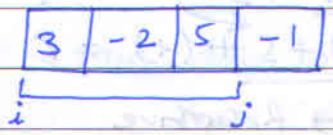
Maximum Subarray Problem

(Divid and Conquer approach)

- A nonempty contiguous subarray of A whose values have the largest sum.
- A → 

ans - 7

- If array A contain all positive - Whole array sum is max
- If array A contain all negative - Single element (having less magnitude) is max subarray.

Brute force approach

```
// possible size of subarray   for( subarraysize = 1 ; subarraysize <= n ; subarraysize ++)
                               {
// index position                for(startindex = 0; startindex <= n-1 ; startindex ++)
                                 {
                                   if( startindex + subarraysize > n )
                                       break;
// sum of subsequence               for( i = startindex ; i < (startindex + subarray ); i++)
                                     { sum += arr[i]
                                       ans = max( ans, sum)
                                     }
                                 } return answer;
                               }
```

Complexity = $\Theta(n^3)$

Improvement 1) Changing the order
2) Use the previously calculated values.

- Sum of size k = (Sum of (k-1)) + k.

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

Sum

3 = 3

3 + (-2) = 1

3 + (-2) + 5 = 7 — Max Sum

3 + (-2) + 5 + (-1) = 6

↳ Recursive

Steps

1. Start index position
2. Subarray Size (here we will use previous Calculated Subarray Sum)
3. Check exceed array bound.
4. Calculate Sum = arr [start_index + Subarray Size - 1]

   ans = max (Sum, ans)

Divide and Conquer A[low ... high]

- Divide and Conquer suggests we divide an array into two subarray of equal size as possible.

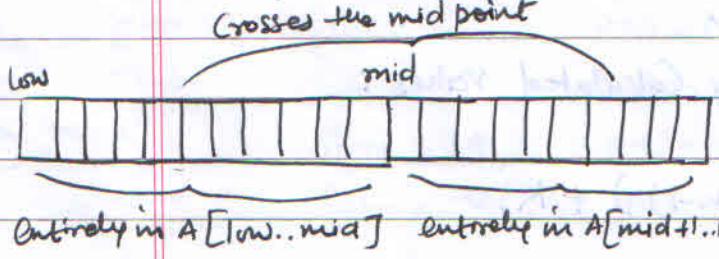
- Uses mid = (low + high)/2, and divide into A[low...mid] and A[mid+1 ... high].

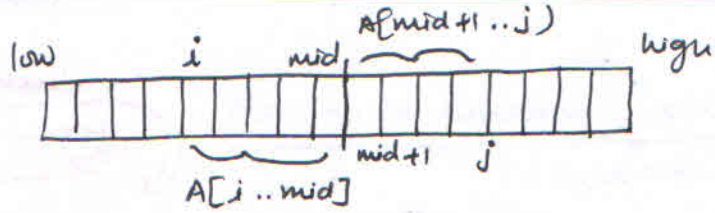
- Any Contigous Subarray (max Subarray also) A[i...j] of A[low .. high] must lie in the following places.

1. Entirely in Subarray A[low..mid] so that low ≤ i ≤ j ≤ mid.

2. Entirely in Subarray A[mid+1 .. high] so that mid < i ≤ j ≤ high.

3. Crossing the midpoint so that low ≤ i ≤ mid < j ≤ high.





classmate
Date
Page

Find-Max-Crossing-Subarray (A, low, mid, high)

1. left-Sum = $-\infty$
2. Sum = 0
3. for i = mid down to low
4. Sum = Sum + A[i]
5. if (Sum > left-Sum)
6. leftSum = Sum
7. max-left = i
8. right-Sum = $-\infty$
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)

- Complexity is - $\Theta(n)$

Find-Maximum-Subarray (A, low, high) (FMS(A, low, high)

1. if high == low
2. return (low, high, A[low] // base case only one element
3. Else mid = (low + high)/2
4. (left-low, left-high, left-sum) = FMS(A, low, mid)
5. (right-low, right-high, right-Sum) = fms(A, mid+1, high)
6. (Cross-low, Cross high, Cross-Sum) = FMCS(A, low, mid, high)
7. if (left-Sum ≥ right-Sum ≥ Cross-Sum
8. return (left-low, left-high, left Sum)
9. Else if (right-Sum ≥ left-Sum and right-sum ≥ Cross-Sum
10. return (right-low, right-high, right-Sum)
11. Else return (Cross-low, Cross-high, Cross Sum)

Analysis

- When $n = 1$ $\quad T(1) = \Theta(1)$ $\quad$ Base Case

- We divide original problem in 2 parts and of half size
  So $= 2T(n/2)$

- finding max crossing subarray takes $= T(n) = \Theta(n)$

- line 7 to 11 takes $T(n) = \Theta(1)$

$$T(n) = \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1)$$
$$= 2T(n/2) + \Theta(n)$$

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

$T(n) = \Theta(n \log n)$ $\quad$ same as merge sort.

Kadane's Algo (linear approach) (Atleast one positive value)

1	-3	2	-5	7	6	-1	-4	11	-23

- for +ve Sum will keep in the array Sum
- otherwise reset for -ve value to Sum = 0

1 → Sum = 1, ans = 1
-3 → Sum = -2/0, ans = 1
2 → Sum = 2, ans = 2
-5 → Sum = -3/0, ans = 2
7 → Sum = 7, ans = 7
6 → Sum = 13, ans = 13
-1 → Sum = 12, ans = 13
-4 → Sum = 8, ans = 13
11 → Sum = 19, ans = 19
-23 → Sum = -4/0, ans = 19

$T(n) = \Theta(n)$

ans = 19