BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	2C

AIM:	Experiment based on divide and conquer approach.
Program 1	
PROBLEM STATEMENT :	For this experiment, you need to implement three algorithms to solve the Maximum Subarray Problem: 1. Brute Force Approach 2. Divide and Conquer Approach 3. Kadane's Algorithm The goal is to compare these algorithms based on their time complexity by measuring the time required to compute the maximum subarray sum for varying input sizes. You need to use high_resolution_clock::now() function to find the time required for 100, 200, 300…. 10000 integer numbers. You have to generate 10,000 integer numbers using the C/C++ Rand function and save them in a text file. Finally, compare these three approaches by plotting the time taken by each algorithm. The x-axis of the 2-D plot represents the block no. of 1000 blocks. The y-axis of the 2-D plot represents the running time to find the maximum subarray sum.
PROGRAM (maxsum.cpp):	see below

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

#define NUM_COUNT 10000
#define OUTPUT_FILE "random_numbers.txt"
#define TIME_RESULT_FILE "timing_results.txt"

//block size 100 to 1000; incr by 100

void generate_random_numbers() {
        ofstream file(OUTPUT_FILE);
        if (!file) {
```

```
        cerr << "Error opening file for writing!" << endl;
        return;
        }
        random_device rd;
        mt19937 gen(rd());
        uniform_int_distribution<int> dist(-1000, 1000);

        for (int i = 0; i < NUM_COUNT; i++) {
        file << dist(gen) << endl;
        }
        file.close();
}

void read_numbers(vector<int>& arr, int size) {
        ifstream file(OUTPUT_FILE);
        if (!file)
        {
        cerr << "Error opening file for reading!" << endl;
        return;
        }
        arr.resize(size);
        for (int i = 0; i < size; i++)
        {
        file >> arr[i];
        }
        file.close();
}

int bruteForce(vector<int>& arr) {
        int maxSum = INT_MIN;
        int thisSum = 0;
        int n = arr.size();
        for (int i = 0; i < n; i++) {
        thisSum = 0;
        for (int j = i; j < n; j++) {
        thisSum += arr[j];
        if (thisSum > maxSum) {
                maxSum = thisSum;
        }
        }
        }
        return maxSum;
}

void maxCrossingSum(vector<int>& arr, int low, int mid, int high, int&
ansLow, int& ansHigh, int& ansSum) {
        int leftSum = INT_MIN;
        int sum = 0;
        for (int i = mid; i >= low; i--) {
        sum += arr[i];
        if (sum > leftSum) {
        ansLow = i;
        leftSum = sum;
        }
        }

        int rightSum = INT_MIN;
        sum = 0;
        for (int i = mid + 1; i <= high; i++) {
        sum += arr[i];
        if (sum > rightSum) {
        ansHigh = i;
        rightSum = sum;
        }
        }
        ansSum = leftSum + rightSum;
}

void divideAndConquer(vector<int>& arr, int low, int high, int& ansLow,
int& ansHigh, int& ansSum) {
        if (high == low) {
        ansLow = low;
        ansHigh = high;
        ansSum = arr[low];
        return;
        }
        else {
        int mid = (low + high) / 2;
        int leftLow, leftHigh, leftSum, rightLow, rightHigh, rightSum,
crossLow, crossHigh, crossSum;
        divideAndConquer(arr, low, mid, leftLow, leftHigh, leftSum);
        divideAndConquer(arr, mid + 1, high, rightLow, rightHigh,
rightSum);
        maxCrossingSum(arr, low, mid, high, crossLow, crossHigh,
crossSum);
        if (leftSum >= rightSum && leftSum >= crossSum) {
        ansLow = leftLow;
        ansHigh = leftHigh;
        ansSum = leftSum;

        }
        else if (rightSum >= leftSum && rightSum >= crossSum) {
        ansLow = rightLow;
        ansHigh = rightHigh;
        ansSum = rightSum;
        }
        else {
        ansLow = crossLow;
        ansHigh = crossHigh;
        ansSum = crossSum;
        }
        }
}

int kadanesAlgo(vector<int>& arr) {
        int maxSum = INT_MIN;
        int thisSum = 0;

        // maxSum = thisSum > maxSum ? thisSum: maxSum;
        // thisSum = thisSum < 0 ? 0: thisSum;

        for (int j = 0; j < arr.size(); j++) {
        thisSum += arr[j];
        if (thisSum > maxSum) {
```

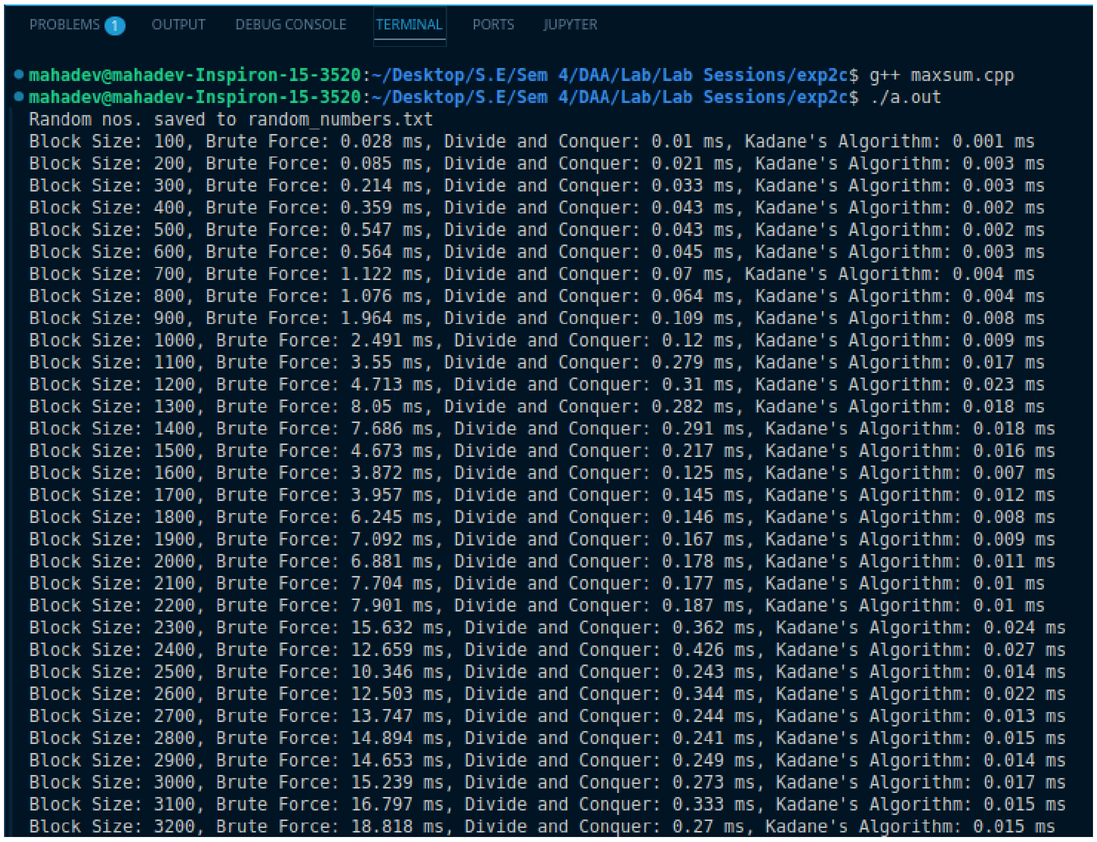
```
        maxSum = thisSum;
        }
        else if (thisSum < 0) {
        thisSum = 0;
        }
        }
        return maxSum;
}

void performTimingAnalysis() {
        ofstream file(TIME_RESULT_FILE);
        if (!file) {
        cerr << "Error opening time result file!" << endl;
        return;
        }
        file << "Block Size\tBrute Force\tDivide and Conquer\tKadane's
Algorithm\n";

        for (int blockSize = 100; blockSize <= NUM_COUNT; blockSize
+= 100) {
        vector<int> arr;
        read_numbers(arr, blockSize);

        auto startBrute = high_resolution_clock::now();
        int bruteResult = bruteForce(arr);
        auto stopBrute = high_resolution_clock::now();
        double bruteTime = duration_cast<microseconds>(stopBrute -
startBrute).count() / 1000.0;

        int ansLow, ansHigh, ansSum;
        auto startDivide = high_resolution_clock::now();
        divideAndConquer(arr, 0, arr.size() - 1, ansLow, ansHigh, ansSum);
        auto stopDivide = high_resolution_clock::now();
        double divideTime = duration_cast<microseconds>(stopDivide -
startDivide).count() / 1000.0;

        auto startKadane = high_resolution_clock::now();
        int kadaneResult = kadanesAlgo(arr);
```

	```
	        auto stopKadane = high_resolution_clock::now();
	        double kadaneTime = duration_cast<microseconds>(stopKadane -
	startKadane).count() / 1000.0;

	        file << blockSize << "\t" << bruteTime << "\t" << divideTime <<
	"\t" << kadaneTime << "\n";
	        cout << "Block Size: " << blockSize << ", Brute Force: " <<
	bruteTime << " ms, " << "Divide and Conquer: " << divideTime << " ms, "
	<< "Kadane's Algorithm: " << kadaneTime << " ms" << endl;
	        }
	        file.close();
	}

	int main() {
	        generate_random_numbers();
	        cout << "Random nos. saved to " << OUTPUT_FILE << endl;
	        performTimingAnalysis();
	        cout << "Max sum & running times computed successfully and
	saved to " << TIME_RESULT_FILE << endl;
	        return 0;
	}
	```

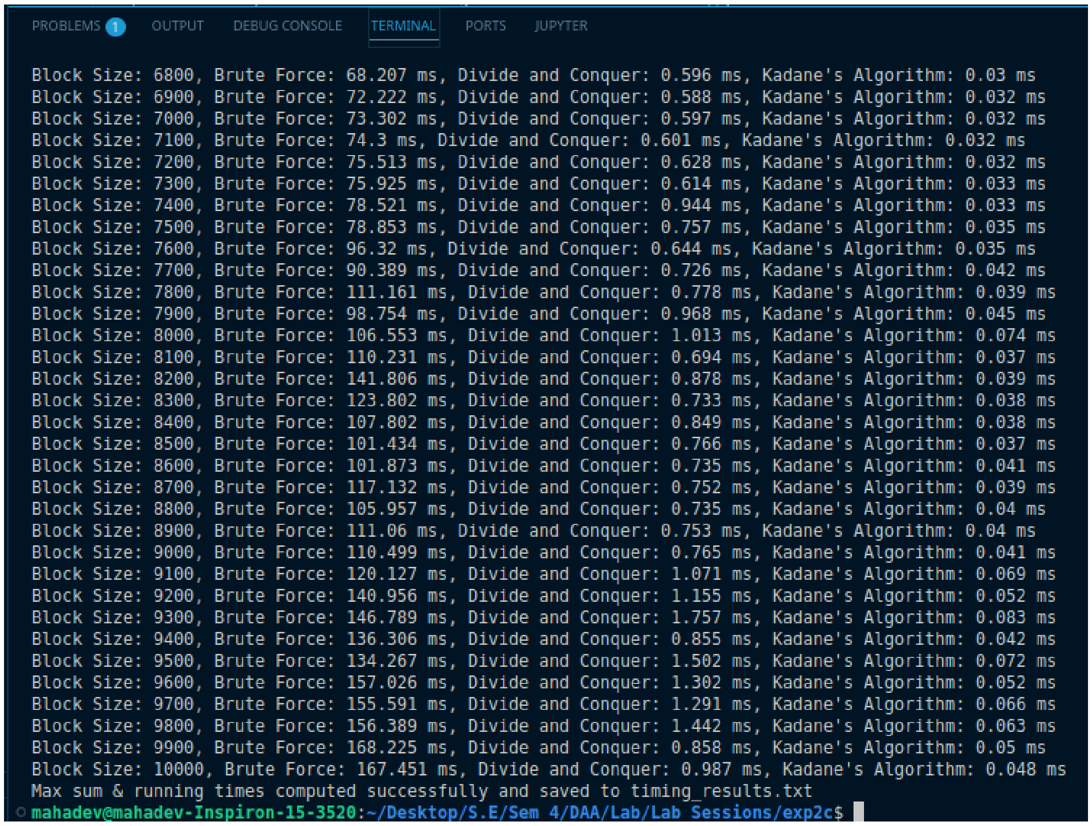
plot.ipynb:	
	```
	import matplotlib.pyplot as plt
	import pandas as pd

	data = pd.read_csv('timing_results.txt', delim_whitespace=True, header=0)

	block_sizes = data['BlockSize']
	brute_force_times = data['BruteForce']
	divide_and_conquer_times = data['DivideandConquer']
	kadane_times = data["Kadane'sAlgorithm"]

	plt.figure(figsize=(8, 6))
	plt.plot(block_sizes, brute_force_times, label='Brute Force', color='blue')
	plt.title('Brute Force Runtime')
	plt.xlabel('Block Size')
	plt.ylabel('Time (ms)')
	plt.grid(True)
	```

```
plt.legend()
plt.show()

plt.figure(figsize=(8, 6))
plt.plot(block_sizes, divide_and_conquer_times, label='Divide and
Conquer', color='green')
plt.title('Divide and Conquer Runtime')
plt.xlabel('Block Size')
plt.ylabel('Time (ms)')
plt.grid(True)
plt.legend()
plt.show()

plt.figure(figsize=(8, 6))
plt.plot(block_sizes, kadane_times, label="Kadane's Algorithm",
color='red')
plt.title("Kadane's Algorithm Runtime")
plt.xlabel('Block Size')
plt.ylabel('Time (ms)')
plt.grid(True)
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(block_sizes, brute_force_times, label='Brute Force', color='blue')
plt.plot(block_sizes, divide_and_conquer_times, label='Divide and
Conquer', color='green')
plt.plot(block_sizes, kadane_times, label="Kadane's Algorithm",
color='red')
plt.title('Comparison of Maximum Subarray Algorithms')
plt.xlabel('Block Size')
plt.ylabel('Time (ms)')
plt.grid(True)
plt.legend()
plt.show()
```

RESULT:

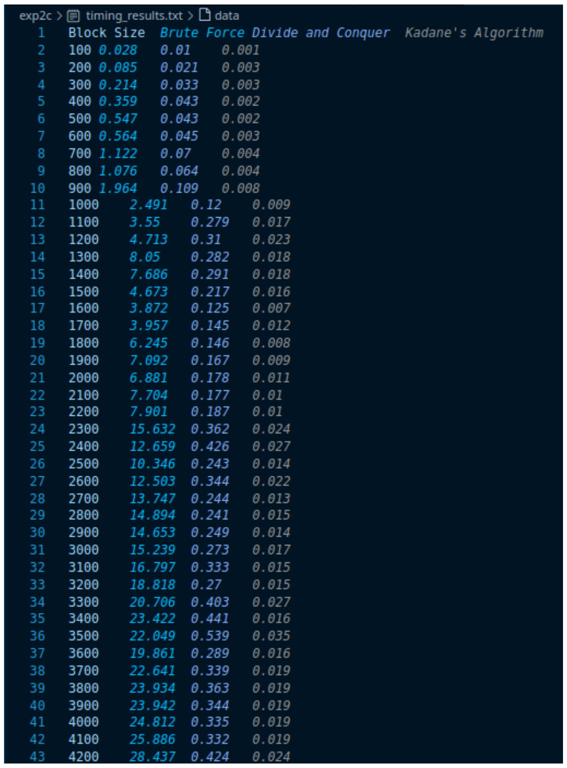

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2c$ g++ maxsum.cpp
mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2c$ ./a.out
Random nos. saved to random_numbers.txt
Block Size: 100, Brute Force: 0.028 ms, Divide and Conquer: 0.01 ms, Kadane's Algorithm: 0.001 ms
Block Size: 200, Brute Force: 0.085 ms, Divide and Conquer: 0.021 ms, Kadane's Algorithm: 0.003 ms
Block Size: 300, Brute Force: 0.214 ms, Divide and Conquer: 0.033 ms, Kadane's Algorithm: 0.003 ms
Block Size: 400, Brute Force: 0.359 ms, Divide and Conquer: 0.043 ms, Kadane's Algorithm: 0.002 ms
Block Size: 500, Brute Force: 0.547 ms, Divide and Conquer: 0.043 ms, Kadane's Algorithm: 0.002 ms
Block Size: 600, Brute Force: 0.564 ms, Divide and Conquer: 0.045 ms, Kadane's Algorithm: 0.003 ms
Block Size: 700, Brute Force: 1.122 ms, Divide and Conquer: 0.07 ms, Kadane's Algorithm: 0.004 ms
Block Size: 800, Brute Force: 1.076 ms, Divide and Conquer: 0.064 ms, Kadane's Algorithm: 0.004 ms
Block Size: 900, Brute Force: 1.964 ms, Divide and Conquer: 0.109 ms, Kadane's Algorithm: 0.008 ms
Block Size: 1000, Brute Force: 2.491 ms, Divide and Conquer: 0.12 ms, Kadane's Algorithm: 0.009 ms
Block Size: 1100, Brute Force: 3.55 ms, Divide and Conquer: 0.279 ms, Kadane's Algorithm: 0.017 ms
Block Size: 1200, Brute Force: 4.713 ms, Divide and Conquer: 0.31 ms, Kadane's Algorithm: 0.023 ms
Block Size: 1300, Brute Force: 8.05 ms, Divide and Conquer: 0.282 ms, Kadane's Algorithm: 0.018 ms
Block Size: 1400, Brute Force: 7.686 ms, Divide and Conquer: 0.291 ms, Kadane's Algorithm: 0.018 ms
Block Size: 1500, Brute Force: 4.673 ms, Divide and Conquer: 0.217 ms, Kadane's Algorithm: 0.016 ms
Block Size: 1600, Brute Force: 3.872 ms, Divide and Conquer: 0.125 ms, Kadane's Algorithm: 0.007 ms
Block Size: 1700, Brute Force: 3.957 ms, Divide and Conquer: 0.145 ms, Kadane's Algorithm: 0.012 ms
Block Size: 1800, Brute Force: 6.245 ms, Divide and Conquer: 0.146 ms, Kadane's Algorithm: 0.008 ms
Block Size: 1900, Brute Force: 7.092 ms, Divide and Conquer: 0.167 ms, Kadane's Algorithm: 0.009 ms
Block Size: 2000, Brute Force: 6.881 ms, Divide and Conquer: 0.178 ms, Kadane's Algorithm: 0.011 ms
Block Size: 2100, Brute Force: 7.704 ms, Divide and Conquer: 0.177 ms, Kadane's Algorithm: 0.01 ms
Block Size: 2200, Brute Force: 7.901 ms, Divide and Conquer: 0.187 ms, Kadane's Algorithm: 0.01 ms
Block Size: 2300, Brute Force: 15.632 ms, Divide and Conquer: 0.362 ms, Kadane's Algorithm: 0.024 ms
Block Size: 2400, Brute Force: 12.659 ms, Divide and Conquer: 0.426 ms, Kadane's Algorithm: 0.027 ms
Block Size: 2500, Brute Force: 10.346 ms, Divide and Conquer: 0.243 ms, Kadane's Algorithm: 0.014 ms
Block Size: 2600, Brute Force: 12.503 ms, Divide and Conquer: 0.344 ms, Kadane's Algorithm: 0.022 ms
Block Size: 2700, Brute Force: 13.747 ms, Divide and Conquer: 0.244 ms, Kadane's Algorithm: 0.013 ms
Block Size: 2800, Brute Force: 14.894 ms, Divide and Conquer: 0.241 ms, Kadane's Algorithm: 0.015 ms
Block Size: 2900, Brute Force: 14.653 ms, Divide and Conquer: 0.249 ms, Kadane's Algorithm: 0.014 ms
Block Size: 3000, Brute Force: 15.239 ms, Divide and Conquer: 0.273 ms, Kadane's Algorithm: 0.017 ms
Block Size: 3100, Brute Force: 16.797 ms, Divide and Conquer: 0.333 ms, Kadane's Algorithm: 0.015 ms
Block Size: 3200, Brute Force: 18.818 ms, Divide and Conquer: 0.27 ms, Kadane's Algorithm: 0.015 ms
```

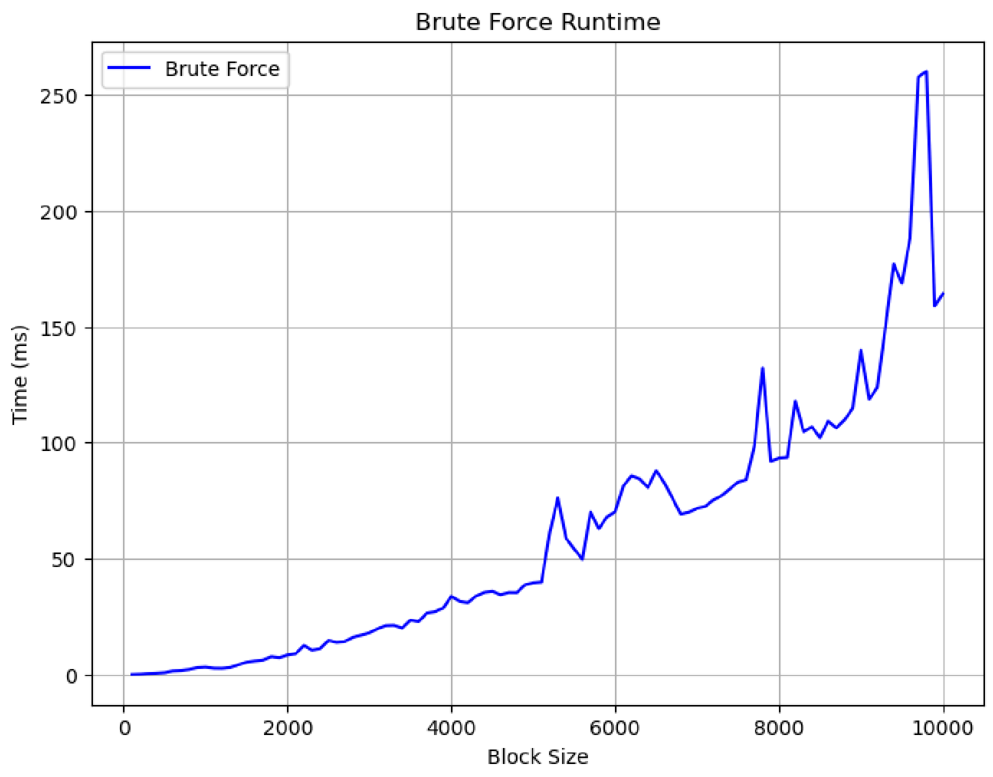


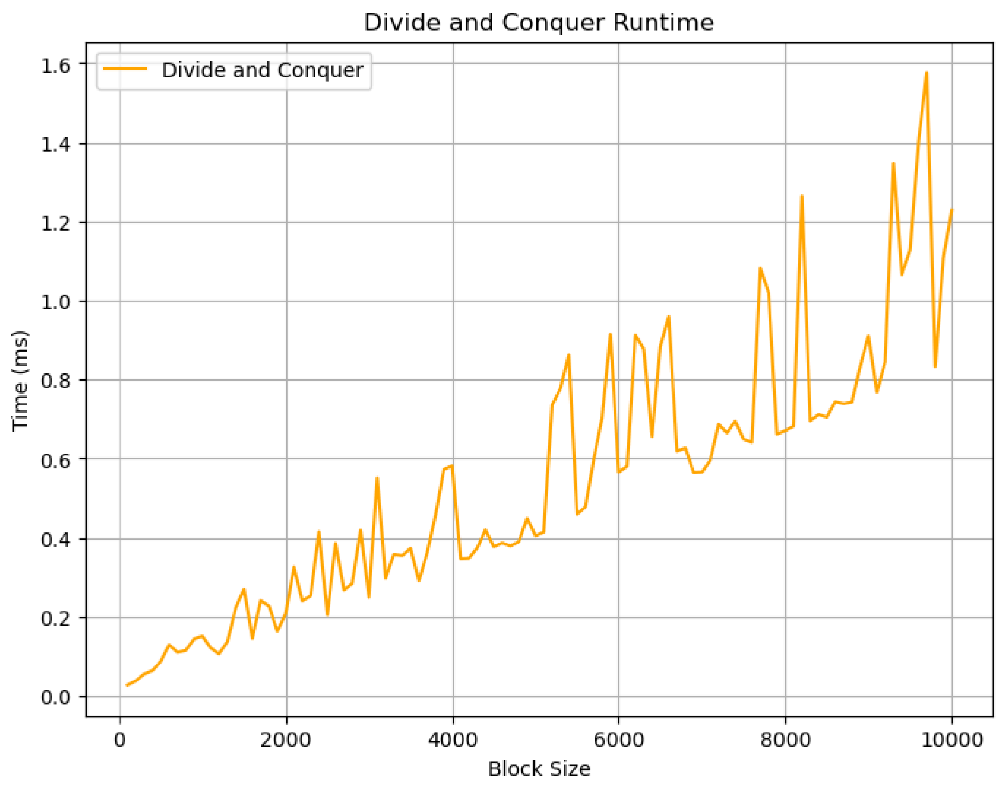
```
Block Size: 6800, Brute Force: 68.207 ms, Divide and Conquer: 0.596 ms, Kadane's Algorithm: 0.03 ms
Block Size: 6900, Brute Force: 72.222 ms, Divide and Conquer: 0.588 ms, Kadane's Algorithm: 0.032 ms
Block Size: 7000, Brute Force: 73.302 ms, Divide and Conquer: 0.597 ms, Kadane's Algorithm: 0.032 ms
Block Size: 7100, Brute Force: 74.3 ms, Divide and Conquer: 0.601 ms, Kadane's Algorithm: 0.032 ms
Block Size: 7200, Brute Force: 75.513 ms, Divide and Conquer: 0.628 ms, Kadane's Algorithm: 0.032 ms
Block Size: 7300, Brute Force: 75.925 ms, Divide and Conquer: 0.614 ms, Kadane's Algorithm: 0.033 ms
Block Size: 7400, Brute Force: 78.521 ms, Divide and Conquer: 0.944 ms, Kadane's Algorithm: 0.033 ms
Block Size: 7500, Brute Force: 78.853 ms, Divide and Conquer: 0.757 ms, Kadane's Algorithm: 0.035 ms
Block Size: 7600, Brute Force: 96.32 ms, Divide and Conquer: 0.644 ms, Kadane's Algorithm: 0.035 ms
Block Size: 7700, Brute Force: 90.389 ms, Divide and Conquer: 0.726 ms, Kadane's Algorithm: 0.042 ms
Block Size: 7800, Brute Force: 111.161 ms, Divide and Conquer: 0.778 ms, Kadane's Algorithm: 0.039 ms
Block Size: 7900, Brute Force: 98.754 ms, Divide and Conquer: 0.968 ms, Kadane's Algorithm: 0.045 ms
Block Size: 8000, Brute Force: 106.553 ms, Divide and Conquer: 1.013 ms, Kadane's Algorithm: 0.074 ms
Block Size: 8100, Brute Force: 110.231 ms, Divide and Conquer: 0.694 ms, Kadane's Algorithm: 0.037 ms
Block Size: 8200, Brute Force: 141.806 ms, Divide and Conquer: 0.878 ms, Kadane's Algorithm: 0.039 ms
Block Size: 8300, Brute Force: 123.802 ms, Divide and Conquer: 0.733 ms, Kadane's Algorithm: 0.038 ms
Block Size: 8400, Brute Force: 107.802 ms, Divide and Conquer: 0.849 ms, Kadane's Algorithm: 0.038 ms
Block Size: 8500, Brute Force: 101.434 ms, Divide and Conquer: 0.766 ms, Kadane's Algorithm: 0.037 ms
Block Size: 8600, Brute Force: 101.873 ms, Divide and Conquer: 0.735 ms, Kadane's Algorithm: 0.041 ms
Block Size: 8700, Brute Force: 117.132 ms, Divide and Conquer: 0.752 ms, Kadane's Algorithm: 0.039 ms
Block Size: 8800, Brute Force: 105.957 ms, Divide and Conquer: 0.735 ms, Kadane's Algorithm: 0.04 ms
Block Size: 8900, Brute Force: 111.06 ms, Divide and Conquer: 0.753 ms, Kadane's Algorithm: 0.04 ms
Block Size: 9000, Brute Force: 110.499 ms, Divide and Conquer: 0.765 ms, Kadane's Algorithm: 0.041 ms
Block Size: 9100, Brute Force: 120.127 ms, Divide and Conquer: 1.071 ms, Kadane's Algorithm: 0.069 ms
Block Size: 9200, Brute Force: 140.956 ms, Divide and Conquer: 1.155 ms, Kadane's Algorithm: 0.052 ms
Block Size: 9300, Brute Force: 146.789 ms, Divide and Conquer: 1.757 ms, Kadane's Algorithm: 0.083 ms
Block Size: 9400, Brute Force: 136.306 ms, Divide and Conquer: 0.855 ms, Kadane's Algorithm: 0.042 ms
Block Size: 9500, Brute Force: 134.267 ms, Divide and Conquer: 1.502 ms, Kadane's Algorithm: 0.072 ms
Block Size: 9600, Brute Force: 157.026 ms, Divide and Conquer: 1.302 ms, Kadane's Algorithm: 0.052 ms
Block Size: 9700, Brute Force: 155.591 ms, Divide and Conquer: 1.291 ms, Kadane's Algorithm: 0.066 ms
Block Size: 9800, Brute Force: 156.389 ms, Divide and Conquer: 1.442 ms, Kadane's Algorithm: 0.063 ms
Block Size: 9900, Brute Force: 168.225 ms, Divide and Conquer: 0.858 ms, Kadane's Algorithm: 0.05 ms
Block Size: 10000, Brute Force: 167.451 ms, Divide and Conquer: 0.987 ms, Kadane's Algorithm: 0.048 ms
Max sum & running times computed successfully and saved to timing_results.txt
mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2c$
```

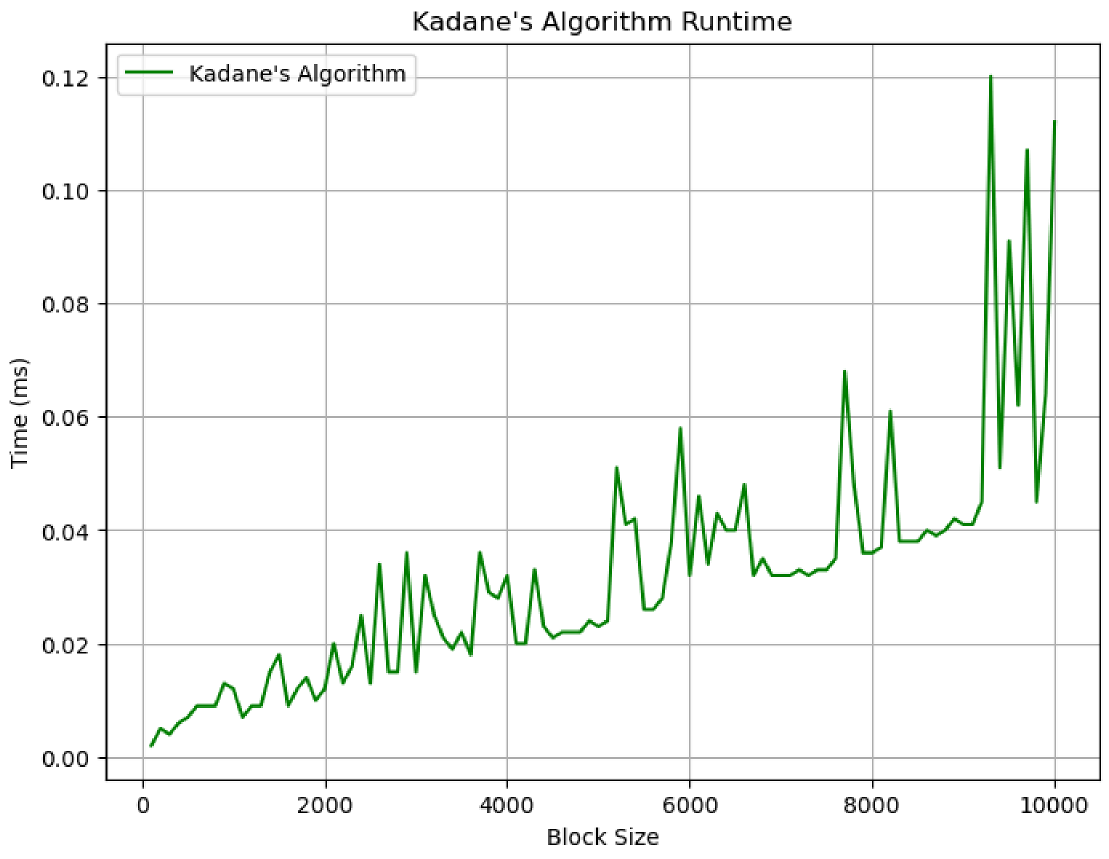
OUTPUT:

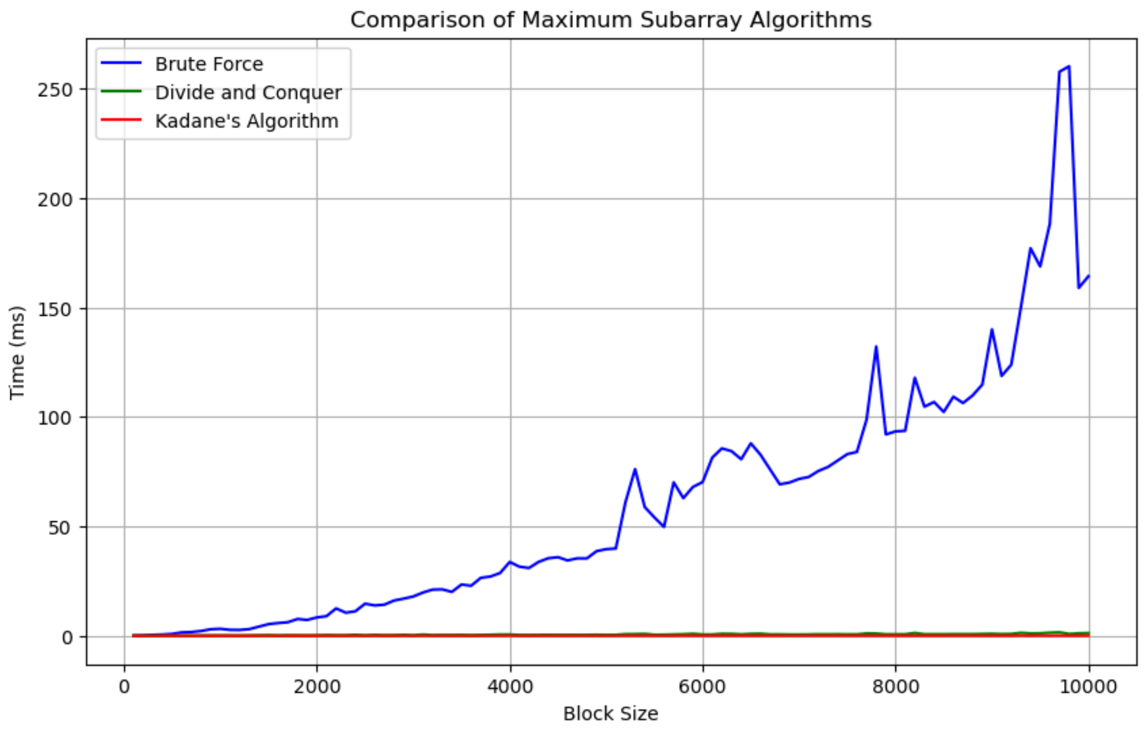


```
Block Size	Brute Force	Divide and Conquer	Kadane's Algorithm
100	0.028	0.01	0.001
200	0.085	0.021	0.003
300	0.214	0.033	0.003
400	0.359	0.043	0.002
500	0.547	0.043	0.002
600	0.564	0.045	0.003
700	1.122	0.07	0.004
800	1.076	0.064	0.004
900	1.964	0.109	0.008
1000	2.491	0.12	0.009
1100	3.55	0.279	0.017
1200	4.713	0.31	0.023
1300	8.05	0.282	0.018
1400	7.686	0.291	0.018
1500	4.673	0.217	0.016
1600	3.872	0.125	0.007
1700	3.957	0.145	0.012
1800	6.245	0.146	0.008
1900	7.092	0.167	0.009
2000	6.881	0.178	0.011
2100	7.704	0.177	0.01
2200	7.901	0.187	0.01
2300	15.632	0.362	0.024
2400	12.659	0.426	0.027
2500	10.346	0.243	0.014
2600	12.503	0.344	0.022
2700	13.747	0.244	0.013
2800	14.894	0.241	0.015
2900	14.653	0.249	0.014
3000	15.239	0.273	0.017
3100	16.797	0.333	0.015
3200	18.818	0.27	0.015
3300	20.706	0.403	0.027
3400	23.422	0.441	0.016
3500	22.049	0.539	0.035
3600	19.861	0.289	0.016
3700	22.641	0.339	0.019
3800	23.934	0.363	0.019
3900	23.942	0.344	0.019
4000	24.812	0.335	0.019
4100	25.886	0.332	0.019
4200	28.437	0.424	0.024
```

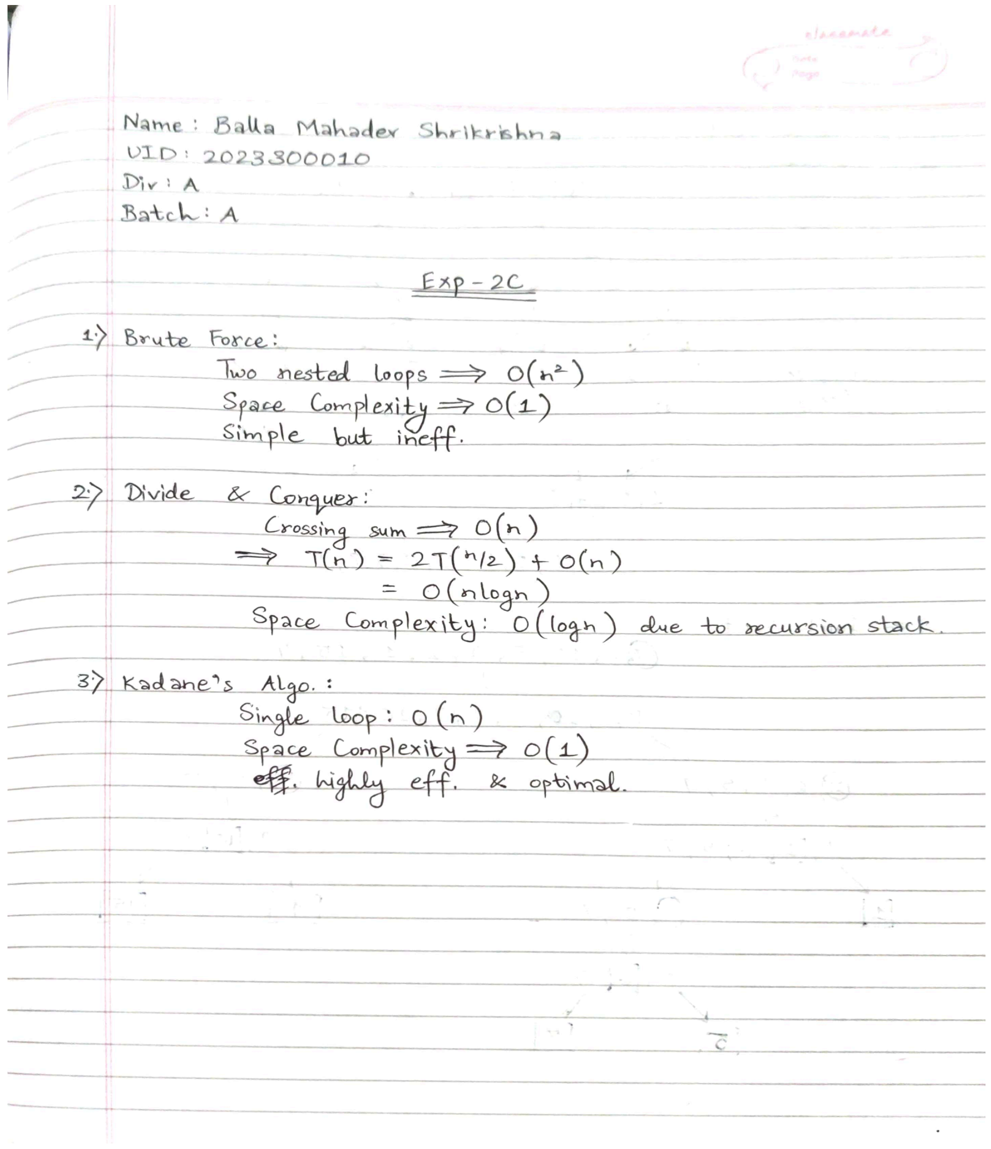

Brute Force Runtime
Brute Force
Time (ms)
0
50
100
150
200
250
0
2000
4000
6000
8000
10000
Block Size


Divide and Conquer Runtime
Divide and Conquer
Time (ms)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
2000
4000
6000
8000
10000
Block Size


Kadane's Algorithm Runtime
Kadane's Algorithm
Time (ms)
Block Size
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
2000
4000
6000
8000
10000


Comparison of Maximum Subarray Algorithms
Brute Force
Divide and Conquer
Kadane's Algorithm
Time (ms)
Block Size
0
50
100
150
200
250
0
2000
4000
6000
8000
10000

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

CONCLUSION:	 Name : Balla Mahadev Shrikrishna UID : 2023300010 Div : A Batch : A <u>Exp - 2C</u> 1:) Brute Force: Two nested loops $\Longrightarrow O(n^2)$ Space Complexity $\Longrightarrow O(1)$ Simple but ineff. 2:) Divide & Conquer: Crossing sum $\Longrightarrow O(n)$ $\Longrightarrow T(n) = 2T(n/2) + O(n)$ $= O(n \log n)$ Space Complexity: $O(\log n)$ due to recursion stack. 3:) Kadane's Algo. : Single loop : $O(n)$ Space Complexity $\Longrightarrow O(1)$ ~~eff.~~ highly eff. & optimal.