Project 1

→ given → array $A[1 \ldots N]$

↳ create m groupings of A.

→ store the m groups in arrays B and G, such that →

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

→ $G[i]$ has # of first 'x' numbers of array A in group 1.

2nd group has $G[2]$ elements. Last group has $G[m]$ elements.

→ $B[i]$ has sum of elements in ith group of A.

→ Let the function call be →

$MMG(A, N, m)$

To find → $G[1 \ldots m]$. OR the optimal G that gives ~~B~~ largest B_{min}.

Example. $A = 3, 9, 7, 8, 2, 6, 5, 10, 1, 7, 6, 4$. $m = 3$, $A = 12$

$G_1 = [3 | 4 | 5]$ $\quad B_1 = [19 | 21 | 28]$ $\rightarrow B_{min} = 19$

$G_2 = [4 | 4 | 4]$ $\quad B_2 = [27 | 23 | 18]$ $\rightarrow B_{min} = 18$

$G_3 = [5 | 4 | 3]$ $\quad B_3 = [29 | 22 | 17]$ $\rightarrow B_{min} = 17$

$\Rightarrow G_1$ is best grouping $\because$ it has max (B_{min})

$\rightarrow$ Find grouping G that gives max B_{min} value.

Further ;

$$B[i] = \sum_{j=K+1}^{K+G(i)} \quad \text{where} \quad K = \sum_{t=1}^{i-1} G(t)$$

Brute force → for each index i of A[1...n], we have a choice to partition (~~after~~ create a group) or not. ⇒ 2^n groups are possible.

⇒ 2^n $G(i)$ groups in worst case.

⇒ for each $G(i)$; $O(n)$ time to create $B(i)$

⇒ $O(n \cdot 2^n)$ time to ~~g~~create all $B(i)$.

⇒ a linear scan through $B(i)$'s will ~~g~~ and finding max − min ⇒ $O(n \cdot 2^n)$

⇒ Brute force time → $O(n \cdot 2^n)$

→ we can use Dynamic programming to solve this problem.

→ optimal substructure property is →

Let $Gr[1 \ldots m]$ be optimal solution for $mmGr(A, N, m)$.

Then, $Gr[1 \ldots m-1]$ is the optimal solution for $mmGr(A, N-n_m, m-1)$

where n_m is the number of elements in mth group.

→ Let C_{ji} denote the ~~answer~~ max of (min element of B across all groups).

assuming there are j groups and array $A[1 \ldots i]$

Subproblem → $mmGr(A, i, j)$

→ As in bottom up DP, we can create a table with entries for $j = 1$ to m and $i = 1$ to N.

$C_{N,m}$ will hold the answer to max - (min element of B) ← Bmin

we can also store a parallel matrix D which holds the values of 'k' → next pos.

This information will allow us to back track and generate groups Gr.

Further, for each entry in matrix C, store the information which 'k' lead to max value; in matrix D.

$$\text{Dji} = \underset{j-1 \le k < i}{\operatorname{argmax}} \left[\min \left(C(j-1, k), \sum_{m=k+1}^{i} A(m) \right) \right]$$

This matrix will enable us to back track & generate optimal grouping of G.

→ elements of A (i) Matrix C

j \ i	1	2	3	4	5	6	7	8	9	10	11	12
A[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	NaN	K=1 3	K=2 ~~7~~	K=2 12	K=2 12	K=3 16	K=3 19	K=4 23	K=4 24	K=5 29	K=5 29	K=6 33
3	NaN	NaN	3	7	7	8	12	15	16	18	19	19
4	NaN	NaN	NaN	3	3	7	7	10	11	12	14	16
5	NaN	NaN	NaN	NaN	2	3	5	7	7	8	11	11

(j) number of groups

$$C_{ji} = \max_{j-1 \le K < i} \left[\min \left(C\left(j-1, K\right), \sum_{m=K+1}^{i} A(m) \right) \right] \quad * \text{ Recursion}$$

($C_{j-1,K}$)

if $j = 1$; $C_{ji} = \sum_{p=1}^{i} A[p]$ → Base case

if $j \ge i$; C_{ji} = NaN → can't make j groups with i elements if $j > i$

$C_{22} = \max_{1 \le K < 2} \min \left[C(i-1, K), \sum_{m=K+1}^{i} A(m) \right]$

$K=1$; $\min (C(1,1), A(2)) = 3$; $B_{22} = 1$ (K value)

$C_{23} \Rightarrow K = 1, 2$

$K=1$; $\min (C(1,1), A(2) + A(3)) = 3$;

$K=2$; $\min (C(1,2), A(3)) = 7$; $B_{23} = 2$

$C_{24} \Rightarrow K = 1, 2, 3$

$K=1$ $\min (C_{11}, A_2 + A_3 + A_4) = 3$

$K=2$ $\min (C_{12}, A_3 + A_4) = 12$ $B_{24} = 2$

$K=3$ $\min (C_{13}, A_4) = 8$

$C_{25} \Rightarrow K = 1, 2, 3, 4$

$K=1 \rightarrow \min (\ ) = 3$ $\qquad B_{25} = 2$

$K=2 \rightarrow \min (C_{12}, \Sigma(A_3 \cdots A_5)) = 2$

$K=3 \rightarrow \min (C_{13}, A_4 + A_5) = 10$

$K=4 \rightarrow \min (C_{14}, A_5) = 2$

$C_{26} \Rightarrow K = 1, 2, 3, 4, 5$

$K=1 \rightarrow \min (\ ) = 3$

$K=2 \rightarrow \min (\ ) = 12$ $\qquad B_{26} = 3$

$K=3 \rightarrow \min (C_{13}, A_4 + A_5 + A_6) = 16$

$K=4 \rightarrow \min (C_{14}, 2) = 2$

$K=5 \rightarrow \min (C_{15}, A_6) = 6$

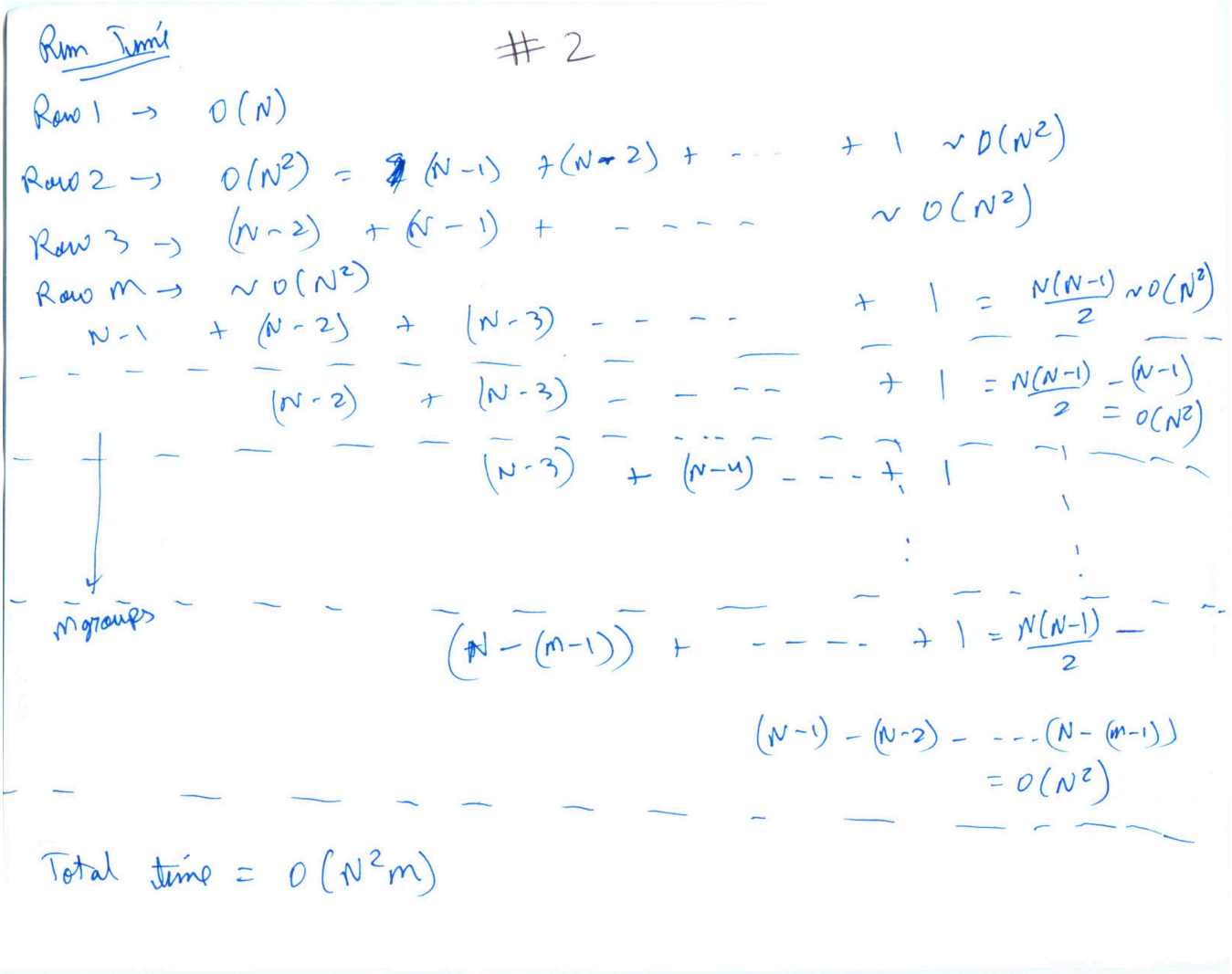
Pseduocode

```
// array indexing from 0 for all arrays
// initialize G as a 1D matrix of size M with 0's
// initialize C as a 2D matrix of size NxM with 0's
// C stores max(Bmin) for every row, column (i,j)
// initialize B as a 2D matrix of size NxM with -1's
// B stores k values that give max(Bmin)
// sum(A[k+1.....i]) is sum of elements of A from index k+1 to i




def Max_Min_Grouping(self,A,N,M):
    C[0][0]=A[0]
    for i in 1 to N-1: // fill 1st row of C
        C[0][i]=A[i]+C[0][i-1]
    // fill rest of the C table based on recursion formula. Also update B
    for j in 1 to M-1:
        for i in j to N-1:
            best_val=-infinity
            arg_max=-1
            for k in j-1 to i-1:
                current_val=min(C[j-1][k],sum(A[k+1.....i])
                if current_val>best_val:
                    best_val=current_val
                    arg_max=k
            C[j][i]=best_val
            B[j][i]=arg_max
    // Use the values in B to get optimal grouping G
    column=N-1
    m=M
    while(m>0):
        m=m-1
        G[m]=column-B[m][column]
        column=B[m][column]
    return G
```

Run time



- To fill every row of table C, it takes O(N^2) time (as elaborated above)
- There are M rows in table C. Total time taken is O(M*N^2)

Grouping results of several input examples

input vector is : 3 9 7 8 2 6 5 10 1 7 6 4
M is 3 and N is 12
optimal grouping is : 3 4 5

---

input vector is : 3 9 7 8 2 6 5 10 1 7 6 4
M is 4 and N is 12

optimal grouping is : 3 3 3 3

---

input vector is : 3 9 7 8 2 6 5 10 1 7 6 4
M is 5 and N is 12
optimal grouping is : 2 2 3 2 3

---

input vector is : 3 9 7 8 2 6 5 10 1 7 6 4
M is 6 and N is 12
optimal grouping is : 2 2 3 1 2 2

---

input vector is : 7 9 2 5 8 5 3 21 3 6 8
M is 3 and N is 11
optimal grouping is : 3 3 5

---

input vector is : 7 9 2 5 8 5 3 21 3 6 8
M is 5 and N is 11
optimal grouping is : 2 3 2 1 3

---

input vector is : 7 9 2 5 8 5 3 21 3 6 8
M is 7 and N is 11
optimal grouping is : 1 1 2 1 2 1 3

---

input vector is : 7 9 2 5 8 5 3 21 3 6 8
M is 4 and N is 8
optimal grouping is : 2 3 2 1

input vector is : 7 9 2 5 8 5 3 21 3 6 8
M is 3 and N is 8
optimal grouping is : 3 3 2

Code

```
// Run command-
// g++ -std=c++11 sol1.cpp -o solution.out && ./solution.out

#include <vector>
#include <stdio.h>
#include <assert.h>
#include <iostream>
using namespace std;
void print_1D_vector(vector<int> v){
  //helper function to print 1D vector
  for (int i=0;i<v.size();i++){
    cout<<v[i]<<" ";
  }
  cout<<endl;
}

void print_2D_vector(vector<vector<int>> const &v) {
  //helper function to print 2D vector
  for (vector<int> row: v) {
    for (int val: row) {
      cout << val << " ";
    }
    cout << '\n';
  }
}


int sum(vector<int> A,int start,int stop){
  //helper function to sum elements of array A[start....stop]
  int sum=0;
  for (int i=start;i<=stop;i++){
    sum=sum+A[i];
  }
  return sum;
}


vector<int> Max_min_grouping(vector<int> A,int N,int M){
  // 2D vector C to store max(Bmin) for every row, coloumn (i,j)
  vector<vector<int>> C(M, vector<int>(N, 0));
  // 2D vector B to store k values that gave max(Bmin). Will help to backtrack and get
  // the optimal G grouping
  vector<vector<int>> B(M, vector<int>(N, -1));
  // fill 1st row of C
  C[0][0]=A[0];
  for (int i=1;i<N;i++){
```

```
    C[0][i]=A[i]+C[0][i-1];
  }
  // fill rest of the C table based on recursion formula. Also update B with appropriate k values
  for (int j=1;j<M;j++){
    for (int i=j;i<N;i++){
      int best_val=-1000;
      int arg_max=-1;
      for (int k=j-1;k<i;k++){
        int current_val=min(C[j-1][k],sum(A,k+1,i));
        if (current_val>best_val){
          best_val=current_val;
          arg_max=k;
        }
      }
      C[j][i]=best_val;
      B[j][i]=arg_max;
    }
  }
  // Initialize 1D vector G. This shall be final optimal grouping
  vector<int> G(M, 0);
  // Use the values in B to get optimal grouping G
  int column=N-1;
  int m=M;
  while(m>0){
    m=m-1;
    G[m]=column-B[m][column];
    column=B[m][column];
  }
  // To further test, uncomment the below lines and check the values of matrix C and B
  //print_2D_vector(C);
  //cout <<endl<<endl;
  //print_2D_vector(B);
  //cout <<endl<<endl;
  return (G);
}

void run_MMG(vector<int> v,int N,int M){
  assert (N<=v.size());
  // helper function to call the function Max_min_grouping, print input parameters and final result
  cout <<"Input array is : ";
  print_1D_vector(v);
  vector<int> G=Max_min_grouping(v,N,M);
  cout <<"M is "<<M<<" and N is "<< N<<endl;
  cout <<"Optimal grouping is : ";
  print_1D_vector(G);
  cout<<endl<<endl;
```

```
}

void pre_defined_test_cases(){
  vector<int> v;
  v={3,9,7,8,2,6,5,10,1,7,6,4};
  run_MMG(v,12,3);

  v={3,9,7,8,2,6,5,10,1,7,6,4};
  run_MMG(v,12,4);


  v={3,9,7,8,2,6,5,10,1,7,6,4};
  run_MMG(v,12,5);

  v={3,9,7,8,2,6,5,10,1,7,6,4};
  run_MMG(v,12,6);


  v={7,9,2,5,8,5,3,21,3,6,8};
  run_MMG(v,11,3);


  v={7,9,2,5,8,5,3,21,3,6,8};
  run_MMG(v,11,5);


  v={7,9,2,5,8,5,3,21,3,6,8};
  run_MMG(v,11,7);

  v={7,9,2,5,8,5,3,21,3,6,8};
  run_MMG(v,8,4);

  v={7,9,2,5,8,5,3,21,3,6,8};
  run_MMG(v,8,3);
}


int main(int argc, char const *argv[]) {
  // uncomment below line to run some predefined test cases
  // pre_defined_test_cases();
  vector<int> v;
  int input;
  int M;
  int N;
  cout <<"Enter N (size of array to be considered) \n";
  cin >> N;
```

```
  cout <<"Enter M (number of groups) \n";
  cin >> M;
  cout<<"Enter array A : \n";
  for (int i=0;i<N;i++){
    cin>>input;
    v.push_back(input);
  }

  run_MMG(v,N,M);

  return 0;
}
```