Analysis of algorithm for Finding a Maximum Subsequence Sum through Divide-and-Conquer

```
FIND-MAXIMUM-SUBARRAY (A, low, high)
  1.  if high == low
  2.        return (low, high, A[low])
  3.  else mid = ( low + high ) / 2
  4.  (left-low, left-high, left-sum) = FIND-MAXIMUM-SUBARRAY ( A, low, mid)
  5.  (right-low, right-high, right-sum) = FIND-MAXIMUM-SUBARRAY ( A, mid+1, high)
  6.  (cross-low, cross-high, cross-sum) = FIND-MAX-CROSSING-SUBARRAY ( A, low, mid, high)
  7.  if left-sum ≥ right-sum and left-sum ≥ cross-sum
  8.        return (left-low, left-high, left-sum)
  9.  elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
  10.       return (right-low, right-high, right-sum)
  11. else
  12.       return (cross-low, cross-high, cross-sum)
```

```
FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high)
  1.  left-sum = -∞
  2.  sum = 0
  3.  for i = mid downto low
  4.        sum = sum + A[i]
  5.        if sum > left-sum
  6.              left-sum = sum
  7.              max-left = i
  8.  right-sum = -∞
  9.  sum = 0
  10. for j = mid + 1 to high
  11.       sum = sum + A[j]
  12.       if sum > right-sum
  13.             right-sum = sum
  14.             max-right = j
  15. return (max-left, max-right, left-sum + right-sum)
```

Explanation of Algorithm:

In this algorithm we use the Divide and Conquer strategy:

The base case for this algorithm, when there is only one element that element is returned as maximum sum subarray.

We **divide** the input array A[low.. high] at the midpoint into two equal size subarrays.

In the **conquer** step we find the maximum subarrays from A[low .. mid] and A[mid+1 .. high]. We search for a maximum subarray which crosses the midpoint. To do this, starting from the midpoint we add elements to the left of mid that is from mid to low, so as to get a maximum sum on the left side of mid (mid included). Similarly, starting from the right of midpoint we add elements, that is from mid to high,

so as to get a maximum sum on the right side of mid. We then add the left-sum and right-sum to get cross sum.
For every maximum subarray calculated it will either lie to the left of midpoint, right of midpoint or it will cross the midpoint. Hence in the **combine** step we return the maximum of the three subarrays.

Time Complexity Analysis

For the **analysis** of this algorithm we assume that the input size is a power of 2.

For the FIND-MAXIMUM-SUBARRAY line 1 and 2 is **base condition** and takes Θ(1). At line 3 as we are dividing the input size by 2 the time required for each side is T(N/2) hence the total time required will be 2 T(N/2) . Therefore lines 4 and 5 take **2 T(N/2)** running time.

For the FIND-MAX-CROSSING-SUBARRAY lines 1, 2, 3, 8, 9 and 15 take Θ(1) time to run. The for loop at line 3 makes mid – low + 1 iterations and the for loop at line 10 makes high – mid – 1 + 1 iterations. Each statement within these for loops have a constant running time of Θ(1). The total number of iterations made by the for loops is

(mid – low + 1) + (high – mid – 1 + 1) = high – low + 1 = n (the input size)

Hence the total running time of **FIND-MAX-CROSSING-SUBARRAY is Θ(N).**
The recurrence relation for FIND-MAXIMUM-SUBARRAY can be given as

$$T(N) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(N/2) + \Theta(N) & \text{if } n > 1 \end{cases}$$

Solving this recurrence using **Master Method** :

$$T(N) = 2T(N/2) + \Theta(N)$$

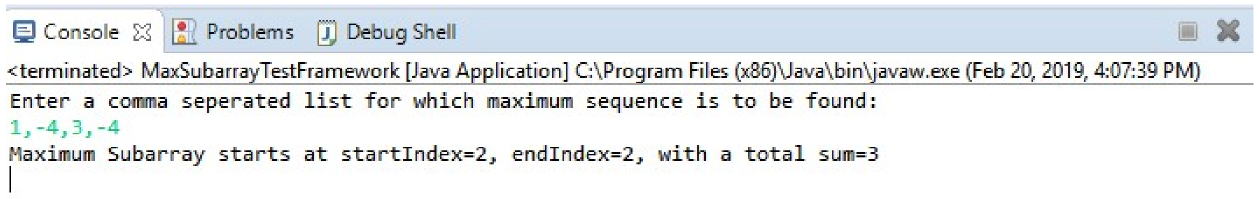
We have **a = 2**,
b = 2,
f(N) = Θ(N)

$$\mathbf{N^{\log_b a}} = N^{\log_2 2}$$
$$= N^1 = N = f(N)$$

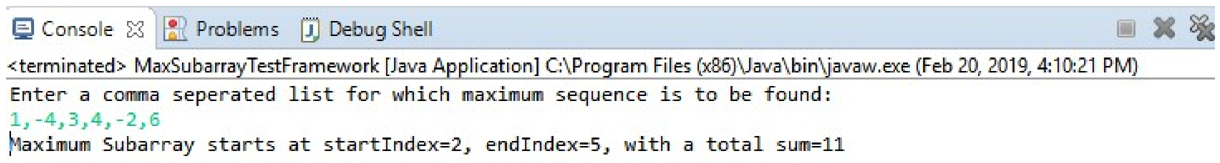
Hence as per **Case 2** of Master method we can say,

$$T(N) = \Theta(N^{\log_b a} \log N)$$
$$= \mathbf{\Theta(N \log N)}$$

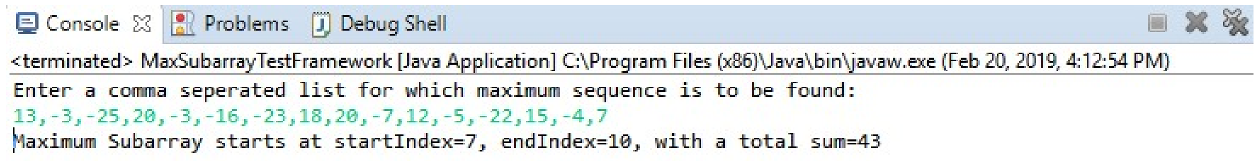
Console Output:
Consider the index to start from zero.



```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 20, 2019, 4:07:39 PM)
Enter a comma seperated list for which maximum sequence is to be found:
1,-4,3,-4
Maximum Subarray starts at startIndex=2, endIndex=2, with a total sum=3
```

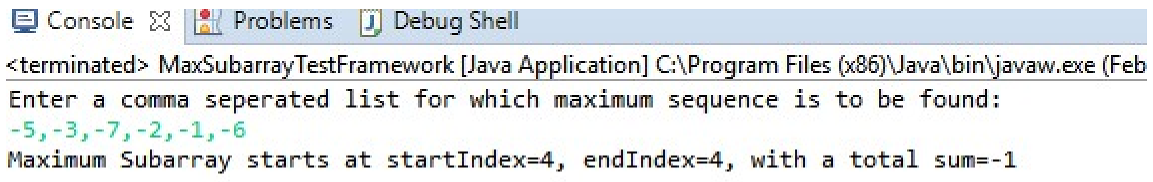


```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 20, 2019, 4:10:21 PM)
Enter a comma seperated list for which maximum sequence is to be found:
1,-4,3,4,-2,6
Maximum Subarray starts at startIndex=2, endIndex=5, with a total sum=11
```



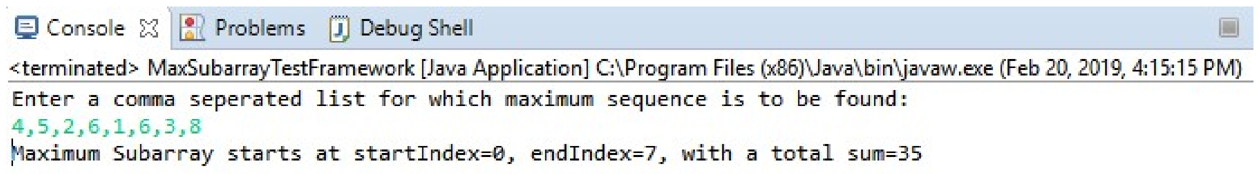
```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 20, 2019, 4:12:54 PM)
Enter a comma seperated list for which maximum sequence is to be found:
13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7
Maximum Subarray starts at startIndex=7, endIndex=10, with a total sum=43
```

When an all negative input is given

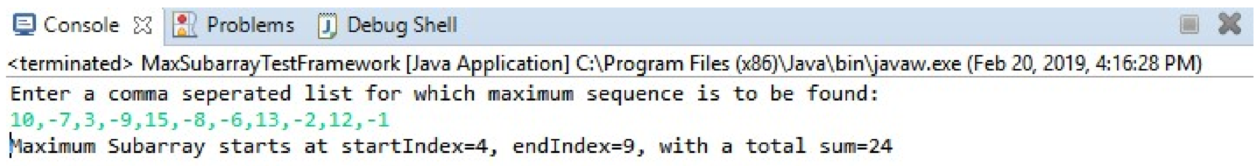


```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb
Enter a comma seperated list for which maximum sequence is to be found:
-5,-3,-7,-2,-1,-6
Maximum Subarray starts at startIndex=4, endIndex=4, with a total sum=-1
```

When an all positive input is given



```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 20, 2019, 4:15:15 PM)
Enter a comma seperated list for which maximum sequence is to be found:
4,5,2,6,1,6,3,8
Maximum Subarray starts at startIndex=0, endIndex=7, with a total sum=35
```



```
Console ⊠  Problems  Debug Shell
<terminated> MaxSubarrayTestFramework [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 20, 2019, 4:16:28 PM)
Enter a comma seperated list for which maximum sequence is to be found:
10,-7,3,-9,15,-8,-6,13,-2,12,-1
Maximum Subarray starts at startIndex=4, endIndex=9, with a total sum=24
```

Proof Of Correctness:

When number of elements is 1: If the array has only one element this is handled by the line 1 and 2 of FIND-MAXIMUM-SUBARRAY which then returns the element as maximum sum.

When more than 1 elements are entered: There are 3 possibilities as to where the maximum subarray lies.

It either lies **entirely on the left of side of mid**, in this case the algorithm returns maximum of left-sum, right-sum and cross-sum which for this case is the left-sum with indexes as left-low and left-high from line 8 of FIND-MAXIMUM-SUBARRAY.

It either lies **entirely on the right of side of mid**, in this case the algorithm returns maximum of left-sum, right-sum and cross-sum which for this case is the right-sum with indexes as right-low and right-high from line 10 of FIND-MAXIMUM-SUBARRAY.

It either lies **across the mid**, in this case the algorithm returns maximum of left-sum, right-sum and cross-sum which for this case has the start index to the left of mid and end index to the right of mid. The cross-sum is computed from the maximum left sum calculated from mid towards low, plus the maximum right sum calculated from mid towards high. For across the mid, we initialize the startIndex to mid and the left-sum to $-\infty$. For every iteration of the loop we shift to the left of mid adding each element to the sum variable. We continue this till the sum value increase, copying each sum to left-sum if left-sum is less than sum. There will be a point when the sum value reaches a peak an then starts to reduce. This peak value is maintained in the left-sum and the index upto which this sum is computed is maintained. Thus we get the maximum sum of the left side of mid. For the right side the endIndex is initialized to mid+1 and the right-sum to $-\infty$. Similar procedure is run to the right side of mid to calculate the maximum sum of the right side of mid.

Optional Question:

Can we design a divide-and-conquer solution with linear running time?

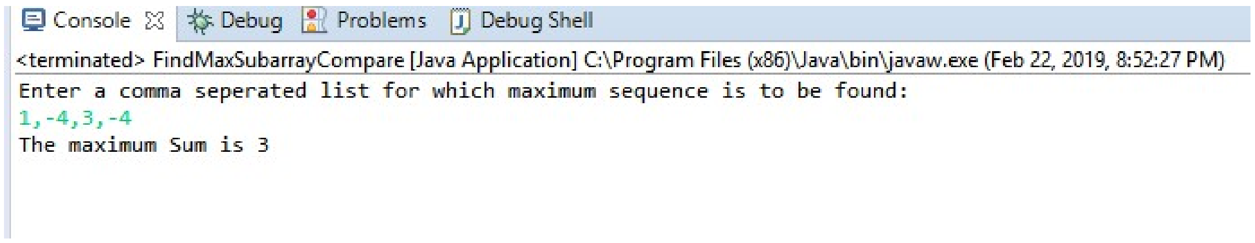
Yes, we can design a divide-and-conquer solution with linear running time to calculate the maximum sequence sum. Linear running time means the time complexity should be O(N). This can be achieved only when we can make the recurrence relation $T(n) = 2T(n/2) + O(1)$, a recursive algorithm that splits the input into 2 halves and does a constant amount of other work. By Case 1 of Master method this gives $N^{\log_2 2} = N^1 = N$ which is polynomially greater than $f(N) = 1$. Hence, $T(N) = \Theta(N)$.

To achieve this recurrence relation, we can use the below algorithm.

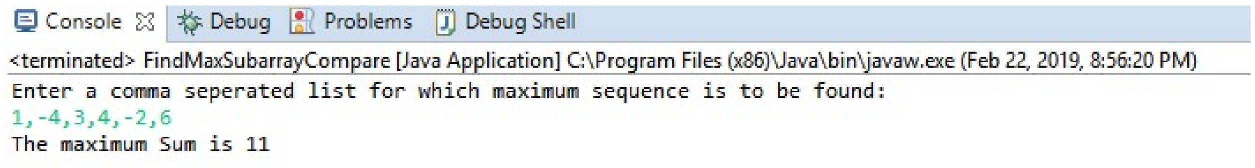
```
FMS-COMPARE(A, low, high)
  1.    if low = high then
  2.        return (A[low], A[low], A[low], A[low])
  3.    else
  4.        mid ← low + high / 2
  5.        Left ← FMS-COMPARE(A, low, mid)
  6.        Right ← FMS-COMPARE(A, mid + 1, high)
  7.        return COMPARE(A, Left, Right)
COMPARE(A, L, R)
  1. totalSum ← L.totalSum + R.totalSum
  2. maxPrefix ← MAX(L.maxPrefix, L.totalSum + R.maxPrefix)
  3. maxSuffix ← MAX(R.maxSuffix, R.totalSum + L.maxSuffix)
  4. maxSum ← MAX(L.maxSum, R.maxSum, L.maxSuffix + R.maxPrefix)
  5. return (totalSum, maxSum, maxPrefix, maxSuffix)
```

In this the indexes for the maximum crossing subarray case, are calculated from the previously computed maximum sums of left and right subarrays. That is the crossing maximum subarray may have the start index as that of the start index of the maximum left subarray (calculated in the previous recursion step) with maximum sum spanning across to the starting index of the right subarray. Similarly, the end index of the crossing maximum subarray may have the end index as that of the maximum right subarray (calculated in the previous recursion step) with the maximum sum spanning across to the end index of the left subarray of previous recursion step. Since these calculations are a constant number of comparisons and additions, they contribute to O(1) time which is what we aimed at achieving.

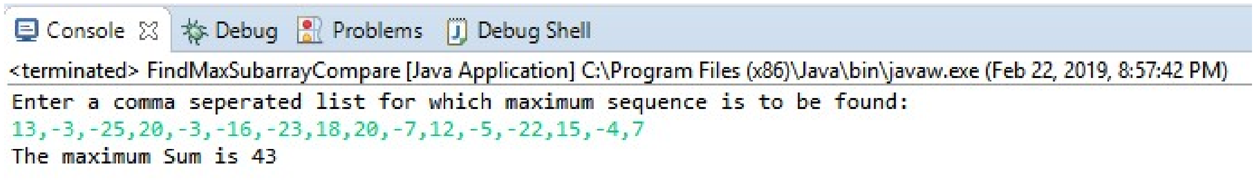
Console Output for Maximum subarray with divide and conquer O(n)



```
<terminated> FindMaxSubarrayCompare [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 22, 2019, 8:52:27 PM)
Enter a comma seperated list for which maximum sequence is to be found:
1,-4,3,-4
The maximum Sum is 3
```



```
<terminated> FindMaxSubarrayCompare [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 22, 2019, 8:56:20 PM)
Enter a comma seperated list for which maximum sequence is to be found:
1,-4,3,4,-2,6
The maximum Sum is 11
```



```
<terminated> FindMaxSubarrayCompare [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Feb 22, 2019, 8:57:42 PM)
Enter a comma seperated list for which maximum sequence is to be found:
13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7
The maximum Sum is 43
```

Proof of Correctness:
When there is only 1 element : The algorithm terminates returning the element as maximum sum.
For more than 1 element: Consider u to be a divided at the middle into two subarrays v and w. Then if the maximum subarray for u does not include the middle it will be the same as the maximum subarray of v. If the maximum subarray of u does include the middle it will start with the maximum subarray of v and continue through the middle to the maximum subarray of w.

Reference: Introduction to algorithms 3rd edition by Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. Research paper “Divide & Conquer strikes back: maximum-subarray in linear time” attached in canvas.