Name Fazeel khalid

Roll # L19-1021

Section CS-4D

Question 1

a) Algorithm:

```
//decleration
bi[n] ← # of bits store in it.
ti[n] ← # of time duration for n stream stor
r ← Constant rate store.

Count = 0 //store # of valid schedule.

for i = 1 to n
        if (bi[i]/2 less or equal to r)
            count++.
            temp = bi[i] - bi[i]/2
        else if (temp less or equal to r)
            count++
        else if (bi[i] less or equal to r)
            count++

return
```

b) Time Complexity:-

As we need to see the upper bound or worst case analysisis. So here we are checking all array of bi one by one so the running time complexity of this algorithm is. $\Theta(n)$

c)

As greedy algorithm is one of the best technique to solve these Proble im Using this one for this Problem.

look at condition that present in for loops that are

if (bi[i]/2 <= r)

else if (temp <= r)

else if (bi[i] <= r)

these are Using to find valid stream. until we check all bits that we are sending on stream of 'n' number.

Question 2

As we know that greedy algorithm choosen the shortest path, With the help of this we can say that this is a greedy approach because we are selecting the last activity instead of selecting the first activity to finish. It is the shortest path to take, as it select the last activity to start.

Explain:=> Let assume a set of activities 'f' is given as a set an $A_1, A_2 \cdots A_n$ are the activities.

$$f = (A_1, A_2, \cdots\cdots A_n) \longrightarrow (i)$$

so

$A_1 = [f_1, d_1)$

$A_2 = [f_2, d_2)$

$\vdots$

$A_n = [f_n, d_n)$

or we can write it as

$A_i = [f_i, d_i)$

now we try to find the optimal solution,

activities→

For the solution crete a set f' with some elements that are activities like $A_1', A_2' ---- A_n'$

so

$$f' = (A_1', A_2' ---- A_n') \longrightarrow (ii)$$

$$A_1' = [f_1', d_1)$$
$$A_2' = [f_2', d_2)$$
$$| \quad\quad | \quad\quad |$$
$$| \quad\quad | \quad\quad |$$
$$| \quad\quad | \quad\quad |$$
$$A_i' = [f_i', d_i)$$

By examine the eqation i) and ii) we can say that iff the subset of

$$(A_1, A_2 ---- A_n) \subset f$$

and

$$(A_1', A_2' ---- A_n') \subset f'$$

and both of them are compatible

with eachother Then there is an optimal solution for f is mapped directly to an optimal solution for f'.

The answer of algorithm for f corresponds to the answer of the f' that's why we can say that this algorithm is optimal.

Question 3

There are two ways to solve this Problem.

(i) first algorithm will take $\Theta(n^2)$ time

(ii) best approach. time complexity is $\Theta(n\log n)$.

Discuss 1st approach⇒

First find a set of maximum **f1, f** Compatiable activites for the first hall. Then use again to find an **f2** set of maximum size **f - f1** compatible activities for the second hall.

So this algorithm will take $\theta(n^2)$ that's not a good approach.

Best approach:

Time complexity for the best approach is **$\theta(n \log n)$**

step 1 ⇒ we need to creat a list that consist of following data member

(i) time
(ii) type
(iii) Activity

this list will Consist of **2n** elements here n is a total # of lecture halls.

Optimal - Solution.

```
//declaration of variables.

Halls   H
list   L[2n]
array   temp[n]  // n # of halls.
Stack   S

for i=1 to 2n; increment by 1

        if (L[i].type == end)
            S. Push (L[i].activity)

        else
            if (!S.empty())
                temp1 = S.top()
                S.pop()
                temp[L[i].activity] = tem1
            else
                H++;
                temp[L[i].activity] = H;

for i= 1 to n   increment by 1
        Print(a[i]);

return
```

Question 4

// declaration of variable.

Graph G(n) // graph consist of following data
// member i) adjmatrix (ii) People
// (iii) vertexNum.

bool inVites [n];

queue rejects.

// creat relation ship amound person.

G = creatRelation(n)

// uninvite those how know less then 5 People.

for i = 1 to n ; incremented by 1
if (G. get Degree(i) < 5) {
rejects. push(i)
invites [i] = 0
else
invites [i] = 1

// count invited people.

for { i = 1 to n ; increment by 1

4

```
if (invite[i]== 1)
    invited ++;

return invited

graph CreatRelation(n)
  Graph G(n
  for i = 1 to n
        // get person
      cout<<G.getperson(i)
      while (input != -1)
          // get person #
          cin >> input
          G.adEdge(i, input-1)

~~return~~
return G.
```