1.5em 0pt

Divide and Conquer Method

Dr. Raghunath Reddy M

Dept. of Computer Science Engineering,
Bennett University, Greater Noida

January 28, 2020

The Maximum Sub-array Sum Problem

Problem Definition

Input: Let A be an array of n elements, namely, $A[1], A[2], \ldots, A[n]$.
Output: Find a sub-array $A[i..j]$ i.e., $A[i], A[i+1], \ldots, A[j]$ such that the sum of the sub-array is maximum.

10	*-12*	*14*	*-10*	*12*	*14*	*10*	*-7*	*10*	*-15*	*20*	*-5*

Figure: An instance of array A.

The Maximum Subarray Sum Problem

Problem Definition

Input: Let A be an array of n elements, namely, $A[1], A[2], \ldots, A[n]$.
Output: Find a sub-array $A[i..j]$ i.e., $A[i], A[i+1], \ldots, A[j]$ such that the sum of the sub-array is maximum.

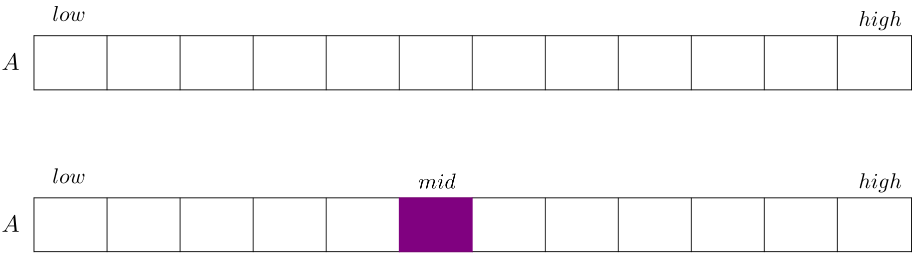
10	-12	14	-10	12	14	10	-7	10	-15	20	-5

Figure: An instance of array A and the colored sub-array is the maximum sum sub-array for the given instance.

Question ?

How to obtain the solution in $O(n \log n)$-time ?

Divide and Conquer Approach



Where can the optimal solution lie ?

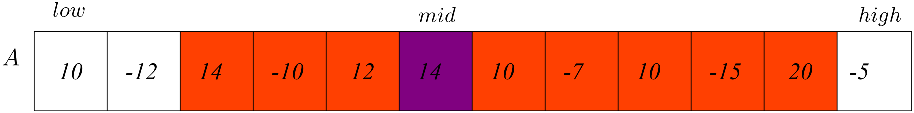
1. Can lie only in the left half $A[low..mid]$
2. Can lie only in the right half $A[mid + 1..high]$
3. Can cross the mid location of the array.

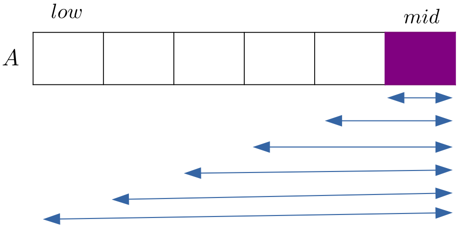
Special case: The optimal solution crosses the mid

Special case: The optimal solution crosses the mid

	low					mid						$high$
A	10	-12	14	-10	12	14	10	-7	10	-15	20	-5

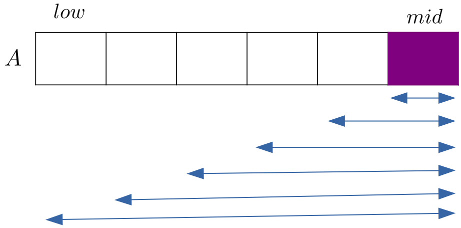
Special case: The optimal solution crosses the mid





Special case: The optimal solution crosses the mid

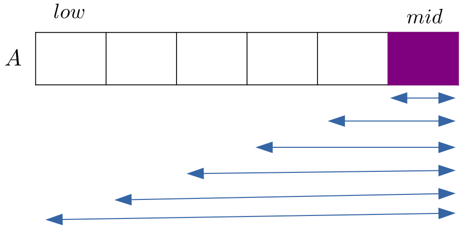
	low					*mid*						*high*
A	10	-12	14	-10	12	14	10	-7	10	-15	20	-5



$left\_sum = -\infty$
$sum = 0$
for $i = mid$ down to low **do**
    $sum = sum + A[i]$
    if $sum > left\_sum$ **then**
        $left\_sum = sum$
        $left\_max = i$
    end if
end for

Special case: The optimal solution crosses the mid

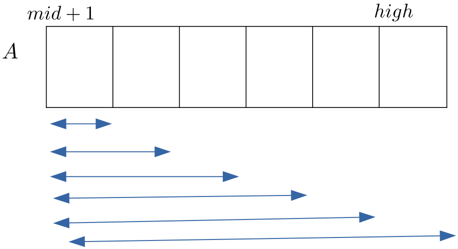
	low					mid						high
A	10	-12	14	-10	12	14	10	-7	10	-15	20	-5



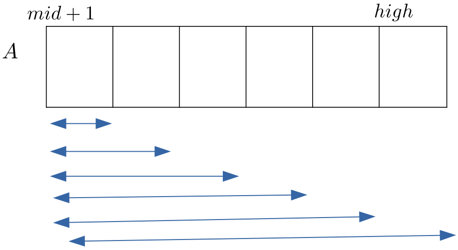

```
left_sum = −∞
sum = 0
for i = mid down to low do
    sum = sum + A[i]
    if sum > left_sum then
        left_sum = sum
        left_max = i
    end if
end for
```

- Similarly, we can extend the solution on the right side of *mid*.

Finish the code for the right of mid !!!!



Finish the code for the right of mid !!!!



$right\_sum = -\infty$
$sum = 0$
for $i = mid + 1$ to $high$ **do**
  $sum = sum + A[i]$
  if $sum > right\_sum$ **then**
    $right\_sum = sum$
    $right\_max = i$
  end if
end for

The complete algorithm for case (3)

In the next slide ...

Algorithm 1 FIND_MAX_CROSSING_SUBARRAY($A, low, mid, high$)

```
left_sum = −∞
sum = 0
for  i = mid down to low do
    sum = sum + A[i]
    if sum > left_sum  then
        left_sum = sum
        left_max = i
    end if
end for
right_sum = −∞
sum = 0
for  i = mid + 1 to high do
    sum = sum + A[i]
    if sum > right_sum  then
        right_sum = sum
        right_max = i
    end if
end for
return (left_max, right_max, left_sum + right_sum)
```

Algorithm 1 FIND-MAX-CROSSING-SUBARRAY(*A*, *low*, *mid*, *high*)

```
left_sum = −∞
sum = 0
for i = mid down to low do
    if sum > left_sum then
        left_sum = sum                O(1)    O(mid − low + 1)
        left_max = i
    end if
end for
right_sum = −∞
sum = 0
for i = mid + 1 to high do
    if sum > right_sum then
        right_sum = sum               O(1)    O(high − mid)
        right_max = i
    end if
end for
return (left_max, right_max, left_sum + right_sum)
```

Algorithm 1 FIND-MAX-CROSSING-SUBARRAY(A, low, mid, $high$)

```
left_sum = −∞
sum = 0
for i = mid down to low do
    if sum > left_sum then
        left_sum = sum            O(1)    O(mid − low + 1)
        left_max = i
    end if
end for
right_sum = −∞
sum = 0
for i = mid + 1 to high do
    if sum > right_sum then
        right_sum = sum           O(1)    O(high − mid)
        right_max = i
    end if
end for
return (left_max, right_max, left_sum + right_sum)
```

- Total time = $O(mid - low + 1) + O(high - mid) = O(high - low + 1) = O(n)$

General case:

Algorithm 2 *FIND_MAXIMUM_SUBARRAY*$(A, low, high)$

if $high == low$ **then**
    return $(low, high, A[low])$ //base case
else $mid = \lfloor (low + high)/2 \rfloor$
    $(llow, lhigh, lsum) = FIND\_MAXIMUM\_SUBARRAY(A, low, mid)$

    $(rlow, rhigh, rsum) = FIND\_MAXIMUM\_SUBARRAY(A, mid + 1, high)$

    $(clow, chigh, csum) = FIND\_MAX\_CROSSING\_SUBARRAY(A, low, mid, hig$

    if $lsum \geq rsum$ and $lsum \geq csum$ **then return** $(llow, lhigh, lsum)$

    else if $rsum \geq lsum$ and $rsum \geq csum$ **then return** $(rlow, rhigh, rsum)$

    else **return** $(clow, chigh, csum)$

    end if
end if

General case:

- Total time $T(n) = 2T(n/2) + O(n) = O(n \log n)$

The maximum sub-array Problem

Problem Definition

Input: Let A be an array of n elements, namely, $A[1], A[2], \ldots, A[n]$.
Output: Find a sub-array two indices i and j of A such that (i) $i < j$ and (ii) $A[j] - A[i]$ is the maximum.

85	*105*	*102*	*86*	*63*	*81*	*101*	*94*	*106*	*101*	*79*	*94*

Figure: An instance of the maximum sub-array problem.

Reduction to The maximum sub-array sum problem

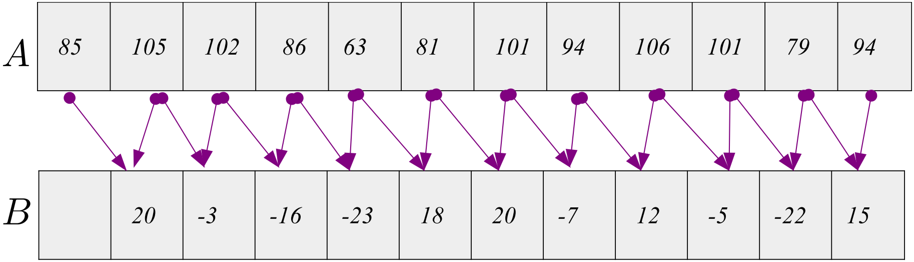


Figure: $B[i] = A[i] - A[i-1]$ for all $i \geq 2$ and $B[1] = NULL$.

- A is an instance of the maximum sub-array problem.
- B is an instance of the maximum sub-array sum problem.

Relation between the solutions of A and B

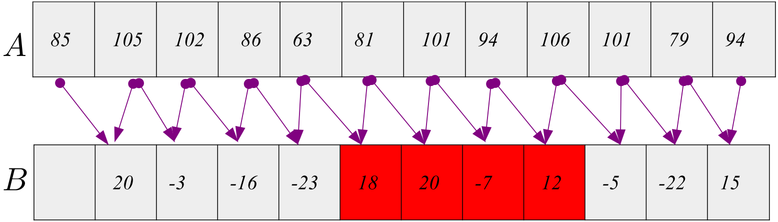


Figure: The red colored sub-array of B is the maximum sum sub-array.

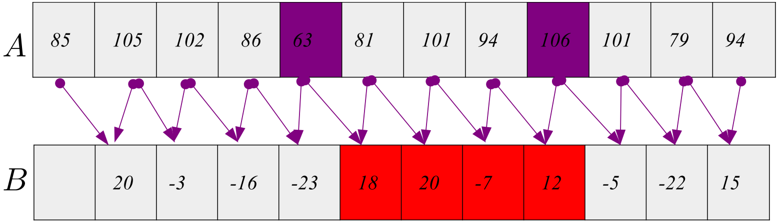


Figure: The colored cells of A denote the solution to the problem.

Finding the Closest Pair of Points in the Plane

Closest pair of points

Problem Definition

Input: Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane where each $p_i = (x_i, y_i)$ for real numbers x_i and y_i.
Output: Find a pair of points p_i and p_j such that the Euclidean distance between p and p_j, $d(p_i, p_j)$ is minimum.

Brute-force algorithm

Check all possible ${}^nC_2$ pairs and pick the best one.
Time : $O(n^2)$

Question

Can we do it in better time, like $O(n \log n)$-time ?