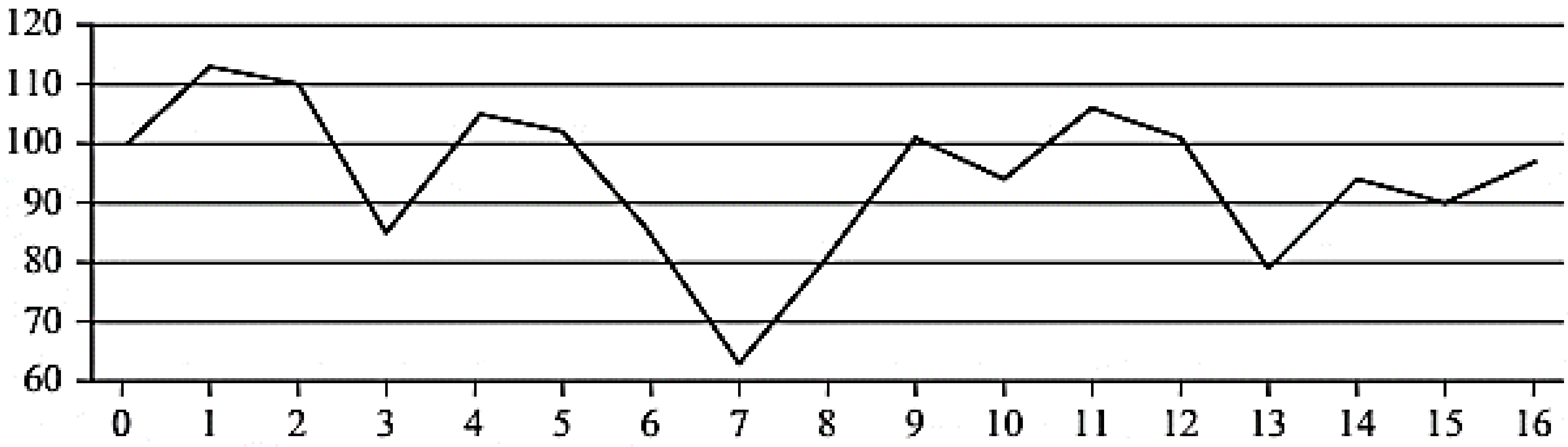
CS 412: The Maximum Subarray Problem

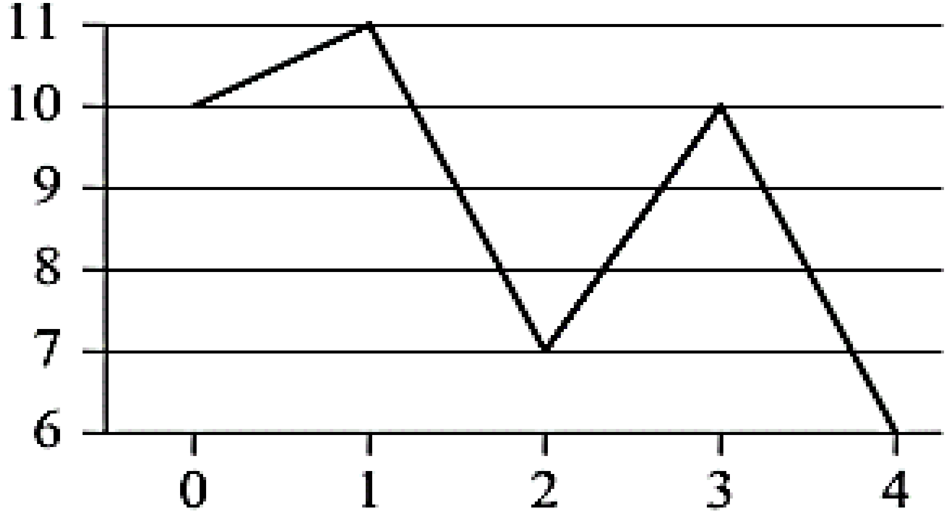
Shah Jamal Alam

Source: ***CLRS***

Stock Price of a Company



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7



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

Figure 4.2 An example showing that the maximum profit does not always start at the lowest price or end at the highest price. Again, the horizontal axis indicates the day, and the vertical axis shows the price. Here, the maximum profit of \$3 per share would be earned by buying after day 2 and selling after day 3. The price of \$7 after day 2 is not the lowest price overall, and the price of \$10 after day 3 is not the highest price overall.

The Maximum Subarray Problem

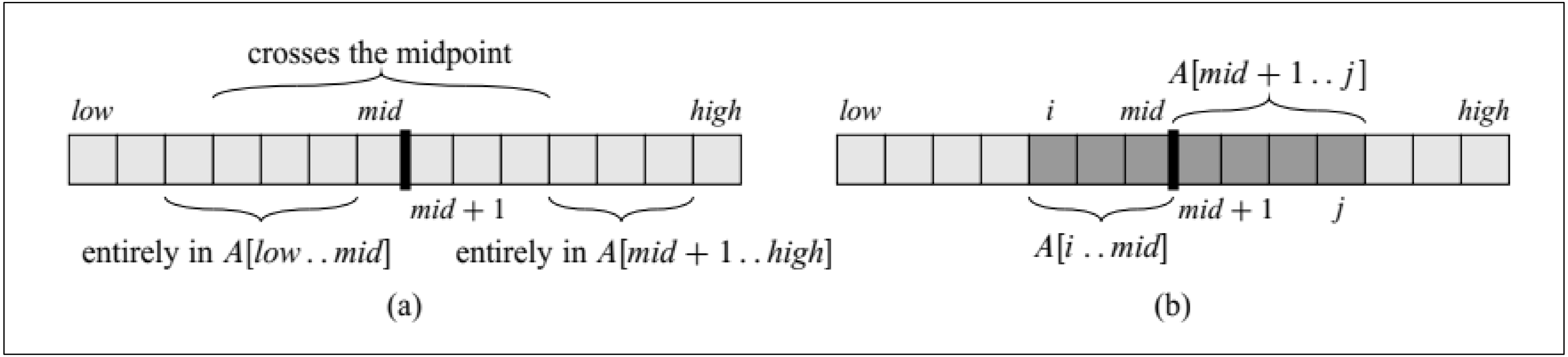
- The **brute-force** version takes $\Theta(n^3)$ and with exploiting the property that the sum of subarray $A[i \dots j]$ is $A[i \dots j-1] + A[j]$, we achieved a runtime of $\Theta(n^2)$ (see Bentley 1984).
- Remember that there are $\sum_{i=1}^{n} i = \frac{\mathrm{n(n+1)}}{2}$ subarrays.
- Can we do better? Or in other words, is there an algorithm that has a runtime $\boldsymbol{o(n^2)}$ [recall little-oh].
- The maximum subarray problem is nontrivial when there are **negative values**!
- **We use a divide-and-conquer approach.**

The Divide-and-Conquer approach

- Given an array $A[low \dots high]$.
- Divide the array A into two halves (of equal sizes, or nearly).
- Hence, the cost of divide $\boldsymbol{D(n)} = \Theta(\mathbf{1})$ $[mid\ =\ (low\ +\ high)$ / 2]
- Given that we now have two subarrays

$$A[low \dots mid] \text{ and } A[mid + 1 \dots high]$$

- **Any [including the maximum]** contiguous subarray $\boldsymbol{A[i \dots j]}$ of $\boldsymbol{A}$ must lie in one of the following three locations:
  - Entirely in the subarray $A[low \dots mid]$ so that $low \le i \le j \le mid$
  - Entirely in the subarray $A[mid + 1 \dots high]$ so that $mid < i \le j \le high$
  - Crossing the mid so that $low \le i \le mid < j \le high$



Any [including the maximum] contiguous subarray $\boldsymbol{A[i \ldots j]}$ of $\boldsymbol{A}$ must lie in one of the following three locations:

- Entirely in the subarray $A[low \ldots mid]$ so that $low \leq i \leq j \leq mid$
- Entirely in the subarray $A[mid + 1 \ldots high]$ so that $mid < i \leq j \leq high$
- Crossing the mid so that $low \leq i \leq mid < j \leq high$

```
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)
```

```
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)
```