1. Algorithm Design: Max-Min Grouping

We define the recurrence for the Dynamic Programming approach. Let $C[i][j]$ be the maximum possible value of the minimal group sum when partitioning the first i elements into j groups.

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

Algorithm 1 Max-Min Grouping DP

```
function MaxMinGrouping(A, N, M)
    P[0] ← 0
    for i ← 1 to N do                                   ▷ Prefix Sums
        P[i] ← P[i − 1] + A[i]
    end for
    Initialize C[0 . . . N][0 . . . M] ← −∞
    C[0][0] ← ∞
    for i ← 1 to N do                                   ▷ Base Case: 1 Group
        C[i][1] ← P[i]
        K[i][1] ← 0
    end for
    for j ← 2 to M do                                   ▷ Iterate over group counts
        for i ← j to N do                               ▷ Iterate over array length
            for k ← j − 1 to i − 1 do                   ▷ Iterate split points
                current_sum ← P[i] − P[k]
                min_val ← min(C[k][j − 1], current_sum)
                if min_val > C[i][j] then
                    C[i][j] ← min_val
                    K[i][j] ← k                         ▷ Store split point for backtracking
                end if
            end for
        end for
    end for
    Backtrack to find group sizes G using K table
    return G
end function
```

2. Running Time Analysis

Time Complexity

1. **Prefix Sum Calculation:** Iterates 1 to N. Complexity: $\Theta(N)$.
2. **DP Initialization ($j = 1$):** Iterates 1 to N. Complexity: $\Theta(N)$.
3. **DP Table Population:** The nesting structure is:
    - Outer loop (Groups j): $2 \dots M$ ($\approx M$ iterations)

- Middle loop (Elements i): $j \dots N$ ($\approx N$ iterations)
- Inner loop (Split points k): $j-1 \dots i-1$ ($\approx i$ iterations, worst case $O(N)$)

Summation: $\sum_{j=2}^{M} \sum_{i=j}^{N} (i-j+1) \approx \sum_{j=2}^{M} \frac{N^2}{2} = \Theta(M \cdot N^2)$.

4. **Backtracking:** Iterates M times. Complexity: $\Theta(M)$.

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

Space Complexity

We store two tables C and K of size $(N+1) \times (M+1)$. **Total Space Complexity:** $\Theta(N \cdot M)$.

3. Execution Traces

Example 1

Input: $A = \{3, 9, 7, 8, 2, 6, 5, 10, 1, 7, 6, 4\}$, $N = 12, M = 3$.
DP Table $C[i][j]$ (Selected Values):

$j \backslash i$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	3	12	19	27	29	35	40	50	51	58	64	68
2	-	3	7	12	12	16	19	23	24	29	29	33
3	-	-	3	7	7	8	12	15	16	18	19	**19**

Optimal Value: $C[12][3] = 19$. **Backtracking via K Table:**

- $j = 3, i = 12 \rightarrow K[12][3] = 7$. Group 3 is $A[8 \dots 12]$ (Sum 28).
- $j = 2, i = 7 \rightarrow K[7][2] = 3$. Group 2 is $A[4 \dots 7]$ (Sum 21).
- $j = 1, i = 3 \rightarrow K[3][1] = 0$. Group 1 is $A[1 \dots 3]$ (Sum 19).

Result: Group Sizes $[3, 4, 5]$, Sums $[19, 21, 28]$, Min Sum = 19.

```
Optimal Grouping Sizes (G[1..M]):
Group 1: 3 elements
Group 2: 4 elements
Group 3: 5 elements

Group Details:
Group 1: A[1...3] = [3, 9, 7], Sum = 19
Group 2: A[4...7] = [8, 2, 6, 5], Sum = 21
Group 3: A[8...12] = [10, 1, 7, 6, 4], Sum = 28
Optimal Grouping G: 3 4 5
```

Figure 1: Trace visualization for Example 1

Example 2

Input: $A = \{4, 2, 5, 1, 6, 7, 3, 8, 2, 4\}$, $N = 10, M = 3$.
DP Table $C[i][j]$:

$j \backslash i$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	4	6	11	12	18	25	28	36	38	42
2	-	2	5	6	7	12	12	18	18	18
3	-	-	2	2	6	7	7	11	12	**12**

Optimal Value: $C[10][3] = 12$. **Groups:** $A[1 \dots 4]$ (Sum 12), $A[5 \dots 6]$ (Sum 13), $A[7 \dots 10]$ (Sum 17). **Result:** Group Sizes $[4, 2, 4]$, Min Sum = 12.

```
Optimal Grouping Sizes (G[1..M]):
Group 1: 4 elements
Group 2: 2 elements
Group 3: 4 elements

Group Details:
Group 1: A[1...4] = [4, 2, 5, 1], Sum = 12
Group 2: A[5...6] = [6, 7], Sum = 13
Group 3: A[7...10] = [3, 8, 2, 4], Sum = 17
Optimal Grouping G: 4 2 4
```

Figure 2: Trace visualization for Example 2

Example 3

Input: $A = \{5, 2, 4, 7, 1, 3, 6, 8, 2, 4, 9\}$, $N = 11, M = 4$. **Optimal Value:** $C[11][4] = 11$. **Groups:** Sums $[11, 11, 14, 15]$. **Result:** Group Sizes $[3, 3, 2, 3]$.

```
Optimal Grouping Sizes (G[1..M]):
Group 1: 3 elements
Group 2: 3 elements
Group 3: 2 elements
Group 4: 3 elements

Group Details:
Group 1: A[1...3] = [5, 2, 4], Sum = 11
Group 2: A[4...6] = [7, 1, 3], Sum = 11
Group 3: A[7...8] = [6, 8], Sum = 14
Group 4: A[9...11] = [2, 4, 9], Sum = 15
Optimal Grouping G: 3 3 2 3
```

Figure 3: Trace visualization for Example 3

Example 4

Input: $A = \{3, 1, 4, 1, 5, 9, 2, 6, 5, 3\}$, $N = 10, M = 3$. **Optimal Value:** $C[10][3] = 11$. **Groups:** Sums $[14, 11, 14]$. **Result:** Group Sizes $[5, 2, 3]$.

```
Optimal Grouping Sizes (G[1..M]):
Group 1: 5 elements
Group 2: 2 elements
Group 3: 3 elements

Group Details:
Group 1: A[1...5] = [3, 1, 4, 1, 5], Sum = 14
Group 2: A[6...7] = [9, 2], Sum = 11
Group 3: A[8...10] = [6, 5, 3], Sum = 14
Optimal Grouping G: 5 2 3
```

Figure 4: Trace visualization for Example 4

4. C++ Implementation

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <numeric>
using namespace std;

vector<int> max_min_grouping(const vector<int>& A, int N, int M) {
    // DP table: C[i][j] = max min sum for partitioning first i elements into j groups
    vector<vector<int>> C(N + 1, vector<int>(M + 1, INT_MIN));
    // Path table for reconstruction
    vector<vector<int>> K(N + 1, vector<int>(M + 1, -1));

    // Prefix sums for O(1) range sum queries
    vector<int> P(N + 1, 0);
    for (int i = 1; i <= N; ++i) P[i] = P[i - 1] + A[i - 1];

    // Base case: j=1 (1 group)
    for (int i = 1; i <= N; ++i) {
        C[i][1] = P[i];
        K[i][1] = 0;
    }

    // Fill DP Table
    for (int j = 2; j <= M; ++j) {
        for (int i = j; i <= N; ++i) {
            // Try all split points k
            for (int k = j - 1; k < i; ++k) {
                int current_group_sum = P[i] - P[k];
                int min_of_split = min(C[k][j - 1], current_group_sum);

                if (min_of_split > C[i][j]) {
                    C[i][j] = min_of_split;
                    K[i][j] = k;
```

```
                }
            }
        }
    }

    // Backtrack to find group sizes
    vector<int> G(M, 0);
    int idx = N;
    for (int j = M; j >= 1; --j) {
        int k = K[idx][j];
        G[j - 1] = idx - k;
        idx = k;
    }
    return G;
}

int main() {
    vector<int> A = {3,1,4,1,5,9,2,6,5,3};
    vector<int> result = max_min_grouping(A, 10, 3);
    cout << "Optimal Group Sizes: ";
    for(int s : result) cout << s << " ";
    cout << endl;
    return 0;
}
```

Listing 1: Max-Min Grouping Implementation