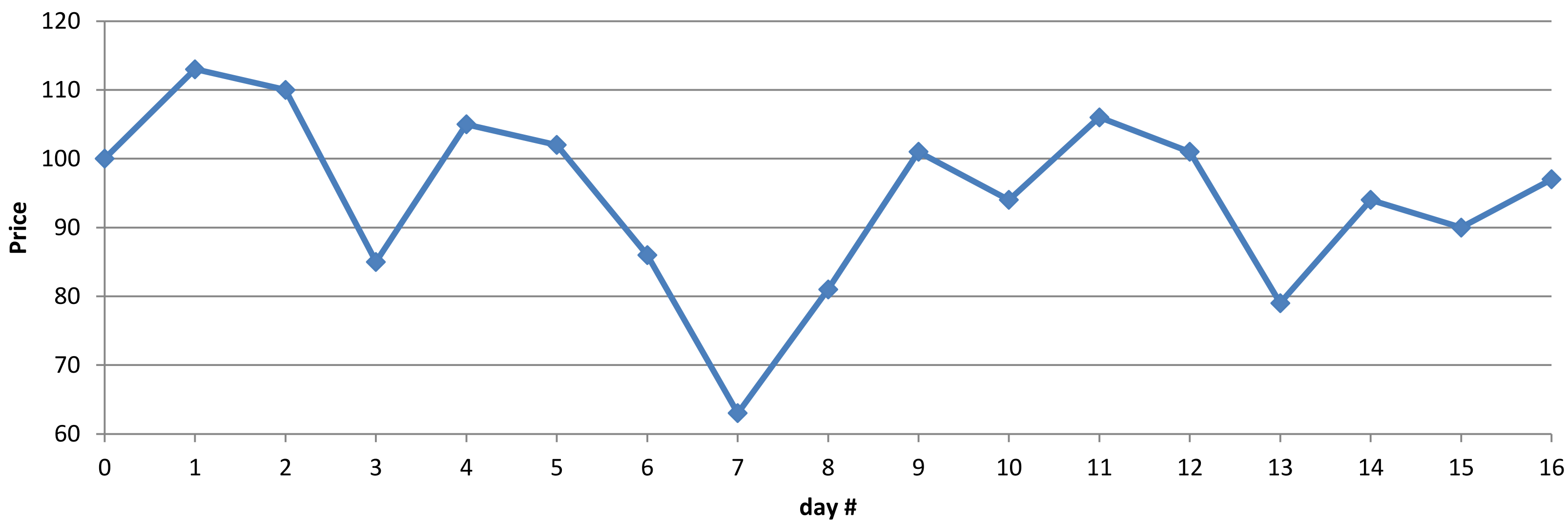
Design and Analysis of Algorithms

Maximum-subarray problem

Slides from Haidong Xue

Maximum-subarray problem back ground

- 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?



Maximum-subarray problem back ground

- What is the **brute-force** solution?

```
max = -infinity;
for each day pair p {
    if(p.priceDifference>max)
        max=p.priceDifference;
}
```

Time complexity? $\binom{n}{2}$ pairs, so O(n^2)

Maximum-subarray problem back ground

- 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 the largest sum

Maximum-subarray problem back ground

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

What is the solution? Buy on day 2, sell on day 3

Can be solve it by the maximum-subarray of difference array?

Sub-array	1-1	1-2	1-3	1-4	2-2	2-3	2-4	3-3	3-4	4-4
Sum	1	-3	0	-4	-4	-1	-5	3	-1	-4

Maximum-subarray problem – divide-and-conquer algorithm

- How to divide?
  - Divide to 2 arrays
- What is the base case?
- How to combine the sub problem solutions to the current solution?
  - A fact:
    - when divide array A[i, …, j] into A[i, …, mid] and A[mid+1, …, j]
    - A sub array must be in one of them
      - A[i, …, mid] // the left array
      - A[mid+1, …, j] // the right array
      - A[…, mid, mid+1….] // the array across the midpoint
  - The maximum subarray is the largest sub-array among maximum subarrays of those 3

Maximum-subarray problem – divide-and-conquer algorithm

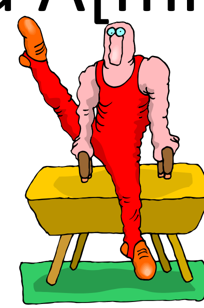
- Input: array A[i, …, j]
- Ouput: sum of maximum-subarray, start point of maximum-subarray, end point of maximum-subarray
- **FindMaxSubarray**:

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 one from those 3

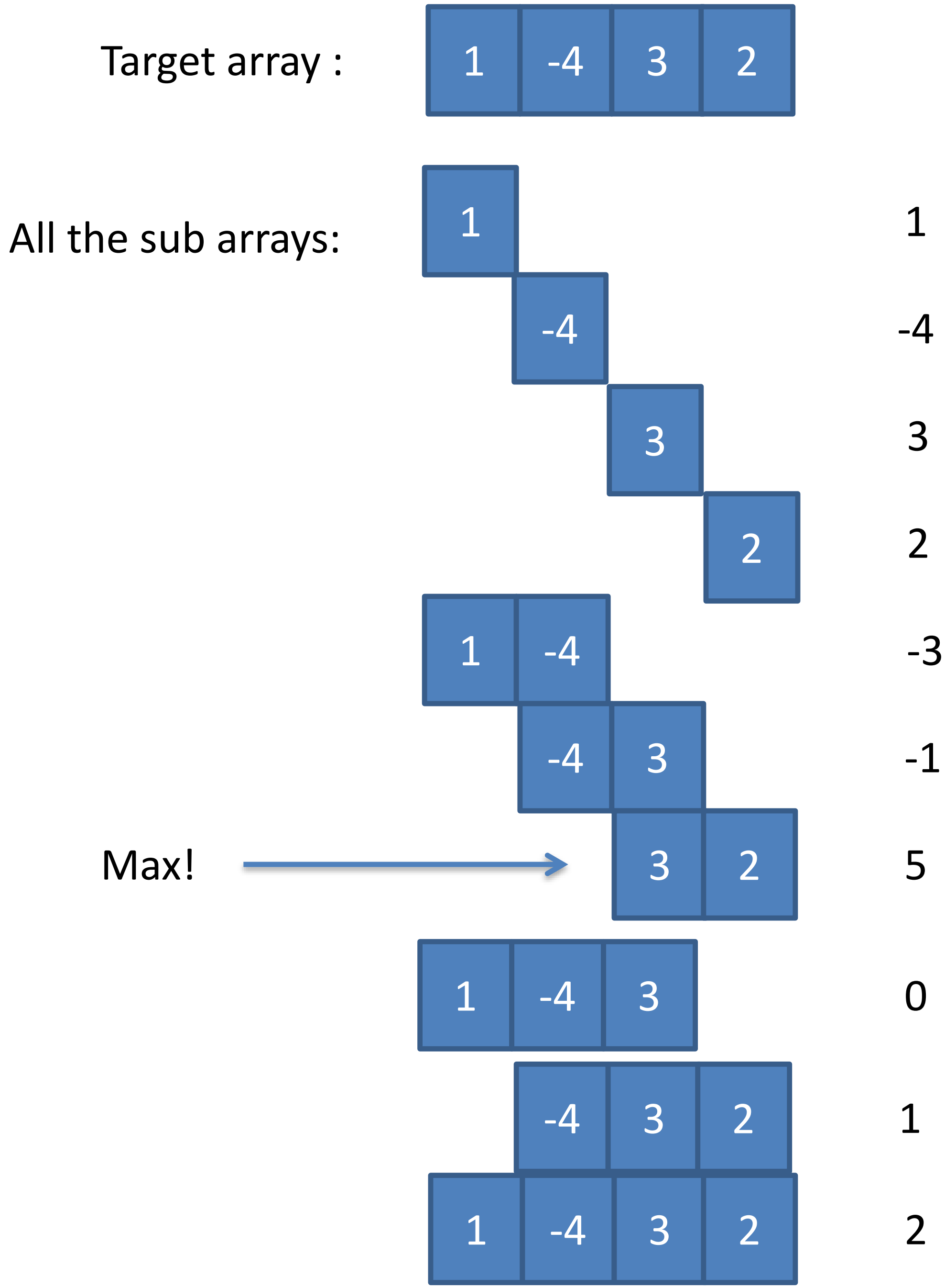
Maximum-subarray problem – divide-and-conquer algorithm

FindMaxCrossingSubarray(A, i, j, mid)

1. Scan A[i, mid] once, find the largest A[left, mid]
2. Scan A[mid+1, j] once, find the largest A[mid+1, right]
3. Return (sum of A[left, mid] and A[mid+1, right], left, right)



Let’s try it in Java



Target array :
1 -4 3 2
All the sub arrays:
1
1
-4
-4
3
3
2
2
1 -4
-3
-4 3
-1
Max!
3 2
5
1 -4 3
0
-4 3 2
1
1 -4 3 2
2

Target array :

All the sub arrays:

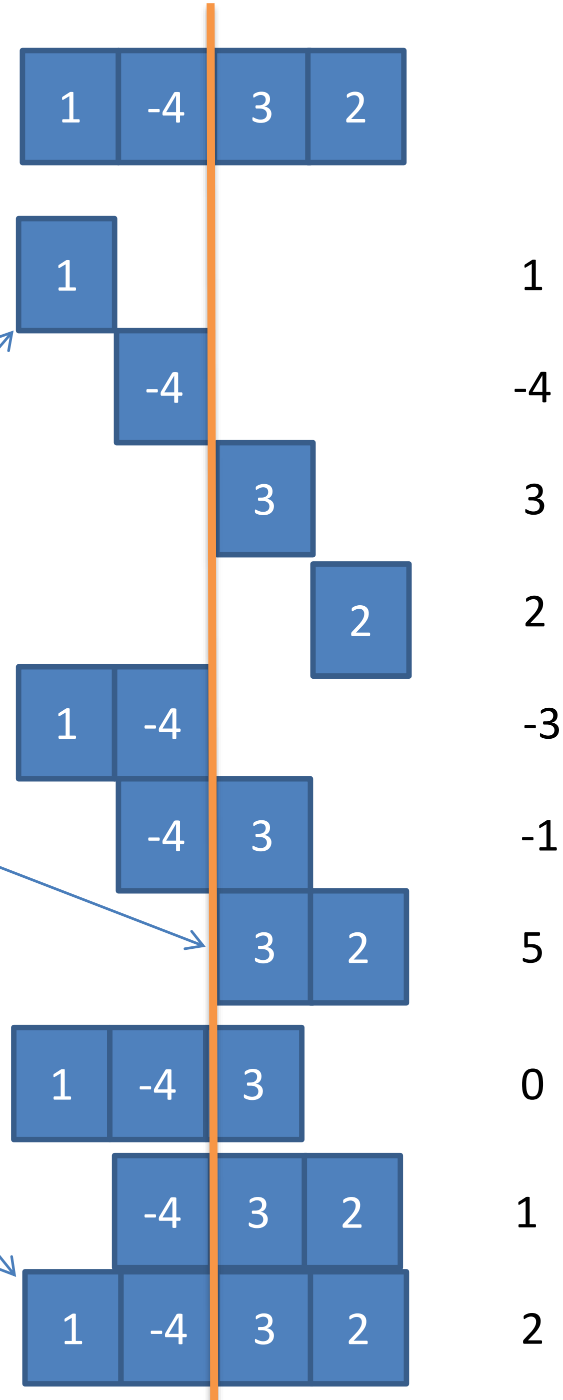
The problem can be then solved by:

1. Find the max in left sub arrays

2. Find the max in right sub arrays

3. Find the max in crossing sub arrays

4. Choose the largest one from those 3 as the final result



Divide target array into 2 arrays

We then have 3 types of subarrays:

The ones belong to the left array

The ones belong to the right array

The ones crossing the mid point

FindMaxSub (| 1 | -4 | 3 | 2 |)

1. Find the max in left sub arrays **FindMaxSub** (| 1 | -4 |)

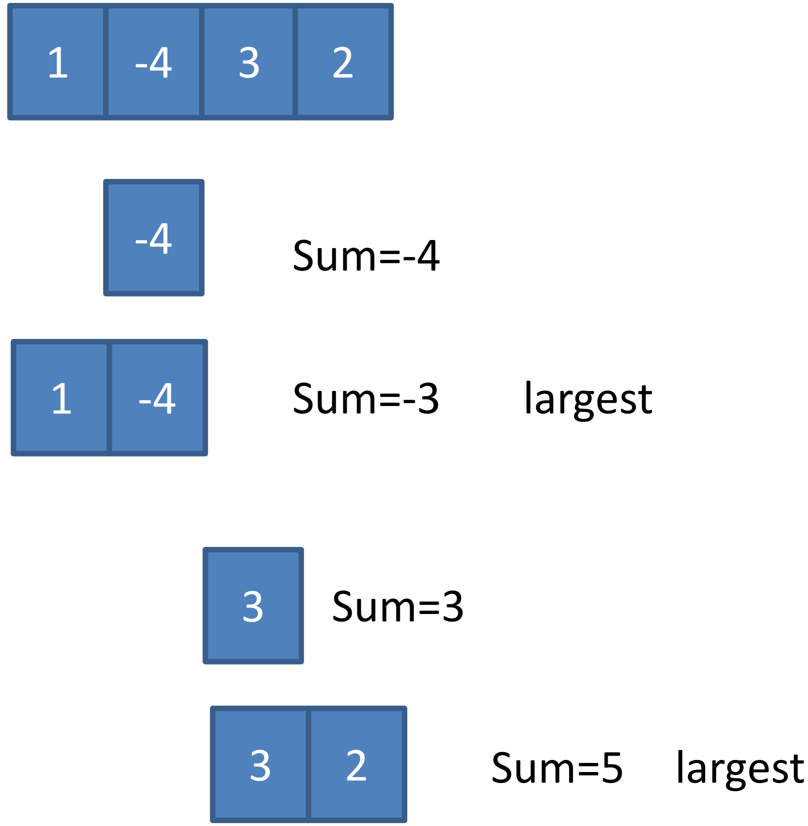
2. Find the max in right sub arrays **FindMaxSub** (| 3 | 2 |)

3. Find the max in crossing sub arrays

   Scan | 1 | -4 | once, and scan | 3 | 2 | once

4. Choose the largest one from those 3 as the final result

3. Find the max in crossing sub arrays



The largest crossing subarray is :

Time Complexity

$$T(n) = O(1) \text{ if n = 1}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \text{ if n > 1}$$