Project Report I

→ Given the input array

A[1, ..., N]

create M grouping of A.

→ 1st group G[1] elements of array A

→ 2nd group G[2] elements of array A hence forth.

→ Last m group has G[m] elements

eg. if N = 8 M = 3

division G[1] = 2

G[2] = 3

G[3] = 3. → addition gives N nos.

divided into three groups

or else

G[1] = 1

G[2] = 5

G[3] = 2 → addition gives N nos.

can be other combinations also.

Now consider case in question.

A = | 3 | 9 | 7 | 8 | 2 | 6 | 5 | 10 | 1 | 7 | 6 | 4 |

Here

N = 12 , M = 3.

G can be {4|4|4}

Now

→ Store the m groups in arrays B and G such that

$$\sum_{i=1}^{M} G[i] = n \qquad \sum_{i=1}^{m} B[i] = \sum_{i=1}^{N} A[i]$$

B[j] has sum of elements in j^{th} group of A.

one way

so $G_1 = [4,4,4]$

3	9	7	8	2	6	5	10	1	7	6	4

B[1] = 3+9+7+8 = 27
B[2] = 2+6+5+10 = 23
B[3] = 1+7+6+4 = 18.
B_1 = [27, 23, 18] = Bmin = 18.

other way

$G_{12} = [3,4,5]$

3	9	7	8	2	6	5	10	1	7	6	4

$B_1 = [3+9+7] = 19$
$B_2 = 8+2+6+5 = 21$
$B_3 = 10+1+7+6+4 = 28$

Barray [19, 21, 28] = Bmin = 19

Here [3,4,5] has the best grouping so Bmin = 19.

grouping G_1 gives max Bmin value

This was found on the basis of max Bmin value.

now,

$$B[i] = \sum_{j=K+1}^{K+G[i]} \quad \text{where} \quad 12 = \sum_{P=1}^{i-1} G_1[P]$$

We can use dynamic programming to solve this problem:

optimal substructure Property is:

Let $G_1[1, \ldots m]$ be optimal solution for maximum minimum grouping (A, N, M)

so the optimal solution to Max-Min grouping (A, N, M) is $G_1[1, \ldots, M] = [n_1, n_2, \ldots, n_{M-1}, n_M]$. Then

$G_1[1, M-1]$ is the optimal solution to the subproblem Max-min-grouping $(A, N-n_M, M-1)$

Assume $C[i]$ denotes max of (min element of B across all the wages in groups) there are i groups array: $A[1 \ldots i]$

using above property we get -

$G_{opt} = B_{min}^{opt} = \max\{B_{min}\}$

$c[i,j]$ = the min element of array B that described by G_{opt}.

so B_{min} goes under $c[i,j]$ to help to find out the best grouping fit.

$$\therefore c[i,j] = \max_{j-1 \le k \le i} \min\{c[k, j-1], \sum_{m=k+1}^{i} A[m]\}$$

$A[i, \ldots k]$ ← into $j-1$ groups.

Put $A[k+1, \ldots i]$ into 1 group i.e the j^{th} group]

Also Range of i & j is defined as

$\boxed{1 \le i \le N}$

$1 \le j \le i$

$\boxed{1 \le j \le M}$

Taking $c[1,1] = \sum_{m=1}^{i} A[m] = 3$ as it a first row.

Also a pointer type of matrix is taken that has information k which leads to max value. This helps in tracking

and generating optimal grouping G_1.

$$P[i,j] = \underset{j-1 \le k < i}{\arg\max} \min\left\{ C[k, j-1], \sum_{m=k+1}^{i} A[m] \right\}$$

$$C[3,1] = \min\left\{ C[1,0], \sum_{m=2}^{i} A[m] \right\}$$
$$C[2,0], \sum_{m=3}^{i} A[m]$$

$1-1 \le k < 3$

$0 \le k < 3$ $= 19$.

$$C[2,2] = \max_{1 \le k < 2} \min\left\{ C[1,1], \sum_{u=2}^{2} A[2] \right\}$$

③ ⑨

where max $k = 1$
and min C is taken $= 3$.

$$C[3,2] = \max_{1 \le k < 3} \min\{ C[k,1], A[k+1] + \ldots A[3] \}$$

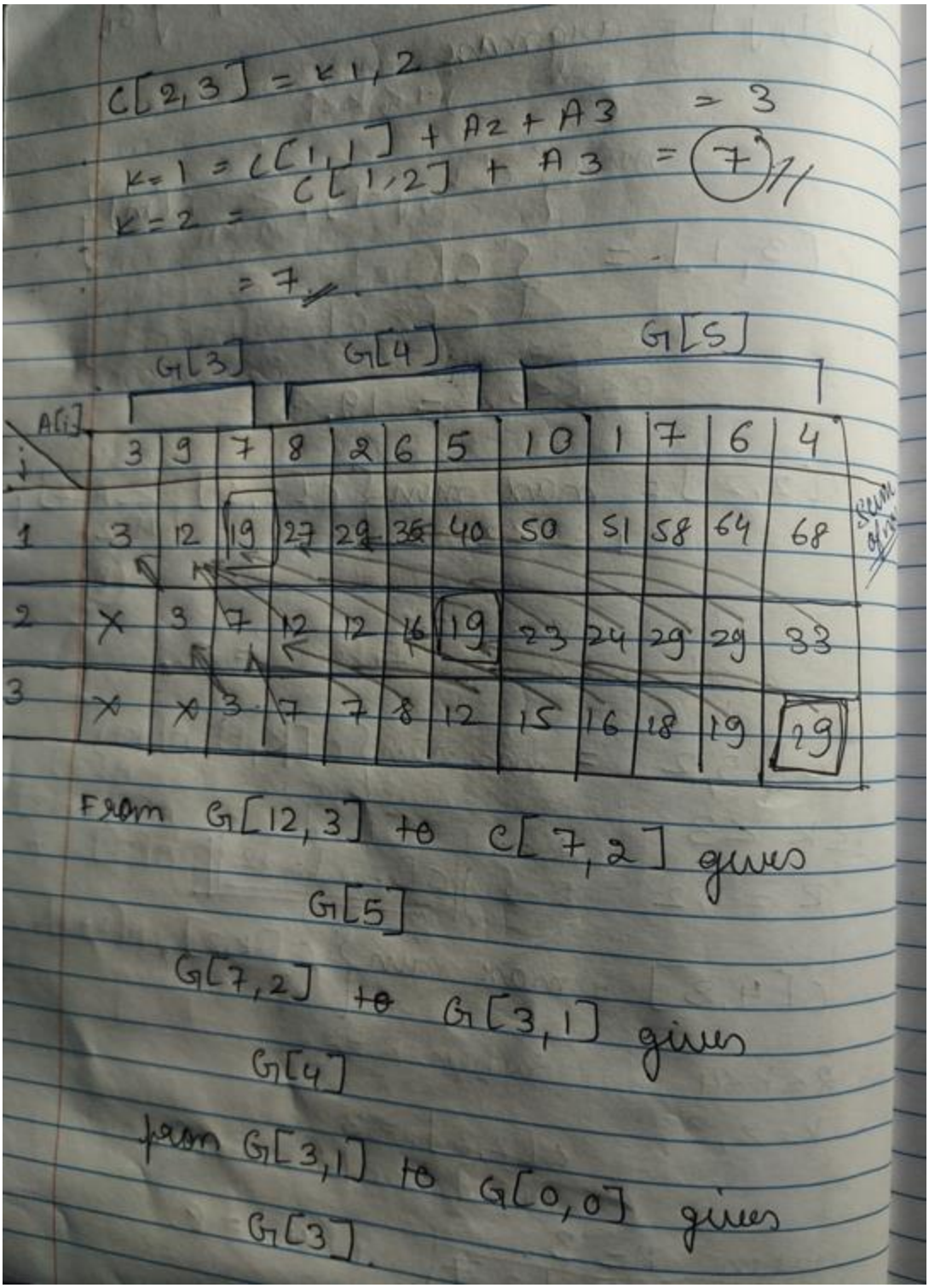
$k = 1, 2$ $\quad k=1 \quad C[1,1] = 3 \quad A[2] + A[3] = 16$
$\quad k=2 \quad C[2,1] = 12 \quad A[3] = 7$

$$C[4,3] = \max \min\{ C[k,2], A[k+1] + \ldots 4 \}$$

$2 \le k < 4$ $\quad$ when $k = 2$
$k = 2, 3$

$C[2,2] = 3$
$A[3] + A[4] = 15$.

when $k = 3$.
$C[3,2] = 7$ $\quad A[4] = 8$.
Hence $= 7$ min //.


C[2,3] = k=1,2
k=1 = C[1,1] + A2 + A3 = 3
k=2 = C[1,2] + A3 = 7
= 7
G[3]
G[4]
G[5]
A[i]
i
3 9 7 8 2 6 5 10 1 7 6 4
1 3 12 19 27 29 35 40 50 51 58 64 68
2 X 3 7 12 12 16 19 23 24 29 29 33
3 X X 3 7 7 8 12 15 16 18 19 19
From G[12,3] to C[7,2] gives G[5]
G[7,2] to G[3,1] gives G[4]
from G[3,1] to G[0,0] gives G[3].

Pseudo code:

1) Enter the size of the array
2) Enter the m groups that it needs to be divided in
3) Input the array in the form of vector.
4) Calling of the Max-Min grouping function
5) Initialize C(carray) vector in the form of 2-D matrix with size n*m with 0's
6) C(carray) stores the max value obtained by considering max k obtained from the max (Bminarray) for every row i and j.
7) Initialize (Bminarray) vector such that it stores k value gives max(min)
8) 1st row of C(carray) table stores the sum of the elements of A(mainarray) having the index k+1 to i
9) def Max_Min_Grouping(mainarray,n,m)
10) first carray[0][0] =mainarray[0]
11) for i in 1 to N-1
12) carray[0][i] =mainarray[i]+carray[0] [i-1]
13) filling of the ctable as per dynamic programming recursive formula
14) for j in1 to m-1:
15) for i in j to n-1:
16) range is a variable that is defined as maximum value
17) range=-1000
18) taking max argument parameter as range1=-1
19) for k in j-1 to i-1:
20) presentvalue=min(carray[j-1] [k], sum(mainarray[k+1…..i]
21) if presentvalue>range
22) range=presentvalue
23) range1=k
24) carray[j][i] =range
25) bminarray[j][i] =range1
26) breadth=n-1
27) a=m
28) while(a>0)
29) a=a-1
30) grouping[a]=breadth-bminarray[a][breadth]
31) breadth=bminarray[a][breadth]
32) return grouping // returning the optimal value of the group

Analysis of time complexity asymptotically :

Time complexity Analysis of running time is asymptotically. is given by tetha O.

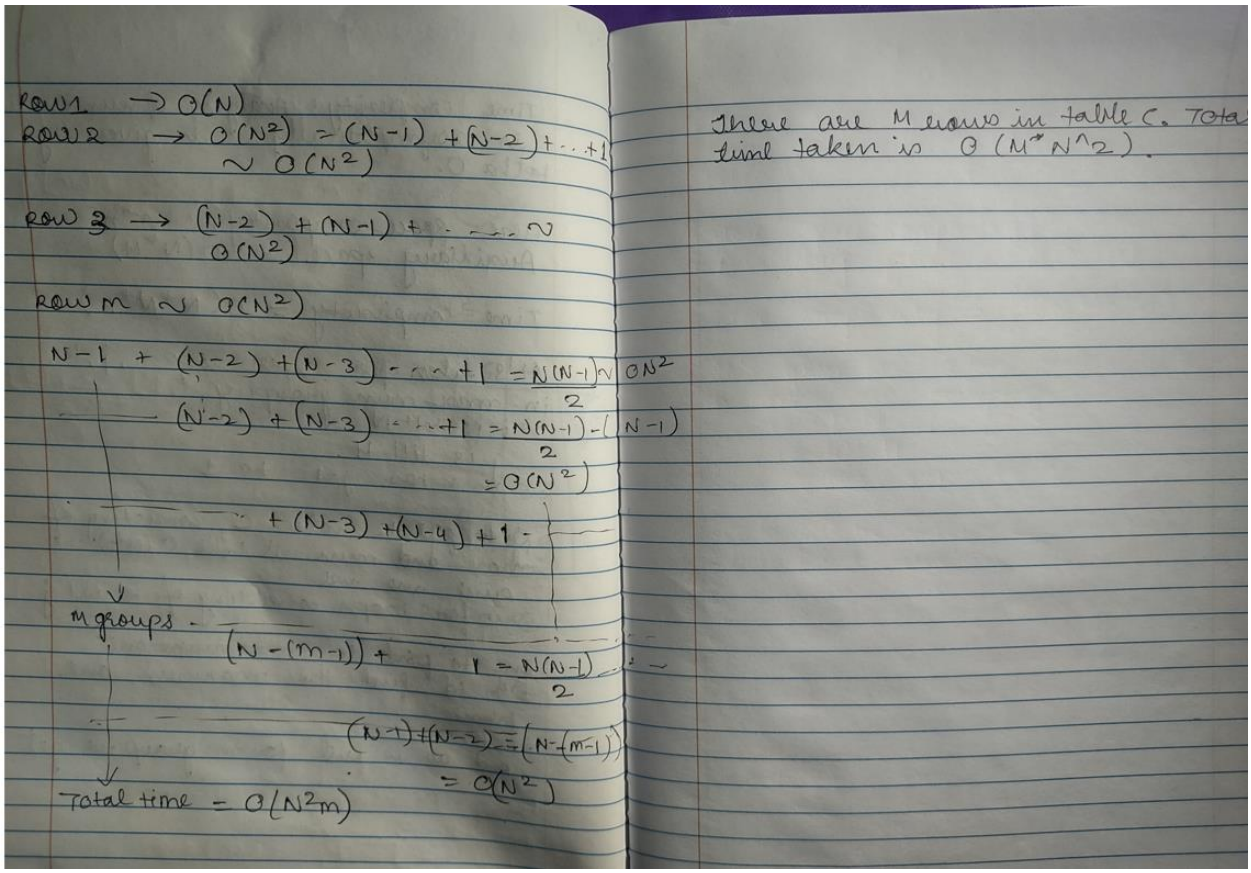
Here space complexity is n*m
Auxiliary space = O(N*M)

Time complexity = O(N*N*M)

It's taken for 3 for loop i, j, k.
in max-min grouping.
i is till N
j is till M
k has j-1 to 1

So to fill the matrix C, consisting of max and min it take $O(N^2)$
since we use
2 for loops consisting of k and j
one to find the minimum one
2nd to find the maximum one.
hence.

And as we find for m group we get



Results:

```
Enter the size of the array
12
Enter the number of groups it needs to be divided in
3
Enter the array  :
3
9
7
8
2
6
5
10
1
7
6
4

 3 12 19 27 29 35 40 50 51 58 64 68
0 3 7 12 12 16 19 23 24 29 29 33
0 0 3 7 7 8 12 15 16 18 19 19


grouping size is 3 of the number of the elements entered in array in form a vector of size 12
Optimal grouping is :
3 4 5
```

```
Enter the size of the array
10
Enter the number of groups it needs to be divided in
4
Enter the array  :
1
1
1
1
1
1
1
1
1
1

 1 2 3 4 5 6 7 8 9 10
0 1 1 2 2 3 3 4 4 5
0 0 1 1 1 2 2 2 3 3
0 0 0 1 1 1 1 2 2 2


grouping size is 4 of the number of the elements entered in array in form a vector of size 10
Optimal grouping is :
2 2 2 4

Enter the size of the array
12
Enter the number of groups it needs to be divided in
3
Enter the array  :
4
6
7
1
10
5
6
2
8
7
9
3

 4 10 17 18 28 33 39 41 49 56 65 68
0 4 7 8 11 16 18 18 21 28 32 33
0 0 4 4 8 8 11 11 16 17 18 19


grouping size is 3 of the number of the elements entered in array in form a vector of size 12
Optimal grouping is :
5 4 3
```

```
Enter the number of groups it needs to be divided in
4
Enter the array  :
7
4
9
3
6
1
5

 7 11 20 23 29 30 35
0 4 9 11 11 11 15
0 0 4 4 9 9 11
0 0 0 3 4 4 6


grouping size is 4 of the number of the elements entered in array in form a vector of size 7
Optimal grouping is :
2 1 2 2
```

Source code :

```
#include <iostream>

#include <stdio.h>

#include <vector>

#include <assert.h>

using namespace std;


void printtable(vector<vector<int>> const &f)

{

 //printing the value of 2d matrix in form of a vector

 for (vector<int>breadth : f) {

  for (int figure: breadth) {

   cout << figure << " ";

  }

  cout << '\n';

 }

}




int addition(vector<int> mainarray,int a,int b)
```

```
{
  // finding the sum of elements of main array
  int s=0;
  for (int i=a;i<=b;i++)
  {
    s=s+mainarray[i];
  }
  return s;
}



vector<int> Max_min_grouping(vector<int>mainarray ,int n,int m)
//Taking mainarray as A
{
  // Taking carray as C to store the 2d vector of the Bmin
  vector<vector<int>> carray(m, vector<int>(n, 0));
  // Taking a bminarray to store the k value obtained from max and min
  vector<vector<int>> bminarray(m, vector<int>(n, -1));
  carray[0][0]=mainarray[0]; // first row of the table
  for (int i=1;i<n;i++)
  {
    carray[0][i]=mainarray[i]+carray[0][i-1];
  }
  // dynamic programming
  for (int j=1;j<m;j++)
  {
    for (int i=j;i<n;i++)
    {
      int range=-1000;
```

```
    int range1=-1;
    for (int k=j-1;k<i;k++)
    {
      int presentval=min(carray[j-1][k],addition(mainarray,k+1,i));
      if (presentval>range){
        range=presentval;
        range1=k;
      }
    }
    carray[j][i]=range;
    bminarray[j][i]=range1;
  }
 }
 vector<int> grouping(m, 0);
 int length=n-1;
 int a=m;
 while(a>0){
  a=a-1;
  grouping[a]=length-bminarray[a][length];
  // using G as grouping array to store the number of value b obtained
  length=bminarray[a][length];
 }
 printtable(carray);
 cout <<endl<<endl;
 return (grouping); //returning optimal group
}


int main(int argc, char const *argv[])
```

```
{
  vector<int> x;// taking x as the array A
  int enter,n,m;
  cout <<"Enter the size of the array";
  cout<<endl;
  cin >> n;
  cout <<"Enter the number of groups it needs to be divided in ";
  cout<<endl;
  cin >> m;
  cout<<"Enter the array  : \n";
  for (int i=0;i<n;i++){
    cin>>enter;
    x.push_back(enter);
  }
  cout<<" \n ";
  assert (n<=x.size());
 // assert is macro to prevent the debug and error inside the function
  vector<int> grouping=Max_min_grouping(x,n,m); // taking G as the grouping array
  cout <<"grouping size is "<<m<<" of the number of the elements entered in array in form a vector of
size "<< n<<endl;
  cout <<"Optimal grouping is : "<<endl;
  for (int i=0;i<grouping.size();i++){
    cout<<grouping[i]<<" ";
  }
  cout<<"\n";
}
```