Binary Search

Divide. – Split it in half,
Conquer – Solve one of the halves
Combine – trivial

Maximum Subarray Problem

Input: An array A of numbers, both positive and negative.
Output: A subarray of A w/ biggest sum.

(subarray: contiguous region of A; w/: with)

Ex: A = −5, −6, 3, −2, 7, −6, 2

−5, −6 → −8
3, −2, 7, −6, 2 → 8

Algorithm: Consider all possible subarrays.

"Brute Force"

~~Final~~

```
for i = 1 to n
    for j = i+1 to n.
        sum = Sum(A[i..j])
        ~~if ...~~
return biggest sum we have seen.
```

$\Theta(n^2)$ — $\Theta(n)$ ~~for~~ j-i+1

$\underline{\Theta(n^3)}$

Can be improved to $\Theta(n^2)$

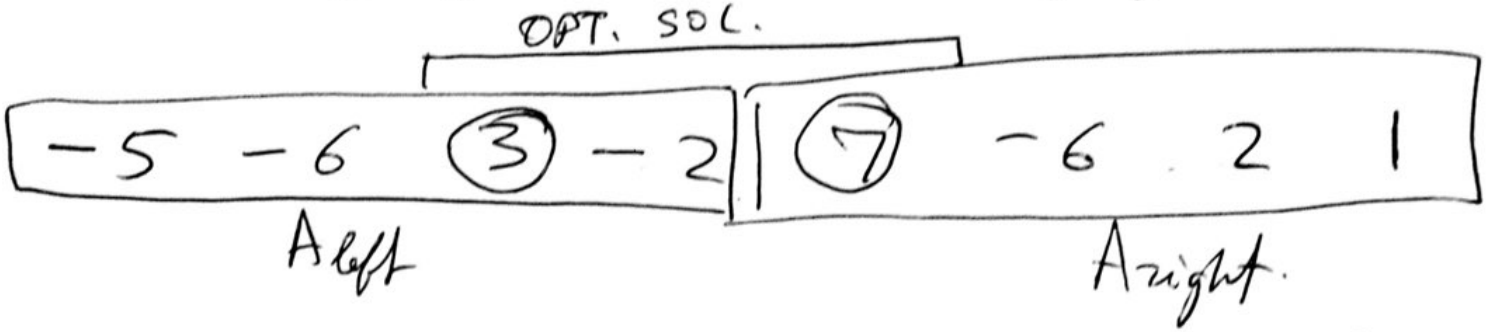
Divide and Conquer approach.

Divide: Split problem into two subproblems.

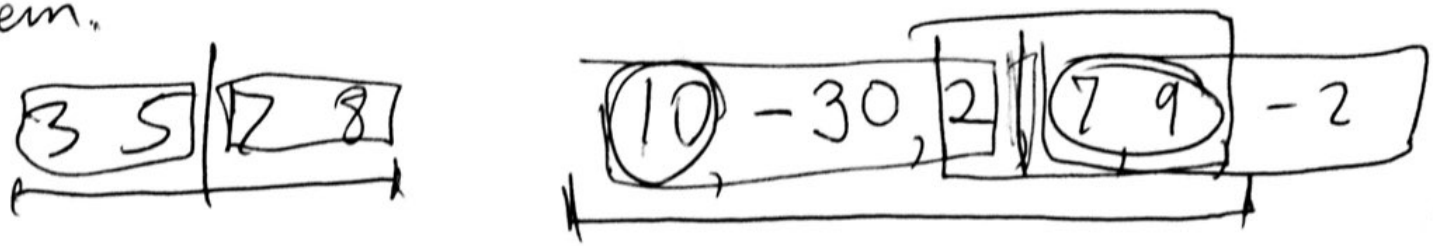
A_{left}, A_{right}

↖ two halves

Conquer: Find Max-subarray of A_{left} and Max-Subarray of A_{right}.



Idea: Consider the subarray that consists of the two sub-solutions and everything between them.



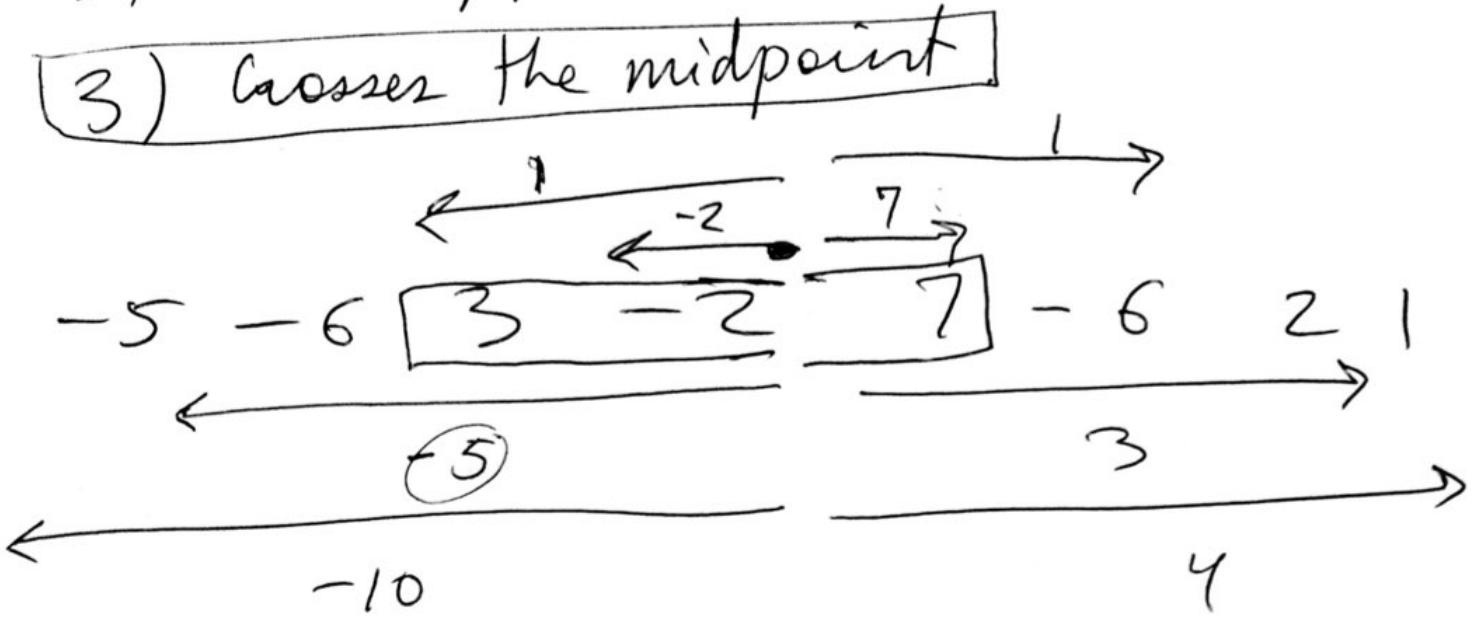
Opt. sol: either

1) in A_{left}

2) in A_{right}

$\Theta(n)$ time?

3) Crosses the midpoint



Find Max Left Subarray (A, mid).

running sum = 0

$\Theta(n)$ for i = mid down to 0.

$\Theta(1)$ runningSum += A[i].

return Max running sum found.

Find Max Subarray (A, i, j)

$mid = \lfloor \frac{i+j}{2} \rfloor$

| Find Max Subbarray (A, 1, mid).

| Find Max Subarray (A, mid+1, j)

| Find Max Subarray Crossing Midpoint (A, mid).

Take best of these three options. // $\Theta(1)$

Running Time: recurrence relation

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) \qquad T(n) = \Theta(n \log n)$$

↑ runtime for inputs of size n