Name : Muhammad Israib Akhtar
Section: ~~211-6294~~ BCS-4B
Rollno: 211-6294

Question 1

(a)

```
int minCut(string a)
{
    int n= a.length();
    int cut[n];
    bool palindrome[n][n];
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            palindrome[i][j]= false;

    for (int i=0; i<n; i++)
    {
        int MinCut= i;
        for (int j=0; j<=i; j++)
        {
            if(a[i]==a[j] && (i-j<2 || palidrome[j+1][i-1]))
            {
                palindrom[j][i]= true;
                minCut= min(minCut, j==0 ? 0 : (cut[j-1]+1));
            }
        }
        cut[i] = minCut;
    }
    return cut[n-1];
```

Scanned with CamScanner

Question 1

(b)

```
int maxRevenue(int M, int x[], int r[], int n, int t)
{
    int maxRev[M+1];
    for(int i=0; i<=M; i++)
        maxRev=0;
    int nextBB = 0;
    for(int i=1; i<=M; i++)
    {
        if (nextBB < n)
        {
            if( x[nextBB] != i )
                maxRev[i] = maxRev[i-1];
            else
            {
                if (i<=t)
                    maxRev[i] = max(maxRev[i-1], r[nextBB]);
                else
                    maxRev[i] = max( maxRev[i-t-1]+r[nextBB], maxRev[i-1]);
                nextBB;
            }
        }
        else
            maxRev[i] = maxRev[i-1];
    }
    return maxRev[m];
```

Time Complexity: O(M) where M is distance of total Highway

Auxiliary Space: O(M)

Scanned with CamScanner

Question 2

(a)

```
int n;
cin >> n;
int t[n];
for(int i=0; i<n; i++){
    cin >> t[i];
}
sort(t);
int waiting_time = 0;
for(int i=0; i<n; i++)
    waiting_time += t[i];
cout << waiting_time;
```

time Complexity : $O(n \log n)$

Auxiliary Space : $O(1)$

Explanation :

Swapping the order of the customers causes minimal total waiting time. Each swap reduce the total waiting time. Hence optimal solution becomes the greedy solution.

Question 2 Part 1

(b) Proposed Algorithm

Iterative

$n = s.length$

$A = \{a_n\}$

$K = n$

for $m = n-1$ down to 1

  if $f(m) \leq s[k]$

    $A = A \cup \{a_m\}$

    $K = m$

return A

Scanned with CamScanner

```
Recursive (s, f, k, n)
    m = k-1
    while m >= 1 and f[m] <= s[k]
        m = m-1
    if m >= 1
        return {a_m} U recursive(s, f, m, n)
    else return ∅
```

initial call

recursive (s, f, n+1, n).

Explanation

The above algorithm tries to find an optimal solution in each stage. This approach is a greedy algorithm. The solution produced by the approach that selects first activity to start can also be obtained by the proposed approach.

Therefore, the proposed approach also produces an optimal solution.

Part 2

approach of least duration

i	1	2	3
si	0	3	4
fi	4	5	7
duration	3	2	3

result : $\{a_2\}$

best result : $\{a_1, a_3\}$

hence not optimal

Scanned with CamScanner

activities with fewest overlaps

i	1	2	3	4	5	6	7	8	9	10	11
si	0	1	1	1	2	3	4	5	5	5	6
fi	2	3	3	3	4	5	6	7	7	7	8
no of overlapping activities	3	4	4	4	4	2	4	4	4	4	3

result : first selects a_6 then ~~one of~~ only two other activities, one from a_1, a_2, a_3, a_4
one from a_8, a_9, a_{10}, a_{11}

optimal result : $\{a_1, a_5, a_7, a_{11}\}$

compatible remaining activities with earliest start time

i	1	2	3	4	5	6	7	8	9	10	11	12
si	1	3	0	5	3	5	6	8	8	2	12	0
fi	4	5	6	7	8	9	10	11	12	13	14	15

$\{0, 15\}$ will be the first activity selected and there are no other compatible activities.

optimal result : $\{a_1, a_5, a_7, a_{11}\}$

~~Point 3~~
~~Point 3~~

Scanned with CamScanner

Part 3

```
Value_activity selector(s, f, v, n)
  val[n+1][n+1];
  act[n+1][n+1];
 for i = 0 to n
  val[i][i] = 0
  val[i][i+1] = 0
val[n+1][n+1] = 0
for l = 2 to n+1
   for i = 0 to n-l+1
       j = i + l
       val[i][j] = 0
       k = j - 1
       while f[i] < f[k]
          if(f[i] ≤ s[k] and f[k] ≤ s[j] and
                (val[i][k] + val[k][j] + v[k] > val[i][j])
             val[i][j] = val[i][k] + val[k][j] + v[k]
             act[i][j] = k
          k = k - 1
print val[0, n+1]
Print set contains
  Print activities(val, act, 0, n+1);
```

```
Print activities(val, act, i, j)
if (val[i][j] > 0
   k = act[i][j]
   Print k
   print activities(val, act, i, k)
   print activities(val, act, k, j)
```

The ~~procedu~~ algorithm runs in $O(n^3)$ time.

Scanned with CamScanner