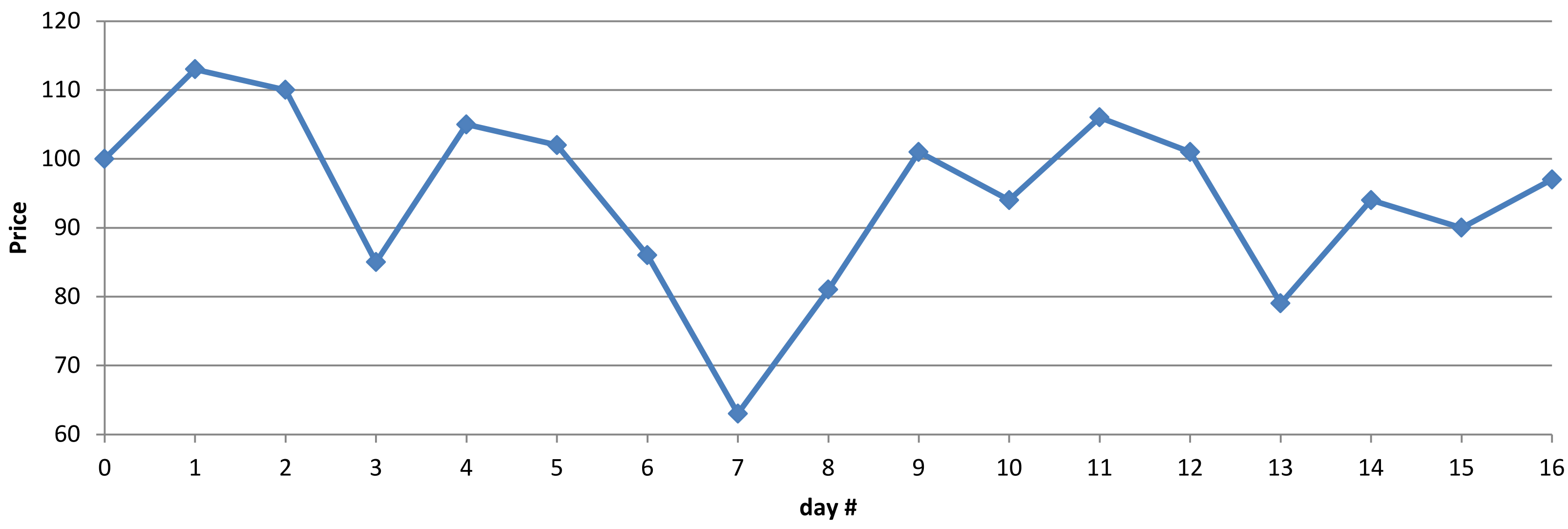
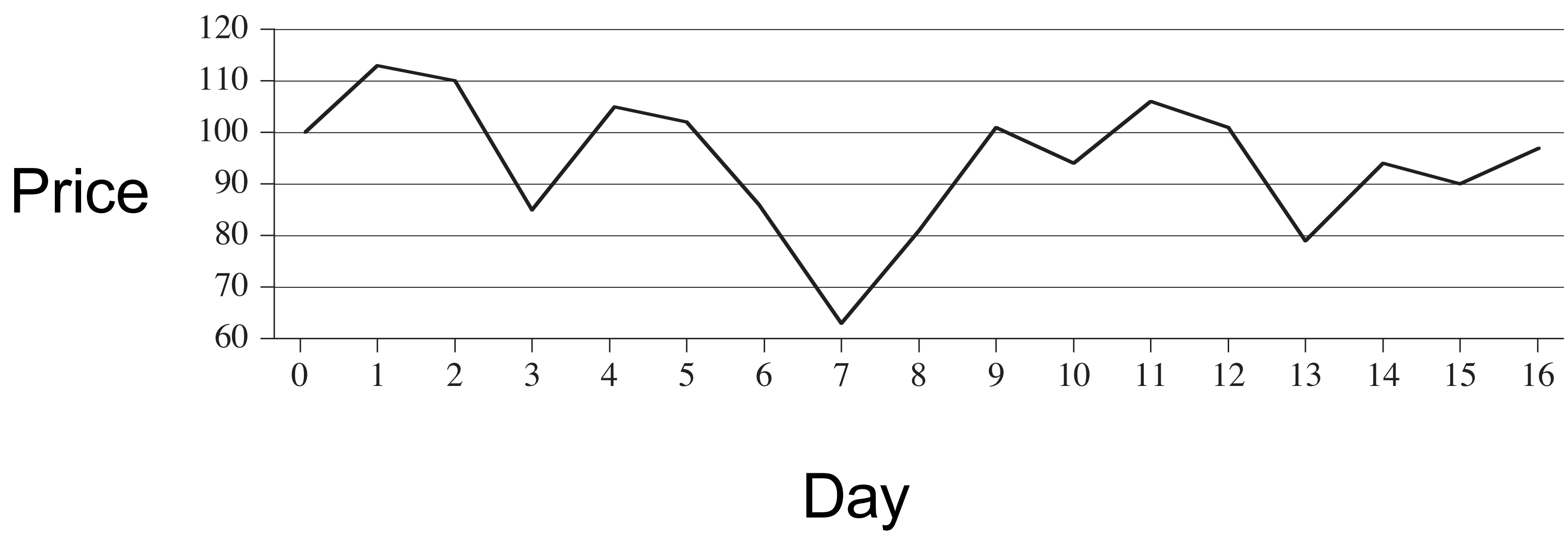
Maximum-subarray problem (Another Divide & Conquer problem)

- If you know the price of certain stock from day *i* to day *j*;
- You can only buy and sell one share once
- How to maximize your profit?

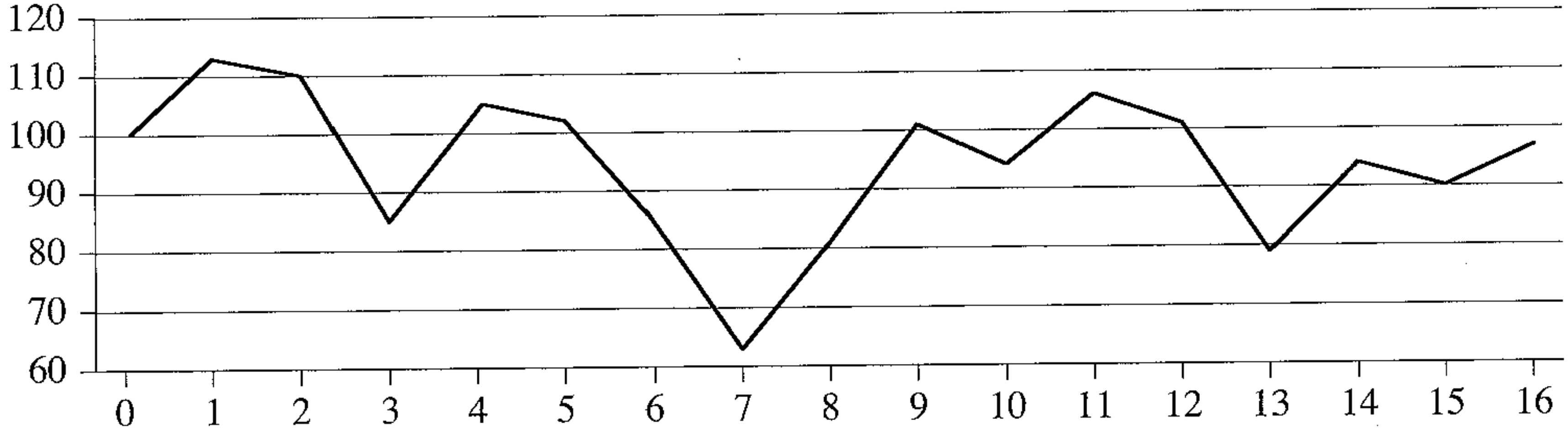


Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future



Maximum-Subarray Example

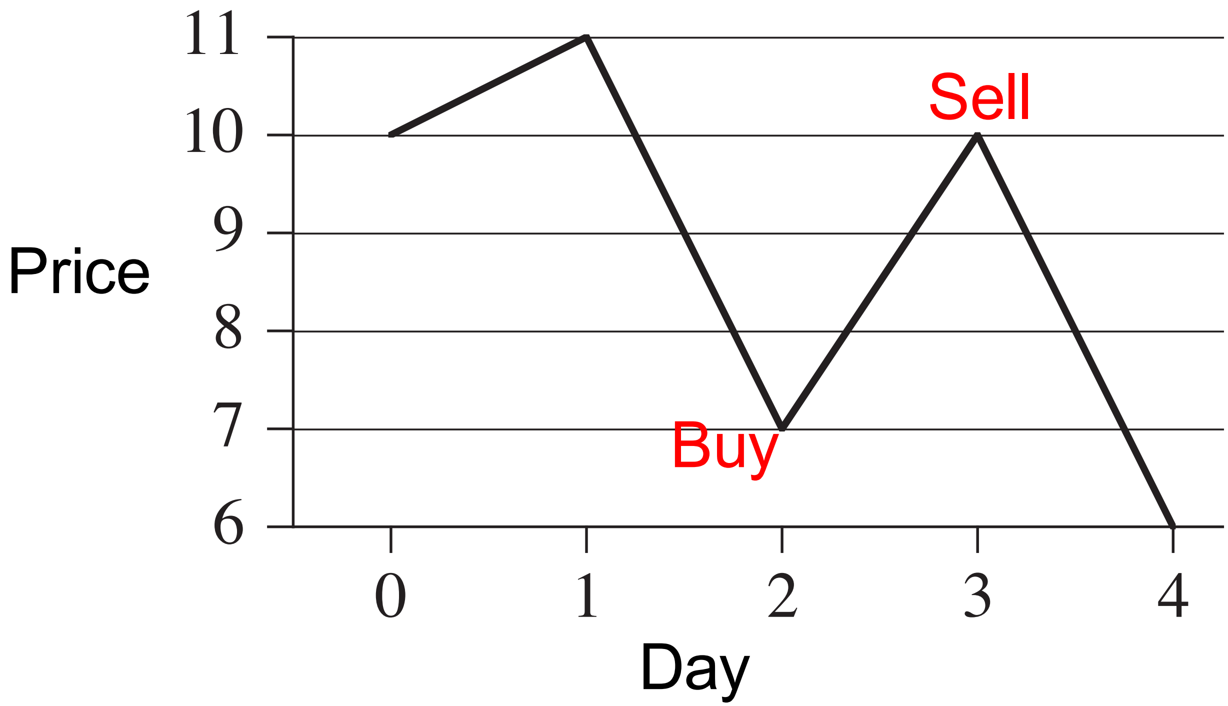


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Figure 4.1 Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.

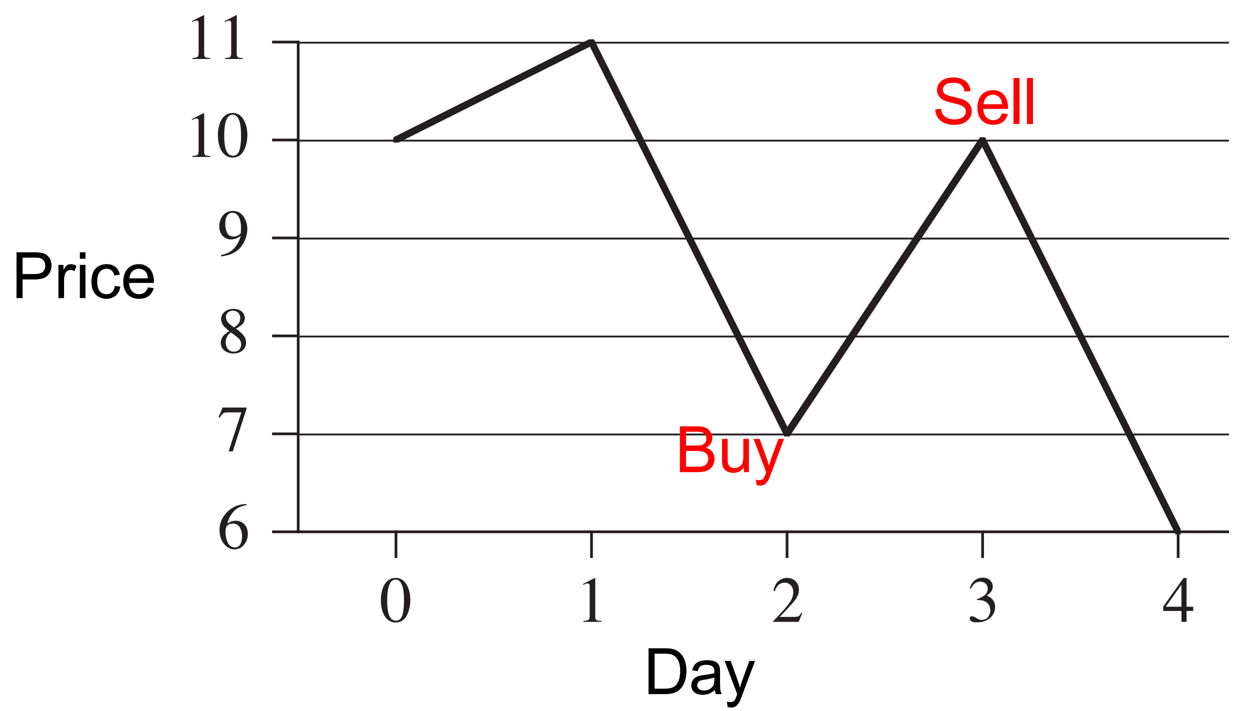
Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future



Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future. Complexity?

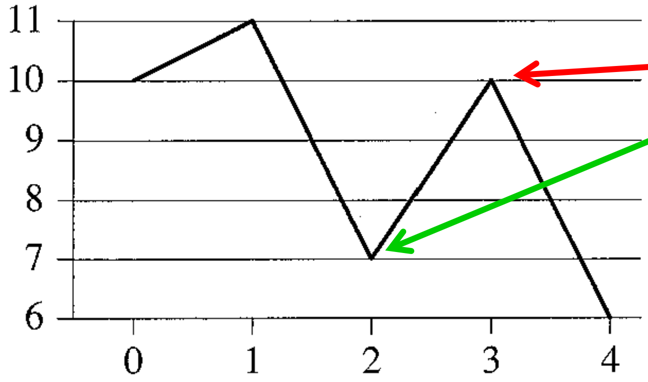


Maximum-Subarray Example

Buying low and selling high doesn't always work

Best strategy:
Buy here
Sell here

But doesn't follow "buy low, sell high" rule

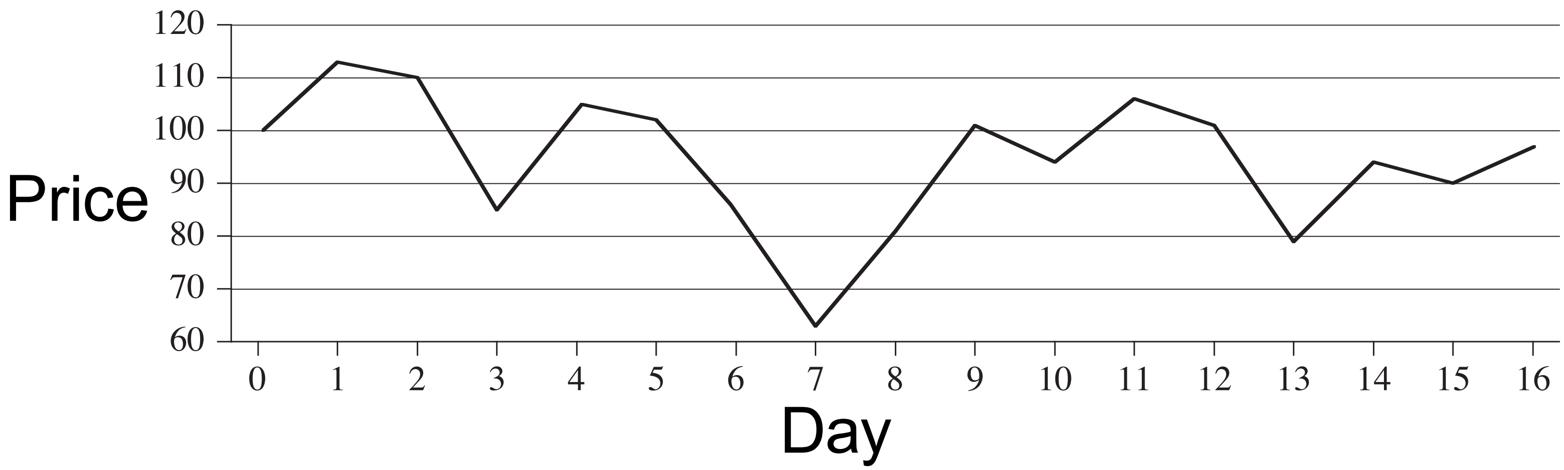


Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

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.

Max Subarray Problem

Brute force: Can we do better? Try to reframe as greatest sum of any contiguous array

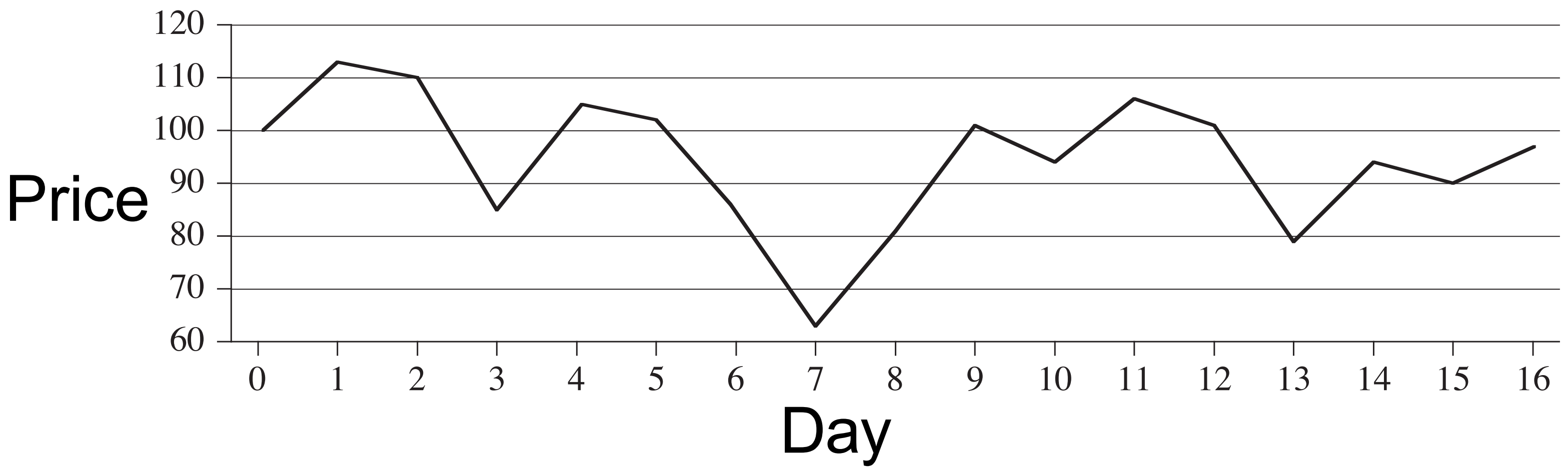


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

best contiguous sum representing gain from buy to sell!

Max Subarray Problem

Brute force: Can we do better? Try to reframe as greatest sum of any contiguous array



	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

Max Subarray Problem

Brute force: Try every possible pair of buy and sell dates:

$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)(n-2)!}{(n-2)!2!} = \frac{n(n-1)}{2} = \boxed{\Theta(n^2)}$$

Can we do better?

Max Subarray Problem

Try to reframe as greatest sum of any contiguous array. **If all the array values were positive?**

A = [1 10 12 13 23 33 2]

Max Subarray Problem

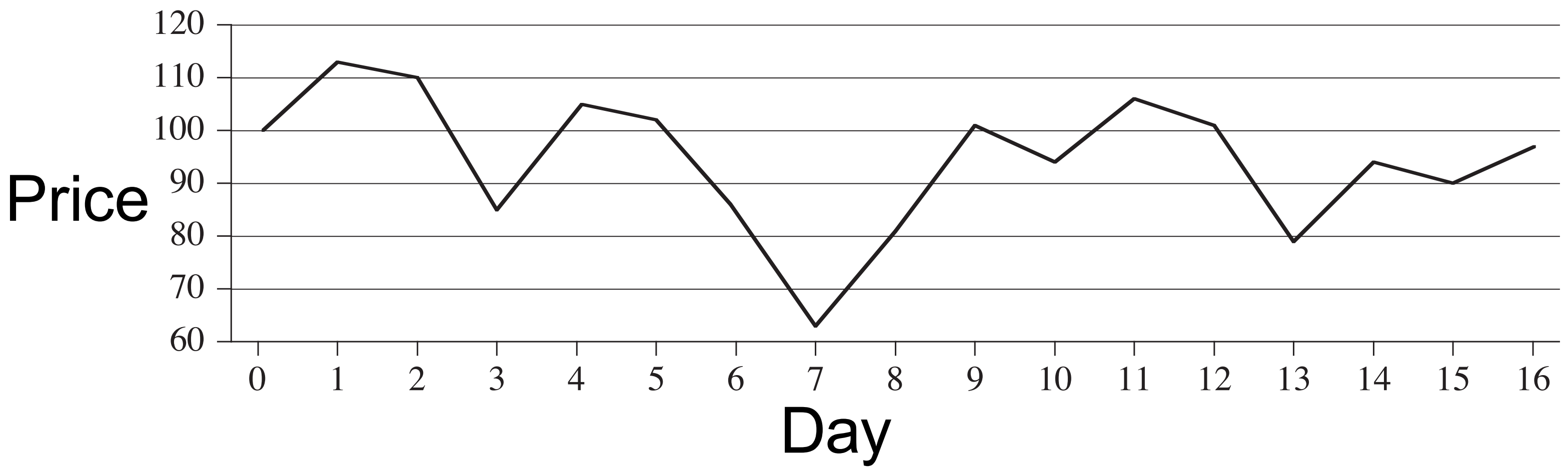
Try to reframe as greatest sum of any contiguous array. **If all the array values were positive?**

A = [1 10 12 13 23 33 2]

Not interesting – summing all array values gives the max…

Max Subarray Problem

For positive and negative values, it's **still brute force.**
Divide and Conquer?



	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

Divide-and-Conquer Approach

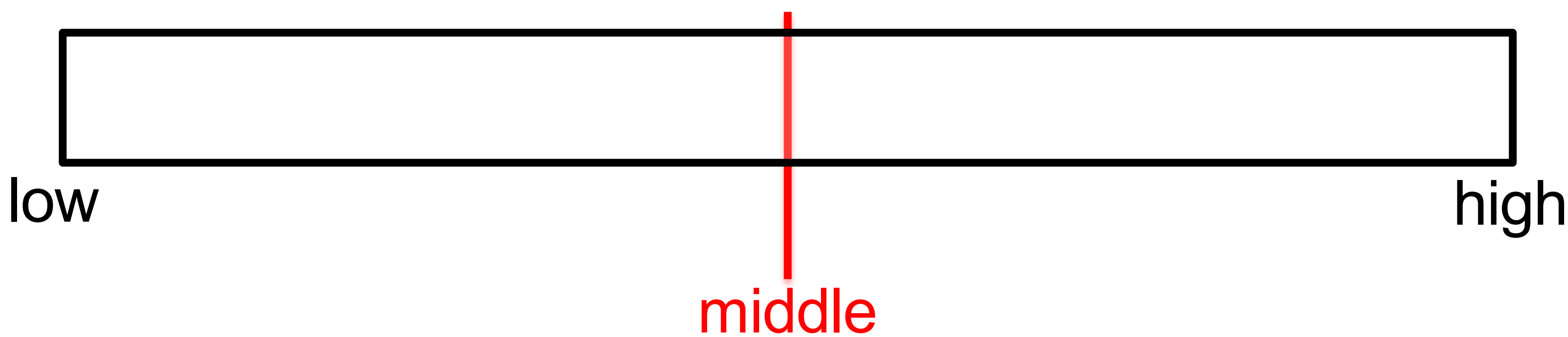
- How to divide?
  - Divide into 2 arrays
- What is the base case?
- How to combine the subproblem solutions?

How to solve more efficiently?

- If we know the price difference of each 2 contiguous days
- The solution can be found from the **maximum-subarray**
- **Maximum-subarray** of array A is:
  - A subarray of A
  - Nonempty
  - Contiguous
  - Whose values have the largest sum

Max Subarray Problem

Divide and conquer approach



1. **Divide** subarray into two equal size subarrays, A[low..mid] and A[mid+1..high]
2. **Conquer**, finding max of subarrays A[low..mid] and A[mid+1..high]
3. **Combine**, finding best solution of:
   a. the two solutions found in conquer step
   b. solution of subarray crossing the midpoint

Max Subarray Problem

Divide and conquer approach

Keep recursing until low=high (one element left)-

1. Divide subarray into two equal size subarrays, A[low..mid] and A[mid+1..high]
2. Conquer, finding max of subarrays A[low..mid] and A[mid+1..high]
3. Combine, finding best solution of:
   a. the two solutions found in conquer step
   b. **solution of subarray crossing the midpoint**

Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 | -7 12 -5 -22]

- **Start from middle**
- **Traverse to left until get maximum sum (?)**
- **Traverse to right until get maximum sum (?)**
- **Return total left and right sum (?)**

Complexity?

Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 | -7 12 -5 -22]

- **Start from middle**
- **Traverse to left until get maximum sum (?)**
- **Traverse to right until get maximum sum (?)**
- **Return total left and right sum (?)**

Complexity? $\Theta(n)$

Divide-and-Conquer Approach

- Note where solution must lie:

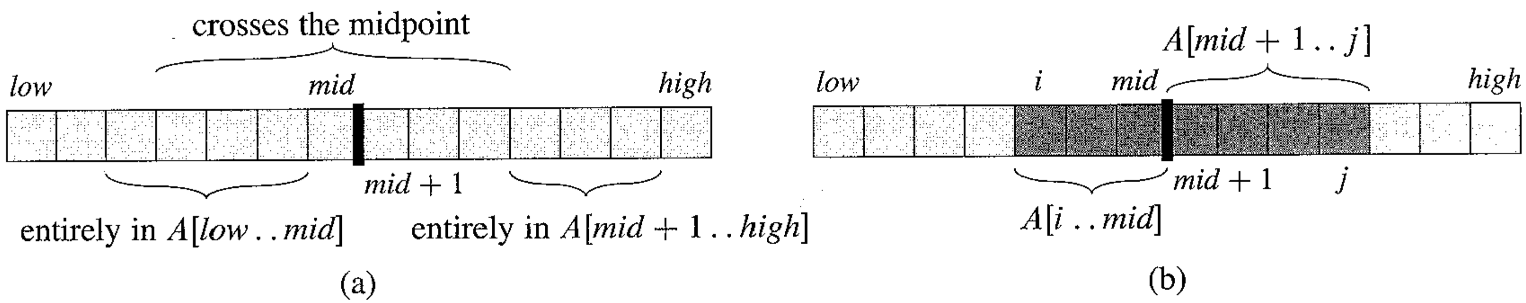


Figure 4.4 **(a)** Possible locations of subarrays of $A[low \, .. \, high]$: entirely in $A[low \, .. \, mid]$, entirely in $A[mid+1 \, .. \, high]$, or crossing the midpoint *mid*. **(b)** Any subarray of $A[low \, .. \, high]$ crossing the midpoint comprises two subarrays $A[i \, .. \, mid]$ and $A[mid+1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

- 3 choices:
  - A[i, …, mid] // best is in the left array
  - A[mid+1, …, j] // best is in the right array
  - A[…, mid, mid+1….] // best is in the array across the midpoint
  - The maximum subarray for A[i,…,j] is the best of these 3 choices

Maximum-subarray problem – divide-and-conquer algorithm

Input: array A[i, …, j]

Ouput: sum of maximum-subarray, start point of maximum-subarray, end point of maximum-subarray

FindMaxSubarray:

1. if(j<=i) return (A[i], i, j);
2. mid = floor(i+j);
3. (sumCross, startCross, endCross) = **FindMaxCrossingSubarray**(A, i, j, mid);
4. (sumLeft, startLeft, endLeft) = **FindMaxSubarray**(A, i, mid);
5. (sumRight, startRight, endRight) = **FindMaxSubarray**(A, mid+1, j);
6. Return the largest of these 3

Maximum-subarray problem – divide-and-conquer algorithm

Input: array A[i, …, j]

Ouput: sum of maximum-subarray, start point of maximum-subarray, end point of maximum-subarray

FindMaxSubarray:

1. if(j<=i) return (A[i], i, j);
2. mid = floor(i+j);
3. (sumCross, startCross, endCross) = **FindMaxCrossingSubarray**(A, i, j, mid);
4. (sumLeft, startLeft, endLeft) = **FindMaxSubarray**(A, i, mid);
5. (sumRight, startRight, endRight) = **FindMaxSubarray**(A, mid+1, j);
6. Return the largest of these 3

Time complexity? $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$ = $\Theta(n \lg n)$

Max Subarray Problem

Divide and conquer approach: full example:

[-16 -23 18 20 -7 12 -5 -22]

On the board…

Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 -7 12 -5 -22]

```
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)
```

$\Theta(n)$

Max Subarray Problem

Costs:

1. **Divide:** $\Theta(1)$

2. **Conquer:** $2T(\frac{n}{2})$

3. **Combine:** $\Theta(n) + \Theta(1)$

Subarray crossing

Comparisons

Total: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n\log n)$

Like merge sort….