Divide and Conquer Paradigm

[Recap]

1. Divide into smaller subproblems
2. Conquer via recursive calls
3. Combine solutions of subproblems.

The Maximum Subarray Problem

[Reference: Coreman book]
Section 4.1

Given the daily stock prices of some company. You are allowed to buy one unit of stock only one time and then sell it at a later date. Our goal is to maximize the profit.

Example ①

Day	0	1	2	3	4
Price	10	11	7	10	6

Sol: Buy after day-2 and sell after day 3

Profit $= 10 - 7 = 3$.

Brute-force Solution:

Try every possible pair of buy and sell dates in which the buy date precedes the sell date.

Total we have $\binom{n}{2}$ pairs, that is, we have to spend $\Omega(n^2)$ time.

Q: Can we do better? ie, Algorithm with running time $o(n^2)$.

Previous example

Day	0	1	2	3	4	5	6	7	8
Price	10	11	7	10	4	9	8	6	12
Change	–	1	-4	3	-6	5	-1	-2	6

Instead of looking at the daily prices, we look at the daily change in price, where the change on day i is the difference between the prices after day $i-1$ and after day i

We use A to denote the change array

"Then the Problem reduces to find a non-empty, contiguous subarray of A whose sum of values is maximum" — Maximum Subarray Problem.

Observe that if the array A contains only nonnegative entries, then the problem is easy, the entire array would give the greatest sum.

Brute-force Solution

```
MAXIMUM-SUBARRAY (A, low, high)
    max sum = -∞
    for i = low to high
        curr sum = 0
        for j = i to high
            curr sum = curr sum + A[j]
            if curr sum > max sum
                max sum = curr sum
```

Time Complexity $= O(n^2)$

where n is the number of elements in A.

A Solution using Divide-and-Conquer

We want to find a maximum subarray of the array $A[low, ..., high]$.

We divide the array into two subarrays of as equal size as possible.

Let mid denotes the midpoint of the array. We consider two subarrays $A[low..mid]$ and $A[mid+1, ..., high]$

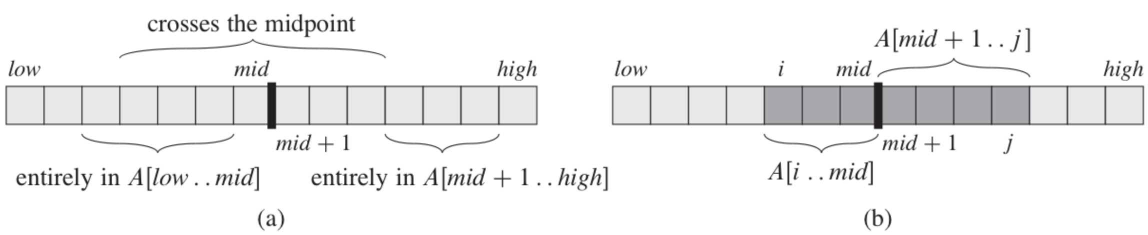
Then any contiguous subarray $A[i..j]$ of $A[low, .. high]$ must lie in exactly one of the following places.

① Entirely in $A[low..mid]$, that is $low \le i \le j \le mid$

② Entirely in $A[mid+1, ..high]$, that is $mid < i \le j \le high$

③ Crossing the midpoint, that is $low \le i \le mid < j \le high$

Therefore, a maximum subarray of A[low,..high] must lie in exactly one of these places.



We can find maximum subarrays of A[low..mid] and A[mid+1,...high] recursively.

So we only have to find a maximum subarray that crosses the midpoint, then we take a subarray with largest sum of the three.

We now describe a linear time algorithm to find maximum subarray crossing the midpoint.

(Note that this is NOT a smaller instance of our original Problem)

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

Total number of iterations $(mid - low + 1) + (high - mid)$

$= n$ $\quad (n : \text{size of } A)$

$\therefore$ FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high) takes $\Theta(n)$ time.

Next we describe a divide and Conquer algorithm to Solve the maximum subarray Problem.

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])            // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

Running time analysis :

$T(n)$: denote the running time of FIND-MAXIMUM-SUBARRAY (A, low, high) on a subarray of n-elements.

If $n=1$ (base case) ; $T(1) = 1 = \Theta(1)$

We get the following recurrence

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

$\therefore \quad T(n) = \Theta(n \log n)$ [asymptotically faster than Brute-force]

In Dynamic Programming topic, we study Kadane's algorithm having $O(n)$ run time, for maximum-subarray Problem.