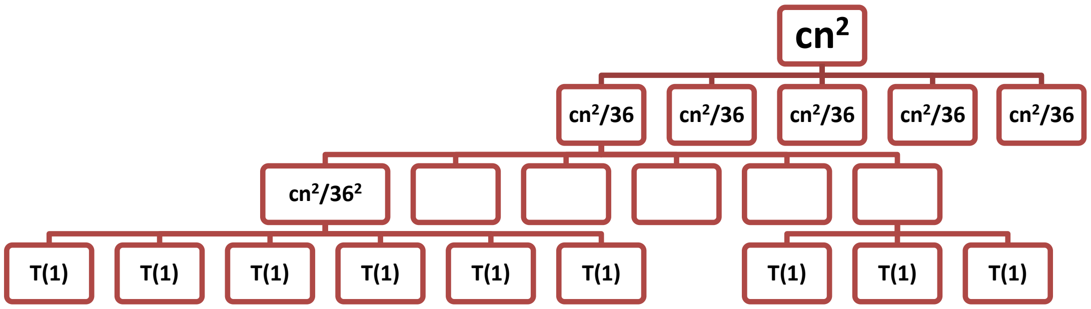
$(4^{k+1}- 1)/4-1 = 4^n/3$

Work at Last level leaf nodes = $\theta(4^n)$

$T(n) = 4^n/3 + \theta(4^n)$

$T(n) = O(4^n)$

b) $T(n) = 5T(n/6) + cn^2$



cn²
cn²/36 cn²/36 cn²/36 cn²/36 cn²/36
cn²/36²
T(1) T(1) T(1) T(1) T(1) T(1) T(1) T(1) T(1)

Work of each level decreases as geometric progression. $cn^2 + 5/36\ cn^2 + (5/36)^2\ cn^2 \dots (5/36)^k\ cn^2$

Sum of geometric series common ration 5/36 < 1 and k = $\log_6$

$1-(5/36)^{k+1}/1-(5/36) = (36/31)\ cn^2$

Work at Last level leaf nodes = $\theta(n^{\log_6 5}) = n^{0.9}$

$T(n) = (36/31)\ cn^2 + \theta(n^{\log_6 5})$

$T(n) = O(n^2)$

Q2)

a) Devise the recurrence relation (recursive equation) for the following recursive function for computing its time complexity. You do not need to solve it. **[2]**

```
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);
}
```

T(n) = T(n-1) + cn

b) You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. How to remove all duplicates from the array in time O(nlgn). Give the idea of your algorithm in 3-4 lines. **[3]**

Sort using merge sort or quick sort in O(nlgn) time, then do a linear scan in the sorted array and remove the duplicates by checking if consecutive numbers are same. Overall running time will be nlgn + n = O(nlgn)

c) Suppose you are given letter grades of students for the course Design and Analysis of Algorithms in the form of an array and you are required to answer the following query. How many students have earned grades from grade X to grade Y? Where X and Y can be any of the letter grades (A+, A, A-, B+, B, B-, C+, C, C-, D+, D, F). We need an algorithm that can answer such queries in O (1). You can preprocess the data to answer such queries. Give the idea of your algorithm in 3-4 lines. **[4]**

Preprocess: Modify count sort, do not sort but create an array B to store count of each letter grade. The size of array will be 12 (One entry for each letter grade). Array B can be created in O(n+k) time using the method similar to count sort.
Apply a for loop on array B to store cumulative counts of grades (B[i] = B[i] +B[i-1])
It will take O(n+k) time where k is range of umbers and in this problem range of data is only 12 so it will take O(n) time as k is very small.
Query: Array B can be used to answer the query in O(1) time.
B[map[Y]] – B[map[X]] //map is an array that maps a letter grade to a number in the range 0 to 11 for using it as array index.

Q3) Job Sequencing Problem – Loss Minimization

We are given **N** jobs numbered **1** to **N**. For each activity, let **T$_i$** denotes the number of days required to complete the job. For each day of delay before starting to work for job **i**, a loss of **L$_i$** is incurred. We are required to find a sequence to complete the jobs so that overall loss is minimized. We can only work on one job at a time. (Hint: Use Greedy algorithm). **[6]**

Examples

Input: L = {3, 1, 2, 4} and
T = {4, 1000, 2, 5}
Output: 3, 4, 1, 2
Explanation: *We should first complete job 3, then jobs 4, 1, 2 respectively.*

Input: L = {1, 2, 3, 5, 6}
T = {2, 4, 1, 3, 2}
Output: 3, 5, 4, 1, 2
Explanation: *We should complete jobs 3, 5, 4, 1 and then 2 in this order.*

https://www.geeksforgeeks.org/job-sequencing-problem-loss-minimization/

Greedy approach

Sort the jobs according to the ratio Li/Ti, in descending order. Sorting can be done using merge sort in O(nlogn)

L = {3, 1, 2, 4} and
T = {4, 1000, 2, 5}
L/T = {0.75, 0.001, 1, 0.8}
After Sorting
L/T = {1, 0.8, 0.75, 0.001}
Solution: 3, 4, 1, 2

L = {1, 2, 3, 5, 6}
T = {2, 4, 1, 3, 2}
L/T = {0.5, 0.5, 3, 1.67, 3}
After Sorting
L/T = {3, 3, 1.67, 0.5, 0.5}
Solution: 3, 5, 4, 1, 2

Time Complexity: O(nlogn)

Q4) When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by. A natural measure of the distance between two strings is the extent to which they can be aligned. An alignment is simply a way of writing the strings one above the other. For instance, here are two possible alignments of SNOWY and SUNNY where the "–" indicates a "gap"; any number of these can be placed in either string.

```
S  –  N  O  W  Y          –  S  N  O  W  –  Y
S  U  N  N  –  Y          S  U  N  –  –  N  Y
    Cost: 3                     Cost: 5
```

The cost of an alignment is the number of columns in which the letters differ. And the edit distance between two strings is the cost of their best possible alignment. Do you see that there is no better alignment of SNOWY and SUNNY than the one shown here with a cost of 3? **Edit distance** is so named because it can also be thought of as the ***minimum number of edits***—insertions, deletions, and replacements of characters—needed to transform the first string into the second. For instance, the alignment shown on the left corresponds to three edits: insert U, replace O → N, and remove W.

The goal is to find the minimum edit distance ED (m, n) between two strings $X_n = (x_1, x_2,...,x_n)$ and $Y_m = (y_1,y_2,..,y_m)$.

- Operation 1 (INSERT): Insert any character before or after any index of X
- Operation 2 (REMOVE): Remove a character of X
- Operation 3 (REPLACE): Replace a character at any index of X with some other character.

Note: All of the above operations are of equal cost of 1 unit.

Below is the recurrence relation for edit distance of strings X_n and Y_m where $X_n = (x_1,x_2,...,x_n)$ is a string of n characters and $Y_m = (y_1,y_2,..,y_m)$ is a string of m character:

$$ED\ (X_n, Y_m) = \begin{cases} m & if\ n = 0 \\ n & if\ m = 0 \\ ED\ (X_n - 1,\ \ Y_m - 1) & if\ x_n = \ y_m \\ Min\big(ED(X_{n-1}, Y_{m-1}), ED(X_{n-1}, Y_m), ED(X_n, Y_{m-1})\big) + 1 & if\ x_n \neq\ y_m \end{cases}$$

a) Compute the edit distance between two strings **X = POLYN** and **Y = EXPON**, by using the above DP solution. Fill the following table for finding minimum edit distance. **[3]**

	X	P	O	L	Y	N
Y	0	1	2	3	4	5
E	1	↘ **1**	→ **2**	→ **3**	→ **4**	→ **5**
X	2	↘ **2**	↘ **2**	→ **3**	→ **4**	→ **5**
P	3	↘ **2**	→ **3**	→ **3**	→ **4**	→ **5**
O	4	↓ **3**	↘ **2**	→ **3**	→ **4**	→ **5**
N	5	↓ **4**	↓ **3**	→ **3**	→ **4**	↘ **4**

b) Typists often make transposition errors exchanging neighboring characters, such as typing "setve" when you mean "steve." This requires two replace operations (edit distance=2) to convert setve to steve. If we incorporate a swap (swaps two adjacent characters) operation into our edit distance functions along with insert, remove and replace, then such neighboring transposition errors can be fixed at the cost of one operation. What would be the recurrence relation for this modified edit distance (that includes swap operation)? **[4]**

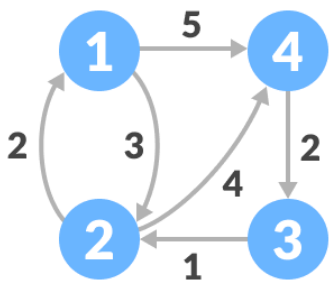
$$ED\,(X_n, Y_m) = \begin{cases} m & if\ n = 0 \\ n & if\ m = 0 \\ ED\,(X_n - 1,\ Y_m - 1) & if\ x_n = y_m \\ \boldsymbol{ED\,(X_n - 2,\ Y_m - 2) + 1\ \ if\ x_n \neq y_m\ and\ x_{n-1} = y_m\ and\ x_n = y_{m-1}} & \\ Min\big(ED(X_{n-1}, Y_{m-1}), ED(X_{n-1}, Y_m), ED(X_n, Y_{m-1})\big) + 1 & if\ x_n \neq y_m \end{cases}$$

c) Write down the pseudo code of a function that computed the edit distance of X_n and Y_m. Also write its time complexity **[5]**

```
for i = 0 to m
   ED [i][0] = i
for j = 1 to n
   ED [0][j] = j
for i = 1 to m
   for j = 1 to n
         if(X[i] = Y[j])
               diff = 0
         else
               diff = 1
               ED [i][j] = min{ ED [i - 1][j] + 1, //removal
                                ED [i][j - 1] + 1, //insertion
                                ED [i - 1][j - 1] + diff} // replacement
               if i > 1 and j > 1 and X[i] = Y[j-1] and X[i-1] = Y[j]
               ED[i][j] = minimum(ED[i][j], ED[i-2][j-2] + 1)  // Swap
return ED [m][n]
```

Time Complexity: O(nm)

Q5) Consider the graph given below and run ***Floyd Warshal*** algorithm to find all pairs shortest paths. Show the results at each step. For your ease matrices are provided. **[5]**



https://www.programiz.com/dsa/floyd-warshall-algorithm

D^0

	1	2	3	4
1	**0**	**3**	∞	**5**
2	**2**	**0**	∞	**4**
3	∞	**1**	**0**	∞
4	∞	∞	**2**	**0**

D^1

	1	2	3	4
1	**0**	**3**	∞	**5**
2	**2**	**0**	∞	**4**
3	∞	**1**	**0**	∞
4	∞	∞	**2**	**0**

D^2

	1	2	3	4
1	**0**	**3**	∞	**5**
2	**2**	**0**	∞	**4**
3	**3**	**1**	**0**	**5**
4	∞	∞	**2**	**0**

D^3

	1	2	3	4
1	**0**	**3**	∞	**5**
2	**2**	**0**	∞	**4**
3	**3**	**1**	**0**	**5**
4	**5**	**3**	**2**	**0**

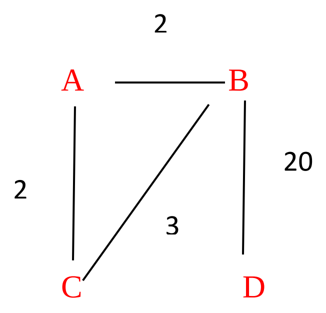
D^4

	1	2	3	4
1	**0**	**3**	**7**	**5**
2	**2**	**0**	**6**	**4**
3	**3**	**1**	**0**	**5**
4	**5**	**3**	**2**	**0**

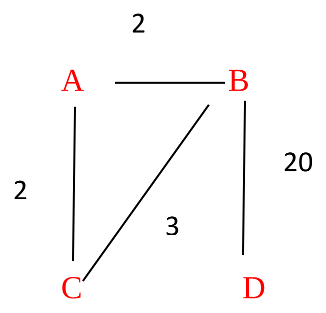
Q6)

a) In what scenario any MST algorithm will leave an edge with smaller weight edge and will pick a heavy weight edge. Explain the scenario with an example. **[3]**

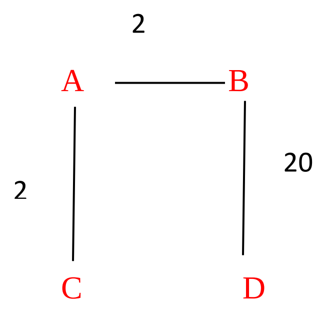
The smaller weight eight edge can create a cycle in exiting partial MST. In following example, edges AC, AB will be selected and then edge BD (cost = 20) will be selected instead of BC (cost = 3) as it creates a cycle in existing partial MST.



b) Let T be a minimum spanning tree of a graph G. Then for any two vertices u,v the path from u to v in T is a shortest path from u to v in G. Is this statement True or False? Justify your answer with an example. **[3]**
False

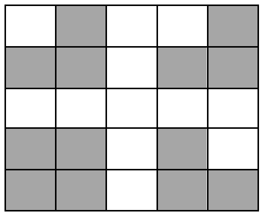


In following MST, shortest path from D to C is 24, where as in original graph shortest path from D to C is 23.



Q7) Nasa has captured an image of a distinct area of plant earth where either land or water is visible. They want to count the number of islands in the captured image. An IT expert in team NASA has converted that image into a N*N binary matrix such that 0 represents land and 1 represents water. Your task is to find the total number of islands in the given N*N matrix. For example, given the following matrix, your answer must be four as there are four isolated land regions. The shaded regions are water in this example.

0	1	0	0	1
1	1	0	1	1
0	0	0	0	0
1	1	0	1	0
1	1	0	1	1



Consider this problem as a graph problem where each cell represents vertex and adjacent cells represent adjacent vertices. An attribute "key" is associated with each vertex where 1 represents that vertex is land and 0 represents the vertex as water. **[6 Marks]**

a) Your task is to design an efficient algorithm to count the number of islands. (Hint: Connected Components). Write pseudo code of algorithm.

```
int DFS_countIsland(mat[n][n], n){

 visited [n][n] // Initialize visited vertices data
  for i=0 to n
   for j=0 to n
     visited[i][j] = 0
  island = 0
  for i=0 to n //count the connected land components that are not visited already
   for j=0 to n
    if mat[i][j] == 1 && visited[i][j] == 0 {
      island++
      DFS_visit(mat, visited, i, j, n)
     }
  return island
}
void DFS_visit (mat[n][n], visited[n][n], i, j, n) {
 visited[i][j] = 1
 for every u adjacent to v(i,j)
   //recursively call DFS for all neighbors that are land and not visited already
   if mat[ui][uj] == 1 && visited[ui][uj]==0
    DFS_visit (mat, visited, ui, uj, n)
}
```

b) Write down the time complexity of your algorithm**.**

$T(n) = O(V + E)$