Task Name : Maximal Subarray

Task Number : 8

Student ID	Student Name
20210140	اسامه سمير عبدالمنعم
20210810	محمد عبدالله عويس
20210014	احلام محمد محمد
20211014	نوران محمد عبدالكريم
20210310	دولاجي سامح لويس
20210325	رحمه خالد طه

- Non-Recursive

I. Pseudocode

```
main fun

array[] <- {2,-4,1,,9,-6,7,-3}
n <- sizeof(array)/ sizeof(array[0])
print(maxSubArray(array,n))

maxSubArray(array,n)
    max_sum <- array[0]
    for i<-0 to n Do
        for j<-i to n Do
            s<- 0
            for k<- I to j Do
                s <- s+array[k]
            if  s> max_sum Then
                max_sum <- s
    retuen max_sum
```

II. Analysis

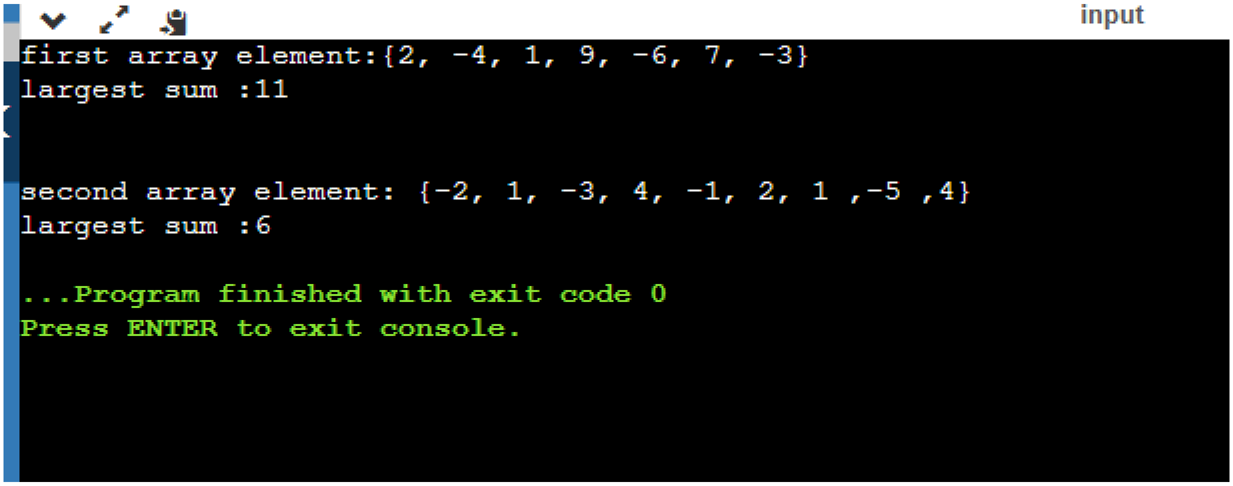
```
                                              Const      Time

fun maxSubArray(array,n)
    max_sum <- array[0]                       C5         1
    for i<-0 to n Do                          C6         n
        for j<-i to n Do                      C7         n
            s<- 0                             C8         1
            for k<- I to j Do                 C9         n
                s <- s+array[k]               C10        1
            if  s > max_sum Then              C11        1
                max_sum <- s                  C12        1
    retuen max_sum
main fun
    array[] <- {2,-4,1,,9,-6,7,-3}            C1         1
    n <- sizeof(array)/ sizeof(array[0])      C2         1
    print(maxSubArray(array,n))
```

input size n

$T(n)=n*n*n = n^3$

$O(n^3)$ $\Omega(n^3)$ $\Theta(n^3)$

III. Output



```
first array element:{2, -4, 1, 9, -6, 7, -3}
largest sum :11


second array element: {-2, 1, -3, 4, -1, 2, 1 ,-5 ,4}
largest sum :6

...Program finished with exit code 0
Press ENTER to exit console.
```

- Recursive

IV. Pseudocode

```
fun max(integer a, integer b)
    if a>b Then
        return a
    else Then
        return b
fun maxSubArrayHelper (arr[],interge  left, interger right)
    if left = right Then
        return arr[left]
    mid <- (left+right)/2
    left_max <- maxSubArrayHelper(arr[],left,mid)
    right_max <- maxSubArrayHelper (arr[],mid+1,right)
    corss_max <-  arr[mid]
    temp_sum <-  arr[mid]
    for i<-mid-1 to left Do
        temp_sum <- temp_sum + arr[i]
        corss_max <- max(corss_max, temp_sum)
    temp_sum = corss_max
    for j<-mid+1 to right Do
        temp_sum <- temp_sum + arr[i]
        corss_max <- max(corss_max, temp_sum)
    return max(max(right_max, left_max), corss_max)
fun maxSubArray(arr[] , n )//n is array size
    return  maxSubArrayHelper(arr,0,n-1)
main fun
    n  // size of arr
    print(enter size of array)
    scan(n)
    arr[n]
    print(enter the array elements)
    for i<-0 to n Do
        scan(arr[i])
    return 0
```

V. Analysis

```
                                                                     time
fun max(integer a, integer b)
    if a>b Then                                                         1
        return a
    else Then
        return b
fun maxSubArrayHelper (arr[],interge  left, interger right)
    if left = right Then                                                1
        return arr[left]
    mid <- (left+right)/2                                               1
    left_max <- maxSubArrayHelper(arr[],left,mid)                  T(n/2)
    right_max <- maxSubArrayHelper (arr[],mid+1,right)             T(n/2)
    corss_max <-  arr[mid]                                              1
    temp_sum <-  arr[mid]                                               1
    for i<-mid-1 to left Do                                           n/2
        temp_sum <- temp_sum + arr[i]                                   1
        corss_max <- max(corss_max, temp_sum)                           1
    temp_sum = corss_max                                                1
    for j<-mid+1 to high Do                                    n/2
        temp_sum <- temp_sum + arr[i]                          1
        corss_max <- max(corss_max, temp_sum)                  1
    return max(max(right_max, corss_max), corss_max)
fun maxSubArray(arr[] , n )//n is array size
    return  maxSubArrayHelper(arr,0,n-1)
main fun
    n  // size of arr
    print(enter size of array)
    scan(n)
    arr[n]
    print(enter the array elements)
    for i<-0 to n Do                                           n
        scan(arr[i])
    return 0
```

input size n

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

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

Solve By Master Method

$a = 2 \qquad b = 2 \qquad f(n) = 2n$

$n^{\log_b a} \qquad F(n)$

$n^1 \qquad 2n^1$

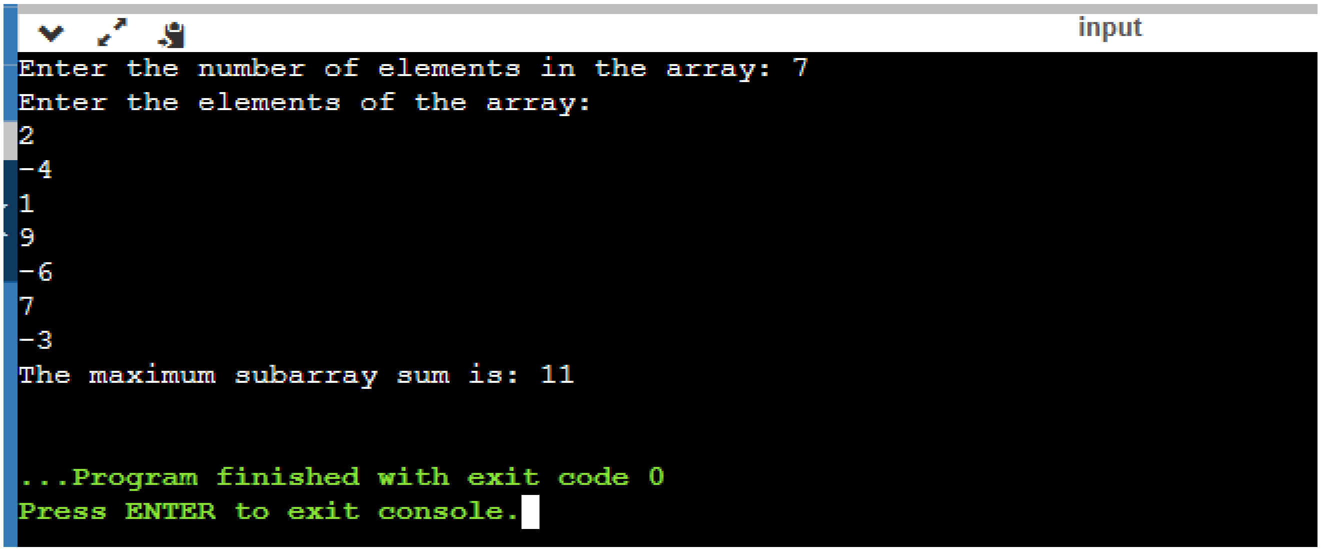
$n \qquad = \qquad 2n$

case two

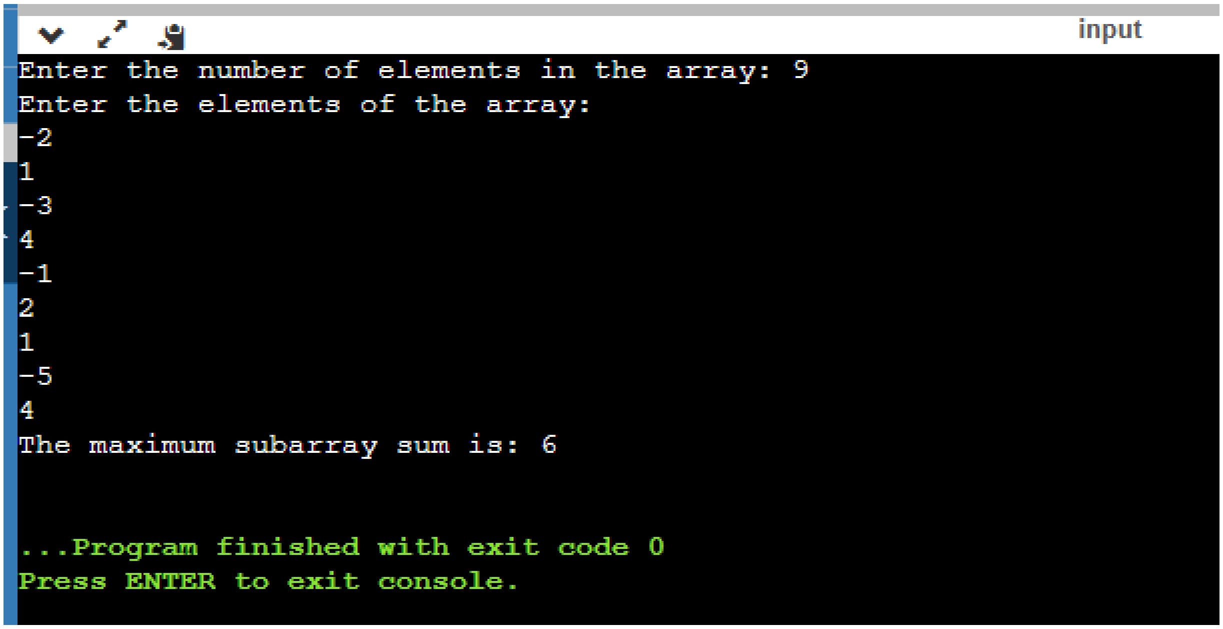
$T(n) = n \log n$

$O(n \log n) \qquad \Omega\ (n \log n) \qquad \Theta(n \log n)$

VI. Output



```
Enter the number of elements in the array: 7
Enter the elements of the array:
2
-4
1
9
-6
7
-3
The maximum subarray sum is: 11


...Program finished with exit code 0
Press ENTER to exit console.
```



```
Enter the number of elements in the array: 9
Enter the elements of the array:
-2
1
-3
4
-1
2
1
-5
4
The maximum subarray sum is: 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Comparison

	Non-Recursive	Recursive
Best Case	n^3	$n \log n$
Worst Case	n^3	$n \log n$
Average Case	n^3	$n \log n$

The best Code is Recursive because $n \log n < n^3$