Project 1. Problem Description: you are given an input array A[1, …, N]. A grouping of the array A is described by an array G[1, …, M], where the array A is partitioned into M groups, the 1st group consists of the first G[1] elements of array A, the 2nd group consists of the next G[2] elements, and so forth. Define array B[1, …, M] such that B[j] is the summation of the elements in the j-th group of array A. Use a dynamic programming algorithm to find a grouping of array A with M groups such that we maximize the minimum elements of array B.

Part 1. Pseudo codes of your dynamic programming algorithm:

```
Max_Min_Grouping(A, N, M){
create dp
for i ← 0 to n do
  dp[i][1] ← prefix[i] of A
for j ← 1 to N do
  for k ← 1 to M do
    for r ← k - 1 to j do
      dp[j][k] ← max(dp[j][k], min(dp[r][k - 1], s[j] - s[r]))
grouping g ← backtraking(dp)
return g
}
```

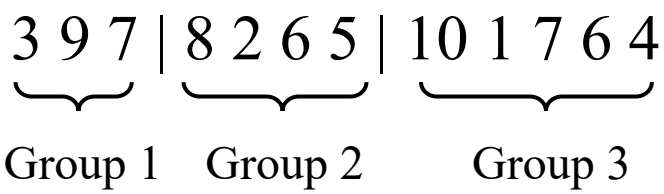
Part 2. Analysis of the running time asymptotically:

The maximum of the minimum value of the respective sum of the prefix [0, i] divided into k segments is dp[j][k], and the maximum of the minimum value of the respective sum of the prefix [0, j] (j < i) divided into k - 1 segments is dp[j][k - 1]. Every parition relies on the previous partition using k to loop 1 parts to M parts. Enumerating all possible k will give the maximum value. Thus we can get:

Time Complexity: $\Theta(N^2M)$

Part 3. Grouping results of several input examples including the one that A = {3,9,7,8,2,6,5,10,1,7,6,4} and M = 3:

Grouping resluts:



Max_Min = 19.

Part 4. Source codes:

```
#include<bits/stdc++.h>
using namespace std;

const int N = 5000;
const int M = 5000;
int dp[N][M];

vector<vector<int>> Max_Min_Grouping(vector<int> a, int n, int m) {
    vector<int> s, u;
    s.push_back(0);
    u.push_back(a[0]);
    for (int i = 0; i < n; i++) {
        s.push_back(s[i] + a[i]);
        u.push_back(min(u[i], a[i]));
    }
    for (int i = 1; i <= n; i++)
        dp[i][1] = s[i];
    for (int i = 1; i <= n; i++) {
        for (int j = 2; j <= m; j++) {
            if (i < j) break;
            if (i == j) dp[i][j] = u[i];
            else {
                for (int k = j - 1; k < i; k++) {
                    int tu = min(dp[k][j - 1], s[i] - s[k]);
                    dp[i][j] = max(dp[i][j], tu);
                }
            }
        }
    }
    int jq = dp[n][m]; // the Max_Min final result.

    vector<vector<int>>G;
```

```
    int qj = 0;
    vector<int>lr;
    for (int i = 0; i < n; ++i) {
        qj += a[i];
        lr.push_back(a[i]);
        if (qj >= jq) {
            G.push_back(lr);
            lr.clear();
            qj = 0;
            if (G.size() == m) {
                i++;
                for (; i < n; ++i)
                    qj += a[i], lr.push_back(a[i]);
                break;
            }
        }
    }
    if (qj) { // Backtracking.
        if (G.size() < m)
            G.push_back(lr);
        else
            G[G.size() - 1] .insert(G[G.size() - 1].end(), lr.begin(),
lr.end());
    }
    return G;
}


int main() {

    int n, m;
    cin >> n; // Input the size of array A.
    vector<int>r;
    vector<vector<int>> q;
    for (int i = 0; i < n; i++) { // Input A[n] elements.
        int x;
        cin >> x;
        r.push_back(x);
    }
    cin >> m; // Input the grouping number m.
    q = Max_Min_Grouping(r, n, m);

    for (auto i : q) { // Print the grouping result.
        for (auto j : i){4
            cout << j << " ";
```

```
        }
        cout << endl; // Each group is a line of the output.
    }
    return 0;
}
```