Divide-and-Conquer

謝仁偉 副教授
jenwei@mail.ntust.edu.tw
國立台灣科技大學 資訊工程系
2017 Spring

Outline

- ***Introduction***
- The Maximum-Subarray Problem
- The Matrix Multiplication Problem

Divide-and-Conquer

- Divide the problem into a number of subproblems that are smaller instances of the same problem.
- Conquer the subproblems by solving them recursively.
  - If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem.

Recurrences (1/2)

- A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.
  - The worst-case running time $T(n)$ of the **MERGE-SORT** procedure can be described by the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 , \end{cases}$$

whose solution we claimed to be $T(n) = \Theta(n \lg n)$.

Recurrences (2/2)

- Recurrences can take many forms:
  - A recursive algorithm might divide subproblems into unequal sizes, such as a 2/3-to-1/3 split.
    - $T(n) = T(2n/3) + T(n/3) + \Theta(n)$.
  - Subproblems are not necessarily constrained to being a constant fraction of the original problem size.
    - For example, a recursive version of linear search would create just one subproblem containing only one element fewer than the original problem.
    - $T(n) = T(n-1) + \Theta(1)$.

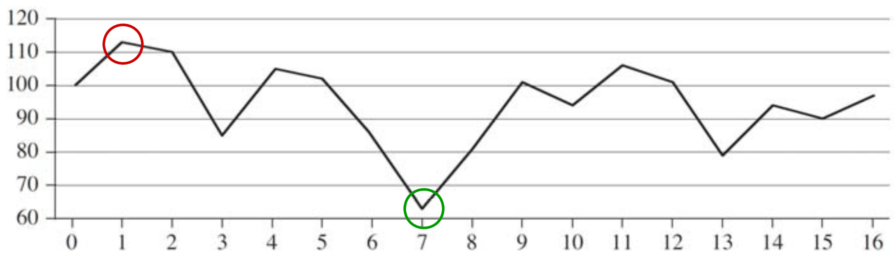
Outline

- Introduction
- ***The Maximum-Subarray Problem***
- The Matrix Multiplication Problem

Maximum-Subarray Problem (1/2)

- Suppose that you been offered the opportunity to invest a company. The stock price of the company is rather volatile.
- You are allowed to buy one unit of stock only one time and then sell it at a later date, buying and selling after the close of trading for the day.
- To compensate for this restriction, you are allowed to learn what the price of the stock will be in the future. Your goal is to maximize your profit.

Maximum-Subarray Problem (2/2)

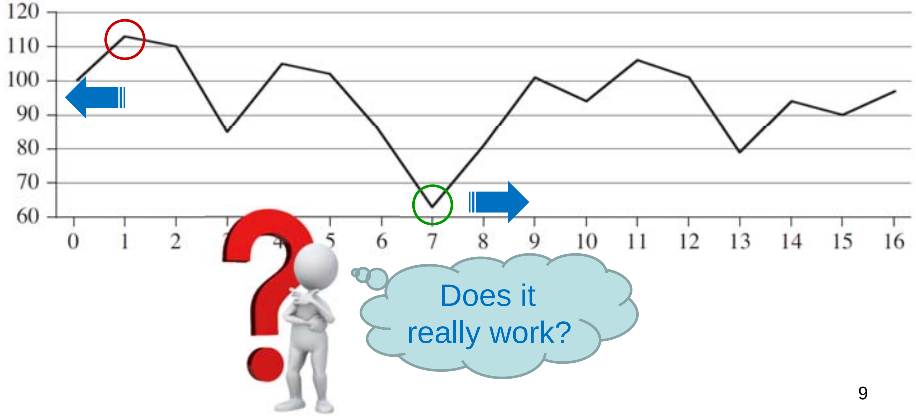


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

- You would want to buy at the lowest possible price and later on sell at the highest possible price—to maximize your profit.
- ☞ Unfortunately, you might not be able to buy at the lowest price and then sell at the highest price within a given period.

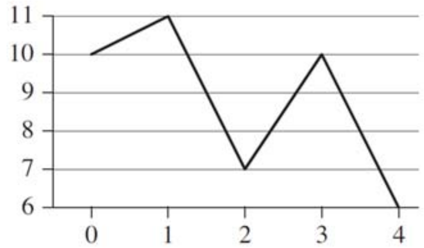
How to Maximize Your Profit? (1/2)

- You might think that you can always maximize profit by either buying at the lowest price or selling at the highest price.



How to Maximize Your Profit? (2/2)

- How about this case?



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

➡ The maximum profit does not always start at the lowest price or end at the highest price!

Brute-Force Solution

- We can easily devise a brute-force solution to this problem: just try every possible pair of buy and sell dates in which the buy date precedes the sell date.

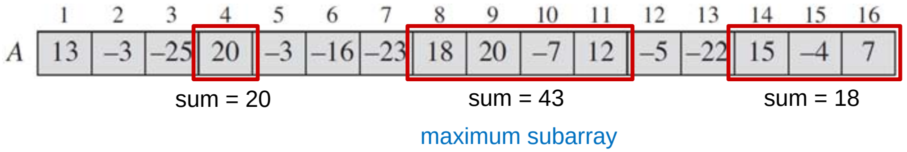
➡ A period of n days has $C(n, 2)$ such pairs of dates, and this approach would take $\Omega(n^2)$ time.

Can we do better?

Transformation

- We want to find a sequence of days over which the net change from the first day to the last is maximum.
- Instead of looking at the daily prices, let us instead consider the daily change in price, where the change on day i is the difference between the prices after day $i - 1$ and after day i.
- We now want to find the nonempty, contiguous subarray of *A* with the largest sum.
  - We call this contiguous subarray the maximum subarray.

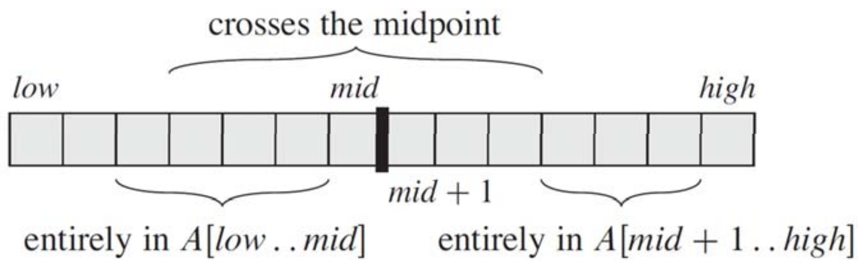
Maximum Subarray



- The maximum-subarray problem is interesting only when the array contains some negative numbers.
  – If all the array entries were nonnegative, then the maximum-subarray problem would present no challenge, since the entire array would give the greatest sum.

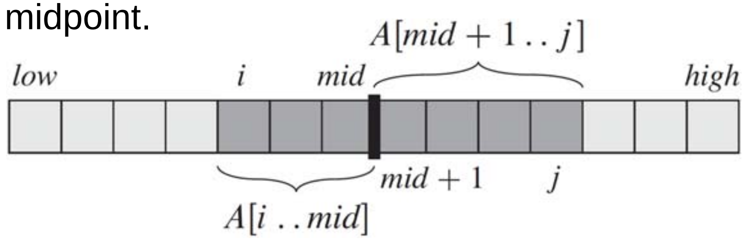
Solution Using Divide-and-Conquer (1/2)

- Suppose we want to find a maximum subarray of the subarray *A*[low .. High].
- Divide-and-conquer suggests that we divide the subarray into two subarrays of as equal size as possible.



Solution Using Divide-and-Conquer (2/2)

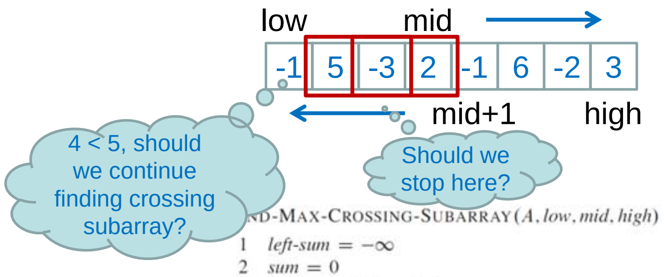
- To find a maximum subarray crossing the midpoint is not a smaller instance of our original problem, because it has the added restriction that the subarray it chooses must cross the midpoint.



  – We just need to find maximum subarrays of the form *A*[*i* .. *mid*] and *A*[*mid* + 1 .. *j*] and then combine them.


```
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)
```

Illustration




```
…ND-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)
```



```
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)
```


Outline

- Introduction
- The Maximum-Subarray Problem
- ***The Matrix Multiplication Problem***

The Matrix Multiplication Problem

- If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then in the product $C = A \cdot B$, we define the entry c_{ij}, for $i, j = 1, 2, \ldots, n$, by $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

It takes $\Theta(n^3)$ time.