

Reursive

Pseudo code

Function maxSubArray(arr, start, end):

if start equals end:

return arr[start]

mid = (start + end) / 2

left = maxSubArray(arr, start, mid)

right = maxSubArray(arr, mid+1, end)

max= arr[mid]

sum = arr[mid]

for i from mid-1 down to start:

sum += arr[i]

if sum is greater than max:

max = sum

sum = max

for i from mid+1 up to end:

sum += arr[i]

if sum is greater than max:

max = sum

if left is greater than right and left is greater than max:

return leftMax

else if right is greater than left and right is greater than max:

return rightMax

else:

return max

Main program:

```

arr = {-2, -3, 4, -1, -2, 1, 5, -3}

n = size of arr

maxSum = maxSubArray(arr, 0, n-1)

print "Maximum contiguous sum is ", maxSum

```

Time complexity

$$T(n) = 2t(n/2) - 4n$$

$$T(n/2) = 2t(n/4) - 4n/2 \text{-----} T(n) = 2(2t(n/4) - 2n) - 4n$$

$$T(n/4) = 2t(n/8) - 4n/4 \text{-----} T(n) = 2(2(2t(n/8) - n)) - 4n$$

$$T(n/8) = 2t(n/16) - 4n/8 \text{-----} T(n) = 2(2(2(2t(n/16) - n/2)) - 4n$$

$$T(n) = 2^4 t(n/2^4) - 2^4 n$$

$$T(n) = 2^k t(n/2^k) - 2^k n$$

$$T(n) = 1$$

$$t(n/2^k) = 1$$

$$n/2^k = 1$$

$$K = \log n$$

$$T(n) = t(1) - 2^n \log n$$

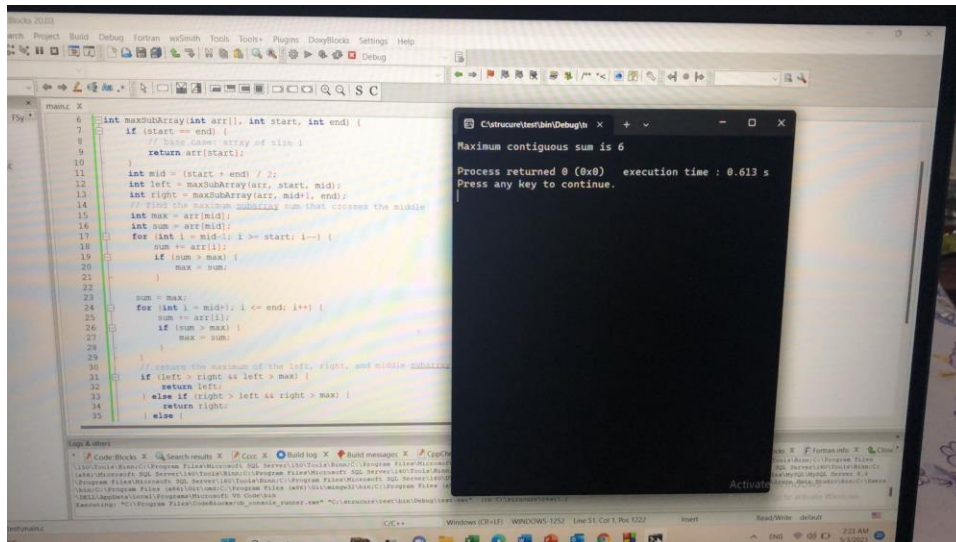
$$T(n) = O(n \log n)$$

Divided and conquer = $O(\log n)$

Maximum subArray = $O(n)$

Total $T(n) = O(n \log n)$

Output



Non-Reursive

Pseudo-code

Function maximal_subarray(n,arr)

```
{
    Max <- arr[0]
    For i<-0 to n
    {
        Sum <- arr[i]
        For j <- i+1 to n
        {
            Sum <- sum + arr[j]
            If max < sum
                Max = sum
        }
    }
```

If l equals n-1 and max < sum

 Max = sum

 }

 Return max

}

Main program:

 arr = {-2, -1, -3, 4, -1, 2, 1, -5, 4}

 n = size of arr

 If n is equal 0

 Print "Enter at least one item"

 else

 LargestSum = maximal_subarray(n, arr)

 print "Largest sum = ", LargestSum

Time Complexity

$$\begin{aligned}
 & \sum_{i=0}^n \sum_{j=i+1}^n 1 \\
 &= \sum_{i=0}^n [n - (i+1) + 1] \\
 &= \sum_{i=0}^n n - i - 1 + 1 \\
 &= \sum_{i=0}^n n - i \\
 &= \sum_{i=0}^n n - \sum_{i=0}^n i \\
 &= n \sum_{i=0}^n 1 - \sum_{i=0}^n i \\
 &= n(n-0+1) - \frac{n(n+1)}{2} \\
 &= n(n+1) - \frac{n}{2}(n+1) \\
 &= n^2 + n - \frac{1}{2}n^2 - \frac{1}{2}n \\
 &= \underline{O(n^2)} \quad O(n^2) \quad O(n^2)
 \end{aligned}$$

Output

The screenshot shows a C++ program in Code::Blocks. The program implements a function `maximal_subarray` that finds the maximum sum of a contiguous subarray. The main function initializes an array `arr[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4}` and calls the function. The output window shows the result: "largest sum = 6", "Process returned 0 (0x0)", "execution time : 0.677 s", and "Press any key to continue."

```

#include <stdio.h>

int maximal_subarray(int n, int arr[])
{
    int max = arr[0];
    for(int i=0; i<n; i++)
    {
        int sum = arr[i];
        for(int j= i+1; j<n; j++)
        {
            sum += arr[j];
            if(max < sum)
            {
                max = sum;
            }
        }
        if( (i== n-1) && (max<sum) )
        {
            max = sum;
        }
    }
    return max;
}

int main()
{
    int arr[] = {-2,1,-3,4,-1,2,1,-5,4};
    int n = sizeof(arr)/sizeof(arr[0]);
    if(n == 0)
    {
        printf("Enter at least one item");
    }
}
  
```

Output:

```

largest sum = 6
Process returned 0 (0x0)   execution time : 0.677 s
Press any key to continue.
  
```

Comparison between codes

	None-Recursive	Recursive
Best-Case	$Sqr(n)$	1
Average-Case	$Sqr(n)$	$n \log n$
Worst-Case	$Sqr(n)$	$n \log n$

Recursive is best solution

-> because complexity of None-Recursive is Quadratic and complexity of Recursive is log linear

-> algorithms with Quadratic time complexity are much slower than those with log-linear time complexity, especially for large input sizes.