Name: Tran Ngoc Son (陳玉山)
ID: 0710185

Report Homework 2

Contents

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

I. Introduction

In this homework, we are implementing the assignment call "Maximum sub-array" by using two different method, Brute force method and Divide and conquer method. After finishing the program, we will make the comparison between the running time of the two method with the small to large input size.

II. Methodology

a) Brute Force

For this method, we assume that the maximum-sub array may start at any position of array; therefore, we go through each elements starting from index 0 to the end of the array, we define a variable call "max" to track the maximum sub-array, and the variable "sum" to calculate the sum from the current index to the end. If sum is greater than max, max is then modified by sum.

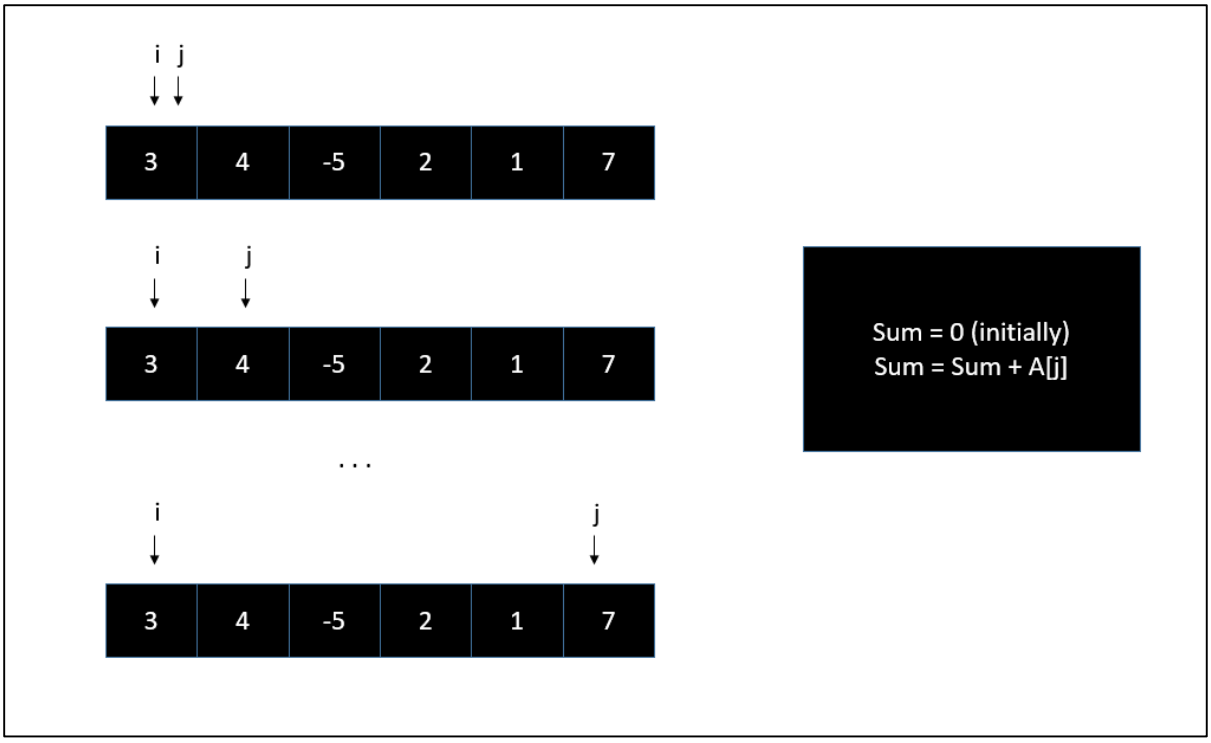


Figure 1: Brute Force method visualization

As we can see in figure (2) below, there would be two for loops to execute this method. The outer loop is to picking a starting index, the inner loop is to go throught the other indexes including starting index to calculate the sum. By repeating calculate the sum and comparing it with the max, we would finally find out which sub-array has the maximum value. This method is intuitive to understand, however it may cost a lot of time when the input size becomes significantly large, we will discuss it in section III.

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

```
int brute_force(int arr[], int n)
{
    int max=INT_MIN;
    int sum;
    for (int i=0; i<=n;i++)
    {
        sum =0;
        for (int j=i; j<=n;j++)
        {
            sum = sum+arr[j];
            if (sum>max)
            max = sum;
        }
    }
    return max;
}
```

Figure 2: Brute Force method's code

b) Divide and conquer.

As we have studied a method called “Divide and conquer”, we now apply it to this assignment. The idea of it is to recursively divide the array into multiple sub-arrays until there are only 2 or 1 element left, and we need to deal with it. If our code works fine with a small sub-array, it means that there is a high chance it can also deal with a large array when we apply the recursion.

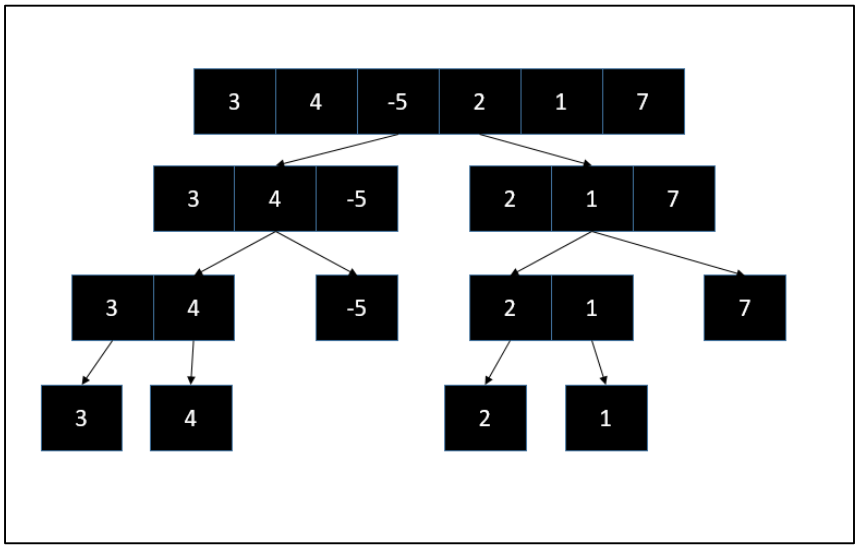


Figure 3: Divide and Conquer visuallization

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

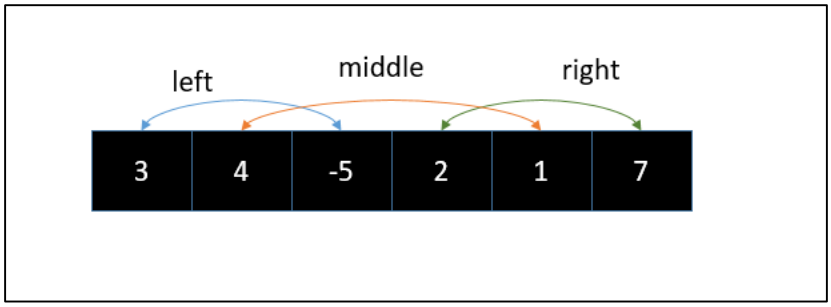


Figure 4: Three parts of an array

From figure (3), we can see that the array is divided into small sub-array with the high of log(n). Until there is one element left, it will return and do the comparison task. However, the largest sub-array might belong to the left part, right part, or middle; therefore, we not only need to compare the maximum sum of the left and right part, but also calculate the maximum sum of the middle part. We finally choose the part that has the largest value to be the maximum sub-array value.

```
int find_max_mid(int arr[],int low, int mid, int high)
{
    int max_left = INT_MIN;
    int sum=0;
    int left_index, right_index;
    for (int i=mid; i >=low; i--)
    {
        sum = sum + arr[i];
        if (sum>max_left)
        {
            max_left = sum;
        }
    }
    int max_right = INT_MIN;
    sum = 0;
    for (int i=mid+1; i<=high;i++)
    {
        sum = sum + arr[i];
        if (sum>max_right)
        {
            max_right = sum;
        }
    }
    int max_overall = max(max(max_left,max_right),max_left+max_right);
    return max_overall;
}
```

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

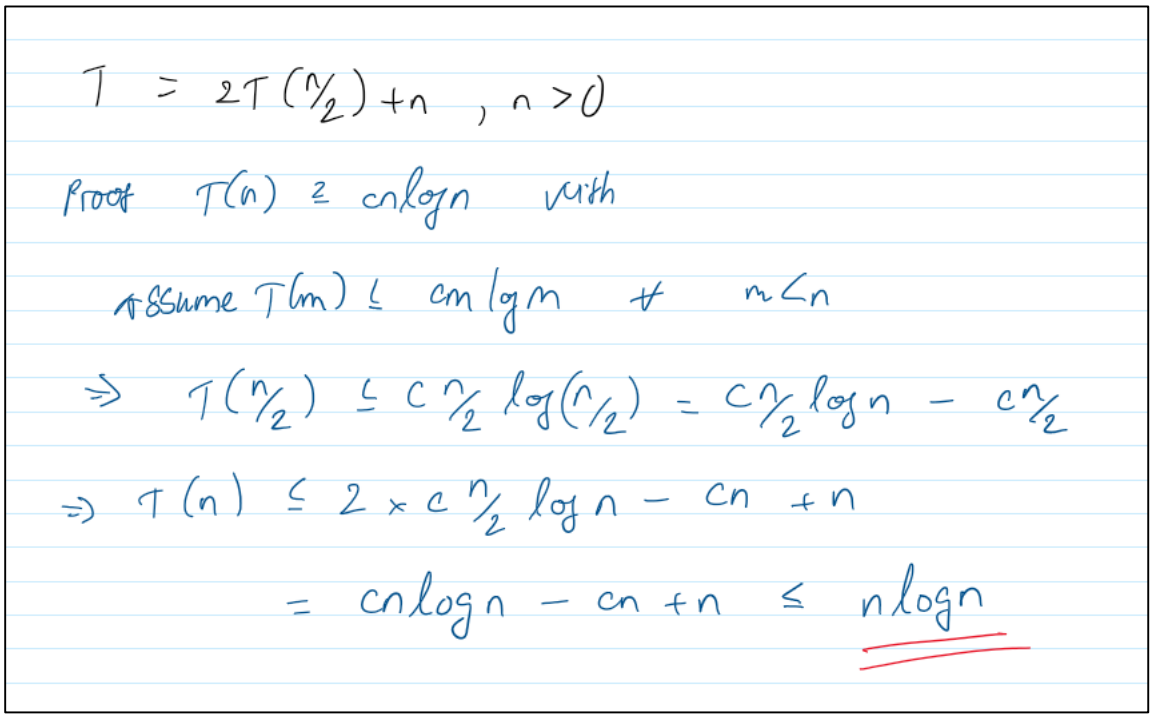
```
int max_sub(int arr[],int low, int high)
{
    if (low==high) return arr[low];
    int mid = low + (high-low)/2;
    int max_left = max_sub(arr, low,mid);
    int max_right = max_sub(arr, mid+1, high);
    int max_middle = find_max_mid(arr,low,mid,high);
    int max_overall = max(max(max_left,max_right),max_middle);
    return max_overall;
}
```

Figure 5: Divide and Conquer method

III. Results

a) Big O analysis

In Divide and Conquer method, we can see that its time complexity can calculate by using this formula $T(n) = 2T\left(\frac{n}{2}\right) + n$. From that, we can prove its big O would be nlog(n) by using the below method.



In the Brute Force method, with using two for loops, and in the worst case, these loops will run through every element of the array. Therefore, we know that its big O is n^2 .

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

b) Executing time

n	Divide and Conquer (microseconds)	Brute force (microseconds)
100	0	0
200	0	0
1000	0	1997
1500	0	3000
2000	0	3995
3000	1004	11064
4000	0	19036
10000	1006	112993
15000	996	244989
20000	2050	433029
30000	3999	1140038
100000	8000	10721992

From the above table, we can see that there is a huge difference in terms of running time when using the two algorithms. When the input size is relatively small, there are not many differences. However, when then the input size is significantly large, the divide and conquer method shows a really fast performance, 8000 microseconds compared to 10721992 of the Brute Force method with the input size of 100 000. Taking advantage of it, we can use the Brute Force method when dealing with small input sizes because of the simplicity in coding, and using the Divide and Conquer method for large input for saving the calculating time.

c) Running time comparison

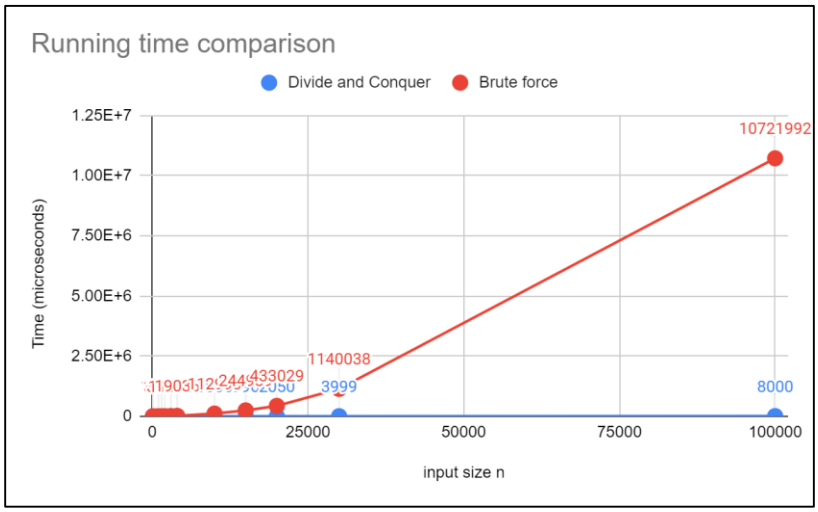


Figure 6: Time comparison

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

IV. Discussion

This assignment is to implement and compare the running time of the two methods, and I was surprised by how fast the Divide and Conquer method performs. It also shows me clearly why should we need to optimize our algorithm when dealing with a very large input size. Furthermore, through this homework, we can have a deeper understanding of how to use the really useful method, Divide and Conquer.