Maximum Subarray problem

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

Naive Approach

Step 1: Read number of elements, n

Step 2: Read 'n' elements in the array

Step 3: Let max = minimum negative value

Step 4: for sub_array_size 1 to n repeat step 5 to 7

Step 5: Generate sub_array of size sub_array_size repeat step 6 and 7

Step 6: Compute sum_of_sub_array generated

Step 7: If sum_of_sub_array > max then make max as sum_of_sub_array

Naive Approach

Complexity – $O(n^3)$

Fine Tuned Naive Approach

- Idea : We are not interested in the order of the elements but only sum
- Traverse array once when you are in ith element find all possible sum of sub arrays that start at index 'i' and formed by including elements at index 'j' >= 'i'

Fine Tuned Naive Approach

Step 1: Read number of elements, n

Step 2: Read 'n' elements in the array

Step 3: Let max = minimum negative value

Step 4: for i in range 1 to n repeat step 5 to 8

Step 5: Let subarray_sum = 0

Step 6: for j in range i to n repeat step 7 and 8

Step 7: subarray_sum += element[j]

Step 8: If subarray_sum > max then make max as sum_of_sub_array

Naive Approach

Complexity – $O(n^2)$

Divide and Conquer Approach

- Idea : any contiguous subarray A[i .. j] of A[low..high] must lie in exactly one of the following places:
- entirely in the subarray A[low..mid], so that low $\leq$ i $\leq$ j $\leq$ mid,
- entirely in the subarray A[mid + 1..high], so that mid < i $\leq$ j $\leq$ high, or
- crossing the midpoint, so that low $\leq$ i $\leq$ mid < j $\leq$ high.

Divide and Conquer Approach

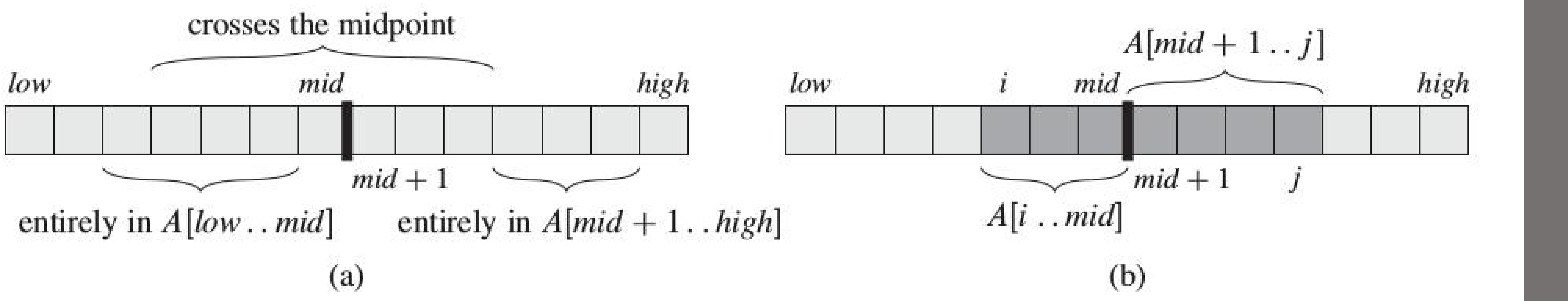

Figure 4.4 **(a)** Possible locations of subarrays of $A[low \, .. \, high]$: entirely in $A[low \, .. \, mid]$, entirely in $A[mid + 1 \, .. \, high]$, or crossing the midpoint *mid*. **(b)** Any subarray of $A[low \, .. \, high]$ crossing the midpoint comprises two subarrays $A[i \, .. \, mid]$ and $A[mid + 1 \, .. \, j]$, where $low \leq i \leq mid$ and $mid < j \leq high$.

Divide and Conquer Approach

```
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)
```

Divide and Conquer Approach

Complexity – $O(n \log n)$

Linear Complexity Approach (Kanden's Algorithm)

- Idea : If array contains only postive numbers then maximum sub array is the array itself
- Negative values inbits the total sum of array
- Negative values can also be part of maximum subarray provided elements around it gives a postive weightage

Linear Complexity Approach (Kanden's Algorithm)

Step 1: Read number of elements, n

Step 2: Read 'n' elements in the array

Step 3: Let global_max = first element of array, positive_sum_till_here = 0

Step 4: for i in range 1 to n repeat step 5 to 8

Step 5: positive_sum_till_here += elements[i]

Step 6: if positive_sum_till_here > global_max then global_max = positive_sum_till_here

Step 7: if positive_sum_till_here<0 then positive_sum_till_here = 0

Step 8: If subarray_sum > max then make max as sum_of_sub_array

Linear Complexity Approach (Kanden's Algorithm)

Complexity – O(n)