Analysis of Execution times for finding maximum subarray using Brute Force and Divide and Conquer Approach

Syed Ahtsham Ul Hassan
04071813015
Analysis and Design of Algorithms
Assignment No. 1

Submitted to: Sir Akmal

23- Nov-2020

Analysis of Brute Force and Divide and Conquer Algorithms' Execution times for Solving Maximum Subarray Problem for different input sizes

I created functions readfile and createfile that are used to create and populate a file with random numbers in range of -1000 to +1000. And the readfile function will then store all the elements into an array of that size.

After that, once the array is populated, the array is passed along with its size to the MaxSubarray functions for Brute Force and Divide and Conquer Approach. These functions will return the maximum subarray. Meanwhile, I ran the time functions to calculate the execution times for these two functions.

I ran the program 4 times using while loop in order to get 4 readings for each file of different size. Thereby, taking the average at the end. And stored these execution times in a separate .csv file. From there I plotted graphs and charts and tables for both approaches.

Time Complexity in terms of Asymptotic Notations

Brute Force Approach Time Complexity:

```
//Brute Force Approach Function
int MaxSubArrayBruteForce(int arr[], int n){
    int max = arr[0];  ------------> 1
    int lower_index = 0;  ------------> 1
    int higher_index = 1;  ------------> 1
    int total = max;  ------------> 1
    for(int i=0; i<n; i++)  ------------> (n+1)
    {

        for(int j=i+1; j<n; j++)  ------------> n(n+1)
        {
           total = total+arr[j]; ------------> n*n
            if(max<total)  ------------> n*n
            {
                max = total;
                lower_index = i;
                higher_index = j;
            }

        }
        total = arr[i+1];  ------------> n
    }
    return max;  ------------> 1
}
```

T(n) = 1+1+1+1+(n+1)+n(n+1)+(n*n)+(n*n) +1

T(n) = 7+n+n^2+n+ n^2+ n^2

$T(n) = 7+2n+3n^2$

Asymptotically:

$T(n) = O(n^2)$

Divide and Conquer Approach Time Complexity:

```
//This function will return the maximum subarray using the
divide and conquer paradigm

int MaxSubArrayDivConquer(int Arr[], int low, int high)
{
    int L, R, C, mid; ⟶ 1
    if(low==high)    ⟶ 1
        return Arr[low];  ⟶ 1
    else{
        mid = (low+high)/2;
        L = MaxSubArrayDivConquer(Arr, low, mid); ⟶ n/2
        R = MaxSubArrayDivConquer(Arr, mid+1, high); ⟶ n/2
        C = MaxCrossing(Arr, low, high, mid); ⟶ n
    }

    return Max(L, R, C); ⟶ 1
}
```

$T(n) = T(n/2) + T(n/2) +1+1+1+n$

$T(n) = 2T(n/2) + 3 + n$

$T(n) = 2T(n/2) + n$

$T(n) = \{ 2T(n/2) + n \text{ for } n>1 \quad \& \quad 1 \text{ for } n=1$

Therefore, Using Master Theorem

$f(n) = (n^k (\log n)^p) = (n^1 (\log n)^0)$

$a=2, b=2, k=1, p=0$

$\log_b a = k$

$\log_2 2 = 1$

$1=1$ (Case 1)

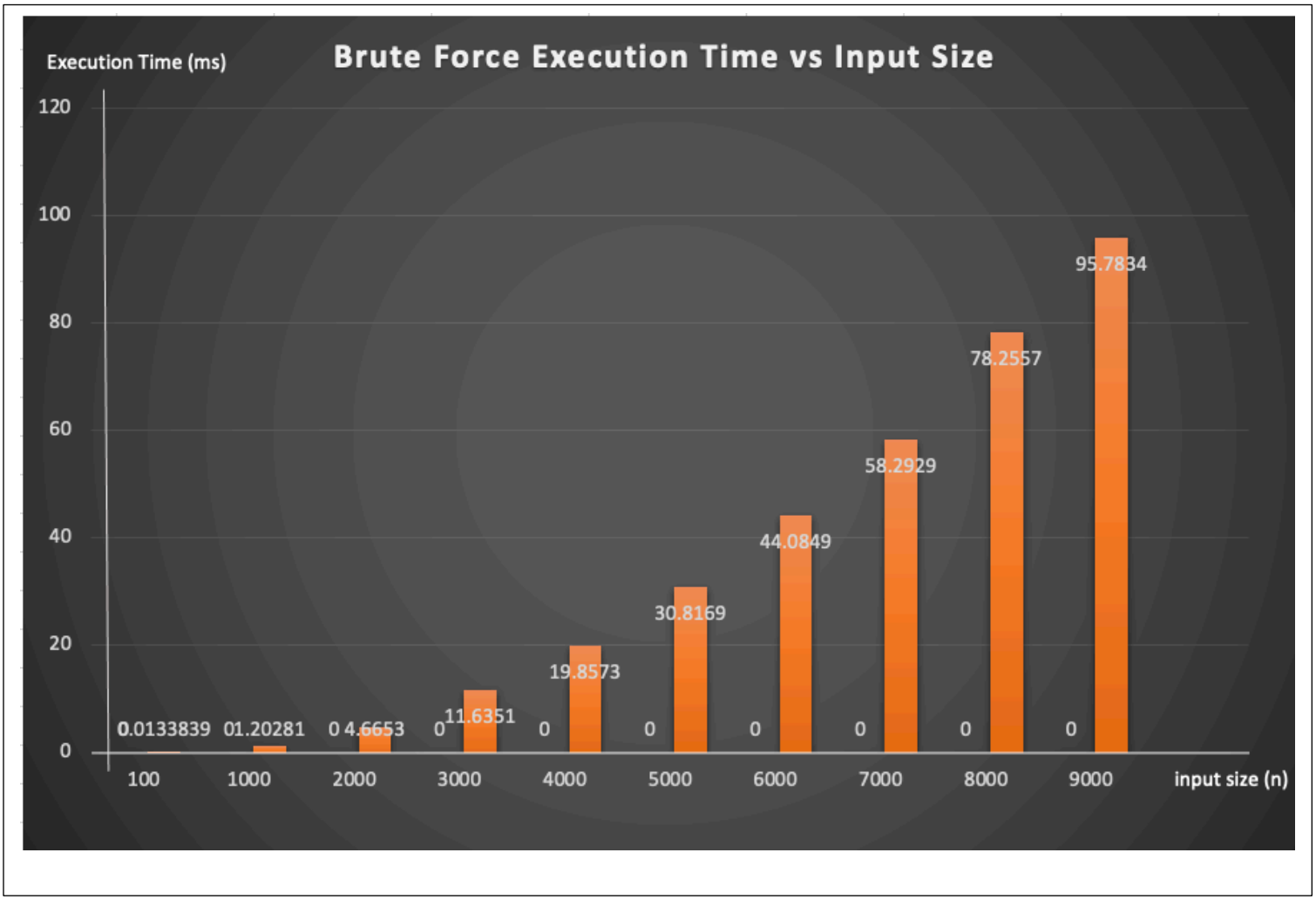
$\theta (n^1 (\log n)^{0+1})$

Time Complexity in Asymptotic Notation:

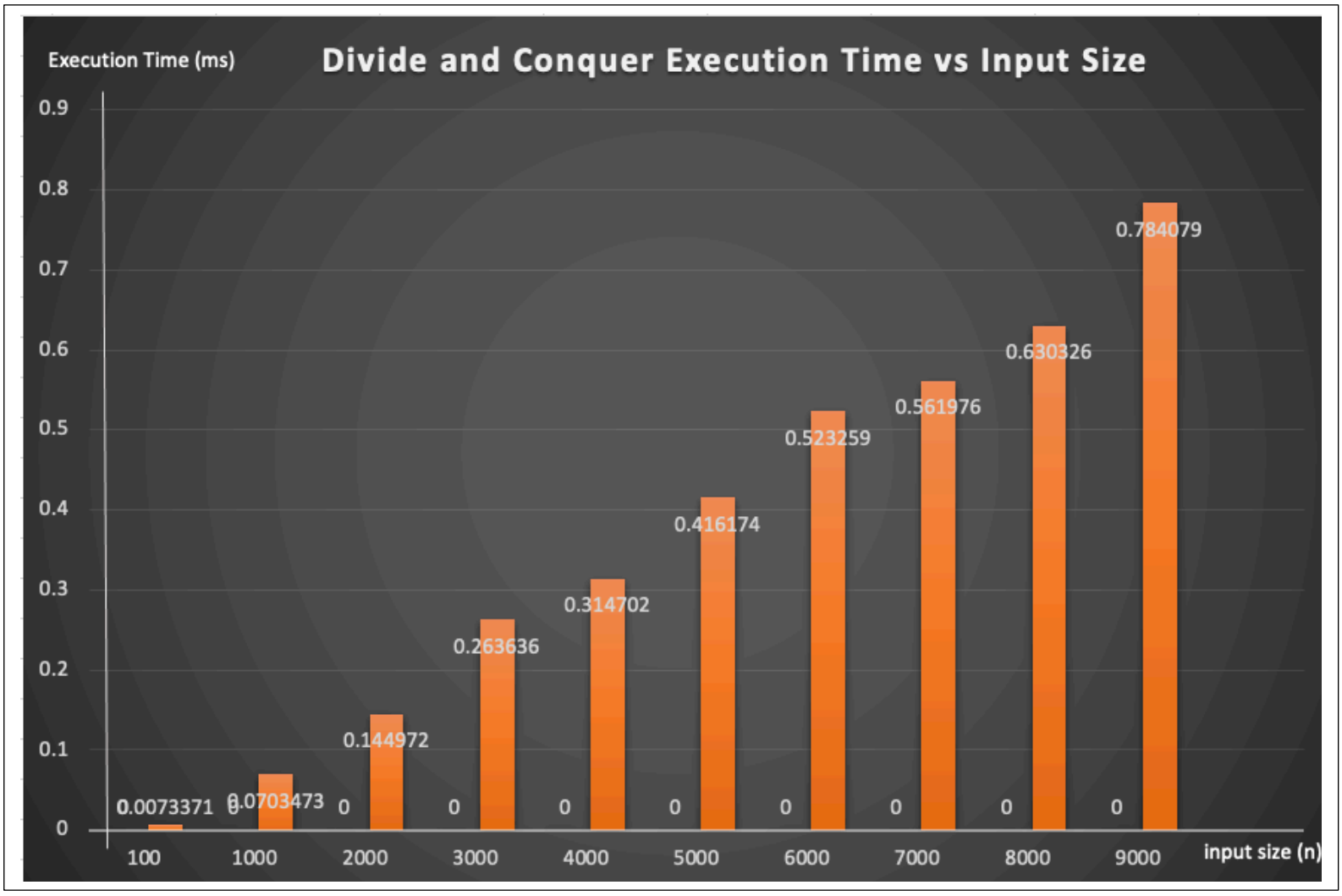
$T(n) = \theta (n \log n)$

Graphs and Charts for the Execution Times and input sizes for both Algorithms (Divide and Conquer and Brute Force)

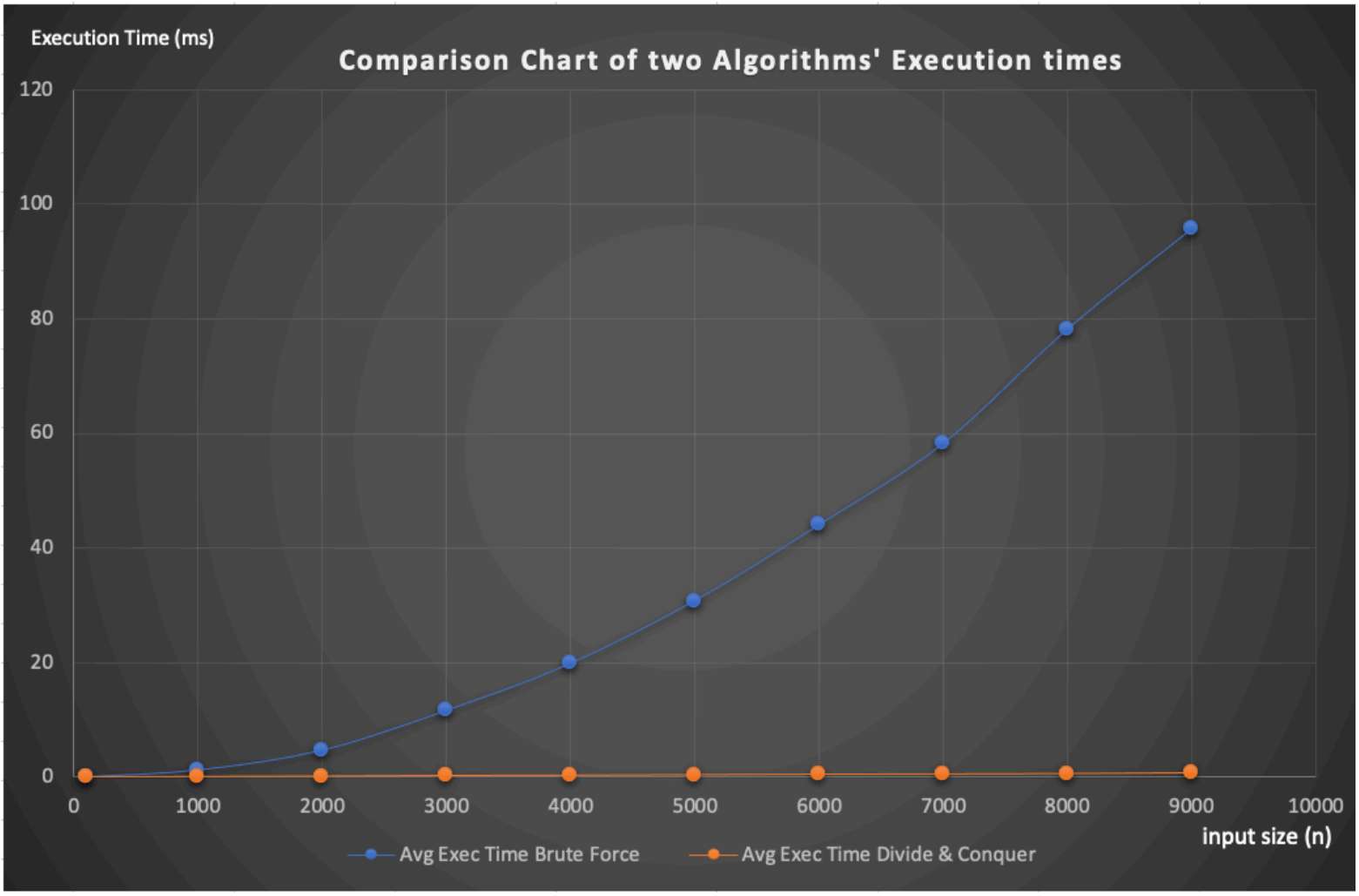
Graph 1:



Graph 2:


Execution Time (ms)
Divide and Conquer Execution Time vs Input Size
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.0073371
0.0703473
0.144972
0.263636
0.314702
0.416174
0.523259
0.561976
0.630326
0.784079
0
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
input size (n)

Graph 3:



Note: Since the divide and conquer approach is taking very less amount of time for the given input sizes therefore the comparison is not clear.

Tables are given in the Excel Sheet in the same folder with name 'Execution Times Tables.xls'.

Conclusions:

We asymptotically found that the time complexity of **Brute force is $O(n^2)$** whereas **for the divide and conquer it is O(n log n)**. And we have implemented the algorithms as functions for finding the **Maximum Subarray** from a given Array of certain size. We found a significant difference between the execution times for both approached for 10 different input sizes.

For example: for input size **n=9000**, the **Brute force** is taking around **99.4947 milliseconds** while **the Divide and Conquer** is taking only **0.711194 milliseconds**.

Hence, the Divide and Conquer approach is the best to find out the maximum subarray of a given size because it performed very well for all input sizes.