LAB REPORT

Group members:

- Laiba Batool
- Mariyam Muzammil
- Hira Arif

Course: DSA

Lab Number: 04

Lab Title Analyzing the Time-bound Operations for Algorithms and verifying their growth rates.

Date: 26/10/24

Objective:

- Practice Sub-Sequence Sum Problem in C++.
- Calculating Running Time of a C++ program.
- Verifying Time bounds of Sub-Sequence Sum Problem.

Description:

In this lab, we implemented three different algorithms to solve the maximum sub-sequence sum problem in C++. We measured how long each algorithm took to run by testing them on inputs of various sizes.

Conclusion:

By completing this lab, we learned how to analyze and compare the performance of algorithms. We successfully verified that the naive algorithm has a time complexity of O(N^3) the improved version is O(N^2) and the divide and Conquer methods run in O(N Log N).After verifying the complexity of each algorithm we successfully implemented the algorithms and measured the time each took for various values of N multiple times. This gave us enough data to create decent readings for plotting. Then, we used matplotlib to create graphs based on the collected data. Also we use two graphs because if we plot all values of algorithms in one graph, the third algorithm is not visible due to the high values of the first and second algorithms.

Main.cpp

```
#include <iostream>
#include <chrono>
#include <thread>
#include <vector>
using namespace std;

// Function implementing the O(n^3) solution
int maxSubSum1(const vector<int> &a){
    int maxSum = 0;
    for(int i=0;i<a.size();i++){
        for(int j=i;j<a.size();j++){
            int thisSum =0;
            // Innermost loop to sum the elements in the subarray from i to j
            for(int k=i;k<=j;k++)
                thisSum += a[k];
            if(thisSum>maxSum)
                maxSum = thisSum;
        }
    }
    return maxSum;
}

// Function implementing the O(n^2) solution
int maxSubSum2(const vector<int> &a){
    int maxSum = 0;
    for(int i=0;i<a.size();i++){
        int thisSum =0;
        for(int j=i;j<a.size();j++){
            thisSum += a[j];
```

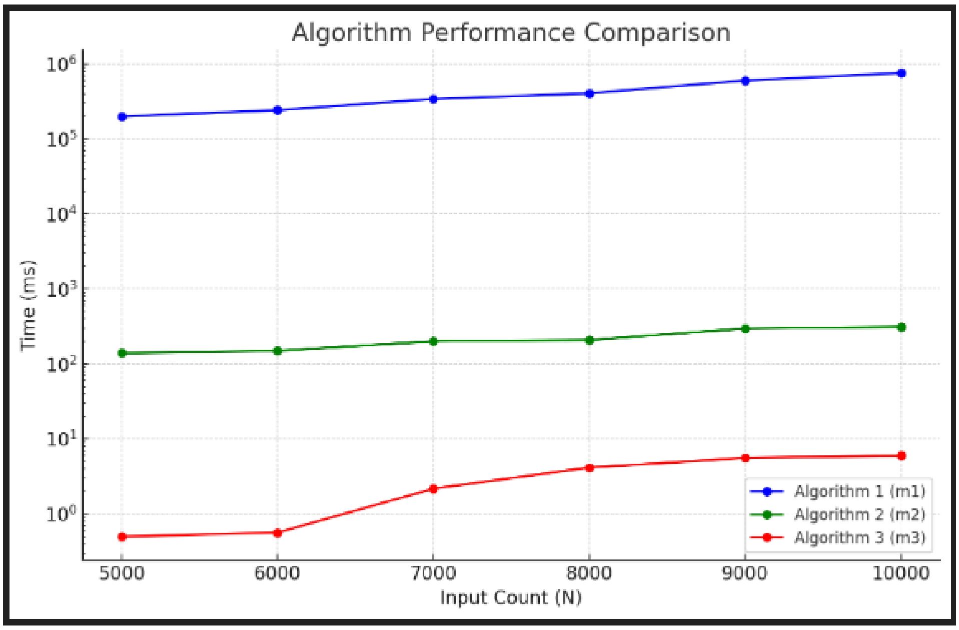
```
         if(thisSum>maxSum)
            maxSum = thisSum;
      }
   }
   return maxSum;
}

// Function implementing the divide and conquer approach (O(n log n))
int maxSubSum3(const vector<int> &a,int left,int right){
   //base case
    if (left==right)
      return a[left];
    else{
      int mid = (left+right)/2;
      int maxLeft = maxSubSum3(a,left,mid);
      int maxRight = maxSubSum3(a,mid+1,right);

      // Calculate the max sum crossing the midpoint from left to right
      int maxLeftBorder = 0;
      int leftBorder = 0;
      for(int i=mid;i>=left;i--){
         leftBorder += a[i];
         if(leftBorder>maxLeftBorder)
            maxLeftBorder = leftBorder;
      }

      int maxRightBorder = 0;
      int rightBorder = 0;
      for(int i=mid+1;i<=right;i++){
         rightBorder += a[i];
```

```
            if(rightBorder>maxRightBorder)
                maxRightBorder = rightBorder;
        }

        // Return the maximum of three: left half, right half, and crossing sum
        if(maxLeft>maxRight && maxLeft>(maxLeftBorder+maxRightBorder))
            return maxLeft;
        else if(maxRight>maxLeft && maxRight>(maxLeftBorder+maxRightBorder))
            return maxRight;
        else
            return maxLeftBorder+maxRightBorder;
    }
}

int main() {
    using std::chrono::high_resolution_clock;
    using std::chrono::duration_cast;
    using std::chrono::milliseconds;

    // Generate random input array with values from -100 to 99
    std::srand(static_cast<unsigned int>(std::time(nullptr)));
    const int inputSize = 10000;
    std::vector<int> input(inputSize);

    for (int i = 0; i < input.size(); i++) {
        input[i] = std::rand() % 200 - 100; // Random values between -100 and 99
    }

    // Measure and display the execution time for maxSubSum1
    auto t1 = high_resolution_clock::now();
    cout << "\tMaxSubSum 1: " << maxSubSum1(input);
    auto t2 = high_resolution_clock::now();
    auto ms_double = std::chrono::duration<double,std::milli>(t2 - t1);
    std::cout << "\tTime: " << ms_double.count() << " ms\n";

    // Measure and display the execution time for maxSubSum2
    t1 = high_resolution_clock::now();
    cout << "\tMaxSubSum 2: " << maxSubSum2(input);
    t2 = high_resolution_clock::now();
    ms_double = std::chrono::duration<double,std::milli>(t2 - t1);
    std::cout << "\tTime: " << ms_double.count() << " ms\n";

    // Measure and display the execution time for maxSubSum3
    t1 = high_resolution_clock::now();
    cout << "\tMaxSubSum 3: " << maxSubSum3(input, 0, input.size() - 1);
    t2 = high_resolution_clock::now();
    ms_double = std::chrono::duration<double,std::milli>(t2 - t1);
    std::cout << "\tTime: " << ms_double.count() << " ms\n";

    return 0;
}
```

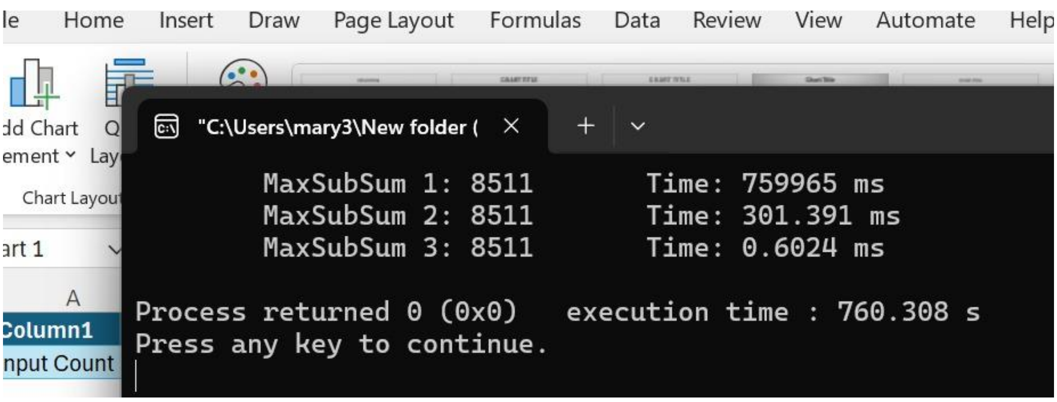
Calculated Average Data for Graph

N	Algorithm 1	Algorithm 2	Algorithm 3
5000	199558.00	139.94	0.4963
6000	242047.00	149.48	0.5644
7000	341444.00	200.30	2.1723
8000	406576.00	207.65	4.1497
9000	601890.93	297.99	5.5998
10000	757430.67	312.17	5.9731

Graph



Output



```
          MaxSubSum 1: 8511        Time: 759965 ms
          MaxSubSum 2: 8511        Time: 301.391 ms
          MaxSubSum 3: 8511        Time: 0.6024 ms

Process returned 0 (0x0)   execution time : 760.308 s
Press any key to continue.
```