Problem statement :-

* Given a one dimensional array that may contain both positive and negative numbers (integers), find the sum of contiguous subarray of numbers which has the largest sum and return its sum.

Example :- Input :- [-2, -5, 6, 2, -3, 1, 5, -6]

Output :- 7

Explanation :- [6, -2, -3, 1, 5] has the largest sum which is 7

Algorithm formulation / Idea :-

a) Divide the given array in two halves.

b) Return the maximum of following three

1) Recursively calculate the maximum subarray in left half

2) Recursively calculate the maximum subarray in right half

3) Recursively calculate the maximum subarray sum such that the subarray crosses the midpoint.

(i) Find the maximum sum starting from midpoint and ending at some point on left of mid.

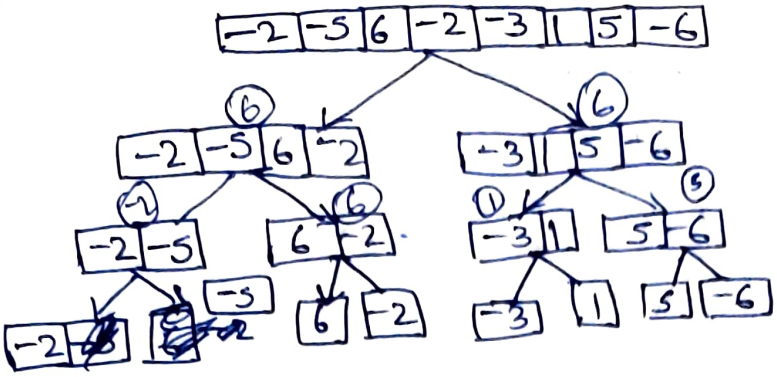
(ii) Find the maximum sum starting from mid+1 and ending with some point on right of mid+1

(iii) Finally combine the two and return.

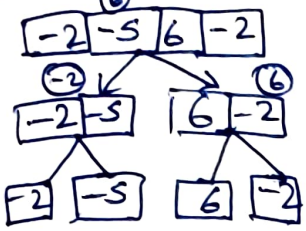
Pseudocode :-

```
Max-Subarray (A, left, Right)
    if (right == left)
        return (left, right, A[left])
    else mid = [(left + right)/2]
    l1 = Find-max_subarray (A, left, mid)
    r1 = Find-max_subarray (A, mid+1, right)
    m1 = Find-max-crossing-subarray (A, left, mid, right)
    if sum(l1) > sum(r1) and sum(l1) > sum(m1)
    return l1
    else if sum(r1) > sum(l1) and sum(r1) > sum(m1)
    return r1
    Else return m1
```

simulation / Illustration:-



Left part:-



Calculations:-
pivot element = -2
left sum = -2
right sum = -5
Cross sum Calculations:-
Left sub sum = -2
right sub sum = -5
Cross sum = -2-5 = -7

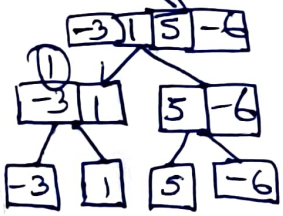
Calculations
pivot element = -5
Left sum = -2
right sum = 6
Cross sum calculations:
Left sub sum = max(-5, -5-2) = -5
right sub sum = max(6, 6-2) = 6
cross sum = -5+6 = 1

→ total array part

max sum = max(left sum, right sum, cross sum) = -2

max sum = max(left sum, right sum, cross sum) = 6

Right part:-



Calculations:-
pivot element = -3
Left sum = -3
right sum = 1
Cross sum calculations:-
Left sub sum = -3
right sub sum = 1
Cross sum = -3+1 = -2

Calculations
pivot element = 5
Left sum = 5
right sum = -6
cross sum calculation:-
Left sub sum = 5
right sub sum = -6
cross sum = 5-6 = -1

max sum = max(Left sum, right sum, cross sum) = 1

max sum = max(left sum, right sum, cross sum) = 5

Left part
Calculations:- pivot element = 6
left sum = 6
right sum = 6-2
cross calculation:-
left sub sum = max(-5, 5-2) = 6
right sub sum = -2
Cross sum = 6-2 = 4
max sum = max(left sum, right sum, cross sum)
= 6

Right part:-
Calculations:- pivot = 1
Left sum = 1
right sum = 5
Cross sum calculation:-
Left sub sum = max(1, 1-3) = 1
right sub sum = max(5, 5-6) = 5
cross sum = 1+5 = 6
max sum = max(left sum, right sum, cross sum)
= 6

Cross sum calculation:-

-2	-5	6	-2	-3	1	5	-6

pivot element = -2

Left sub sum = max(-2, -2+6, -2+6-5, -2+6-5-2)

= max(-2, 4, -1, -3)

= 4

right sub sum = max(-3, -3+1, -3+1+5, -3+1+5-6)

= max(-3, -2, 3, -3)

= 3

Cross sum = 4+3 = 7

max sum = max(left sum, right sum, cross sum)

= 7

So, the maximum subarray sum is 7.

Time complexity analysis:-

Find max-cross subarray takes : O(n) times

two recursive calls on input size n/2 takes : 2T(n/2) time

Hence: T(n) = 2T(n/2) + O(n)

Here,

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

$= 2[2T(n/2^2) + n/2]$

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

$= 2^2 [2T(n/2^3) + n/2^2] + 2n$

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

$T(n) = 2^k T(n/2^k) + kn$

Use case / variations:-

→ Basis of efficient algorithms such as quick sort, merge sort

→ Multiplying large number (e.g. the karatsuba algorithm)

→ Finding the closest pair of points

→ Syntactic analysis (e.g. top down parsers)

→ Computing the discrete fourier transform (FFT)

Alternative solutions of solving:-

* Brute force solution * Greedy algorithm * Kadane's algorithm