PROJECT-1 REPORT

PART-1: Pseudo codes of your Dynamic Programming Algorithm

```
Max_Min_grouping(A, N, M)
    Initialize dp[i, j] = INT_MIN for all i, j // Create 2 matrix arrays dp and partitionTable each of size (N+1) x (M+1)
    dp[0, 0] = INT_MAX

    // Compute prefix sums for efficient range sum calculation
    Initialize prefixSums[0] = 0
    For i from 1 to N:
        prefixSums[i] = prefixSums[i - 1] + A[i]

    For i from 1 to N:
        dp[i, 1] = ∑A[k] // k from 1 to i (prefix sum up to i)

// Fill dynamic programming table for multiple groups (j > 1)
For j from 2 to M:
   For i from j to N:
      max_min_value = INT_MIN
      optimal_partition = -1

     // Find the maximum minimum sum by iterating through possible partitions
      For k from j-1 to i-1:
         current_sum = ∑A[m] // m from k+1 to i (using prefix sums)
         min_value = Min(dp[k, j - 1], current_sum)

          If min_value > max_min_value:
              max_min_value = min_value
              optimal_partition = k
      dp[i, j] = max_min_value
      partitionTable[i, j] = optimal_partition

// Return back to find optimal sizes and sums
I = N
J = M

Declare Optimal_size array of size M
Declare Optimal_sum array of size M
```

```
While J > 0:
    start = partitionTable[I, J] + 1
    Optimal_size[J] = I - start + 1
    Optimal_sum[J] = ∑A[m] // m from start to I (using prefix sums)
     I = partitionTable[I, J]
    J = J - 1

// Output the results
Print "Optimal sizes:", Group_size[1] to Group_size[M]
Print "Optimal sums:", Group_sum[1] to Group_sum[M]
Return dp[N, M] // The maximum minimum sum of all groups
```

PART-2: Asymptotic Running Time Analysis

We will examine the code on an individual loop basis, as it contains numerous loops, each of which has its own time complexity.

In the '***PrefixSums***' loop, the calculations are done on only one loop for '**N**' number of times.
So the time complexity is **O(N)**.

During the **initialization of the dpTable** loop, there are two nested loops with iterations of N and M, resulting in a time complexity of **O(N * M)**.

In the **dynamic programming** loop, the filling of *dpTable* and *groupTable* happens using nested for loop where:

- The outer loop executes **M** iterations
- The middle loop executes **N** times for each value of i.
- The inner loop executes **N** times for each combination of i and j.

Given that each iteration is nested, the worst-case time complexity for filling dpTable is **O(M * N * N) = O(M * N^2).**

In the **while** loop, the **optimal sizes** and **sums** are determined through *groupTable* and this loop runs M times. Therefore, the time complexity of the loop is **O(M).**

In the simple **k** loop that determines outputs of optimal sizes and sums, each for loop runs **M** times, hence the time complexity is **O(M)**.

From the aforementioned, it is evident that the **dynamic programming** loop is the dominant factor among the others. Consequently, the program's asymptotic time complexity is **O(M * N^2)**.

PART-3: Output

Output-1:

```
Enter the size of array A: 12
Enter the elements of array A:
3
9
7
8
2
6
5
10
1
7
6
4
Enter the number of groups (M): 3
Optimal sizes: 3 4 5
Optimal sums: 19 21 28
The maximum minimum sum is: 19
Press Enter to exit...
```

Output-2:

```
Enter the size of array A: 15
Enter the elements of array A: 7
23
44
2
71
24
36
17
19
32
18
7
39
68
57
Enter the number of groups (M): 5
Optimal sizes: 3 2 3 4 3
Optimal sums: 74 73 77 76 164
The maximum minimum sum is: 73
Press Enter to exit...
```

Output-3:

```
Enter the size of array A: 9
Enter the elements of array A: 2
4
8
7
6
5
3
1
4
Enter the number of groups (M): 3
Optimal sizes: 3 2 4
Optimal sums: 14 13 13
The maximum minimum sum is: 13
Press Enter to exit...
```

Output-4:

```
Enter the size of array A: 5
Enter the elements of array A: 6
10
3
4
8
Enter the number of groups (M): 2
Optimal sizes: 2 3
Optimal sums: 16 15
The maximum minimum sum is: 15
Press Enter to exit...
```

Output-5:

```
Enter the size of array A: 10
Enter the elements of array A: 5
14
17
8
4
19
2
6
11
7
Enter the number of groups (M): 4
Optimal sizes: 2 2 2 4
Optimal sums: 19 25 23 26
The maximum minimum sum is: 19
Press Enter to exit...
```

PART-4: Source Code

```
#include <iostream>
#include <climits>
#include <algorithm>
#include <array>

const int MAX_N = 45;  // Maximum number of elements in array A
const int MAX_M = 55;  // Maximum number of groups

std::array<int, MAX_N + 1> A;  // Input array
std::array<std::array<int, MAX_M + 1>, MAX_N + 1> dpTable;  // DP table
std::array<std::array<int, MAX_M + 1>, MAX_N + 1> partitionTable;  // To store group partitions

// Function to compute the sum of elements
int FindSum(int port, int dexter, const std::array<int, MAX_N + 1>& prefixSums) {
  return prefixSums[dexter] - prefixSums[port - 1];
}

// Dynamic programming function to solve the problem
int max_min_grouping(int N, int M) {
  std::array<int, MAX_N + 1> prefixSums = { 0 };  // Prefix sum array

  // Compute prefix sums
  for (int i = 1; i <= N; ++i) {
    prefixSums[i] = prefixSums[i - 1] + A[i];
  }

  // Initialize dp table
  for (int i = 0; i <= N; ++i) {
    for (int j = 0; j <= M; ++j) {
      dpTable[i][j] = INT_MIN;
    }
  }
  dpTable[0][0] = INT_MAX;

  // Fill dp table
  for (int j = 1; j <= M; ++j) {
    for (int i = 1; i <= N; ++i) {
      int max_min_v = INT_MIN;
```

```
        int optimal_p = 1;

        for (int k = j - 1; k < i; ++k) {
            int current_sum = FindSum(k + 1, i, prefixSums);
            int min_v = std::min(dpTable[k][j - 1], current_sum);

            if (min_v > max_min_v) {
                max_min_v = min_v;
                optimal_p = k;
            }
        }

        dpTable[i][j] = max_min_v;
        partitionTable[i][j] = optimal_p;
    }
}

// Construct the optimal sizes (G) and optimal sums (B) from the `groupTable`
int i = N;
int j = M;
std::array<int, MAX_M + 1> G;  // Array for optimal sizes
std::array<int, MAX_M + 1> B;  // Array for optimal sums
while (j > 0) {
    int start = partitionTable[i][j] + 1;
    G[j] = i - start + 1;
    B[j] = FindSum(start, i, prefixSums);
    i = partitionTable[i][j];
    --j;
}



std::cout << "Optimal sizes: ";
for (int k = 1; k <= M; ++k) {
    std::cout << G[k] << " ";
}
std::cout << std::endl;

std::cout << "Optimal sums: ";
for (int k = 1; k <= M; ++k) {
    std::cout << B[k] << " ";
```

```
    }
    std::cout << std::endl;

    return dpTable[N][M];
}

int main() {
    int N, M;

    // Prompt user for the size of array A
    std::cout << "Enter the size of array A: ";
    std::cin >> N;

    std::cout << "Enter the elements of array A: ";
    for (int i = 1; i <= N; ++i) {
        std::cin >> A[i];
    }

    std::cout << "Enter the number of groups (M): ";
    std::cin >> M;

    int result = max_min_grouping(N, M);
    std::cout << "The maximum minimum sum is: " << result << std::endl;

    // Pause the program before it closes
    std::cout << "Press Enter to exit...";
    std::cin.ignore();
    std::cin.get();

    return 0;
}
```