CSE331 Introduction to Algorithms
Lecture 5: Maximum Subarray

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

- Last time, we saw a divide-and-conquer algorithm: MERGE SORT.
- Today, we will see another example of a divide-and-conquer algorithm.

- **Reference for this lecture**:
  - Section 4.1 of the textbook (p. 68–74)
    Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

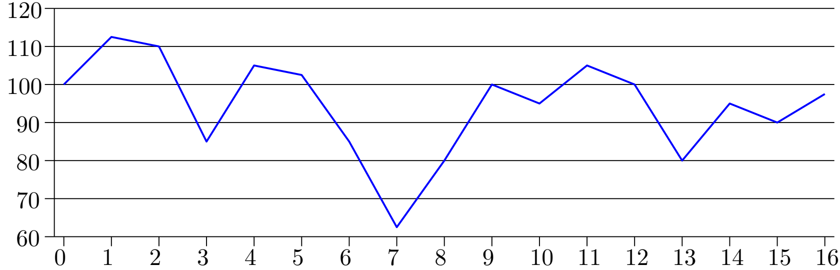
The Divide-and-Conquer Approach

- MERGE SORT follows a *divide-and-conquer* approach.
- Divide and Conquer approach:
  - **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.
- MERGE SORT follows this approach.
  - **Divide**: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.
  - **Conquer**: Sort the two subsequences recursively using MERGE SORT.
  - **Combine:** Merge the two sorted subsequences to produce the sorted answer.

The Divide-and-Conquer Approach

- The divide-and-conquer approach is one of the most important algorithm design technique.
- It can often be analyzed using the recursion tree method.
  - We will see other methods later this semester.
- Not all algorithms follow this approach.
- For instance, INSERTION SORT follows an *incremental* approach:
  - After sorting $A[1 \dots j-1]$, we insert $A[j]$ at the proper place.
  - An incremental algorithm adds elements one by one.
- In this lecture, we present a divide and conquer algorithm for the MAXIMUM SUBARRAY problem.
- We will see more examples in the next lectures.

Buying and Selling

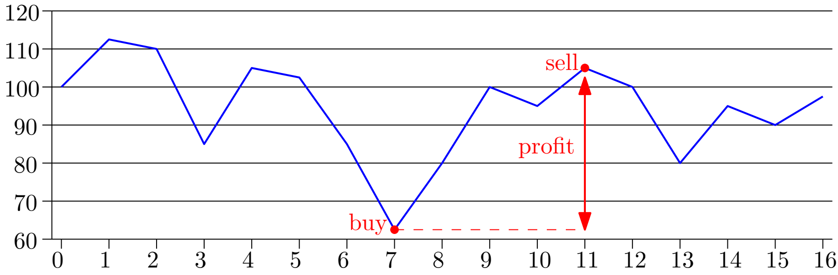


day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Problem

When were the best times to buy and sell?

Buying and Selling



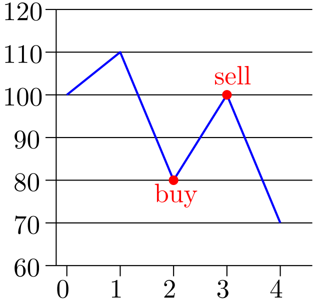
day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Answer

Buy on day 7, sell on day 11. Profit: 43

Buying and Selling

- Difficulty: We don't necessarily buy at the lowest price or sell at the highest price.
- Example:



Problem Transformation

- Consider the array $A[1 \dots 16]$ of change in price.

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

Maximum subarray $A[8 \dots 11]$

- The *maximum subarray* is the contiguous subarray whose elements have the largest sum. Here, it is $A[8 \dots 11]$.
- So the best times to buy and sell are days 7 and 11.
- We reduced our original problem to:

Problem

Given an array $A[1 \dots n]$ of n numbers, find the indices p, q such that $1 \leqslant p \leqslant q \leqslant n$ and $\sum_{i=p}^{q} A[i]$ is maximum.

The Maximum Subarray Problem

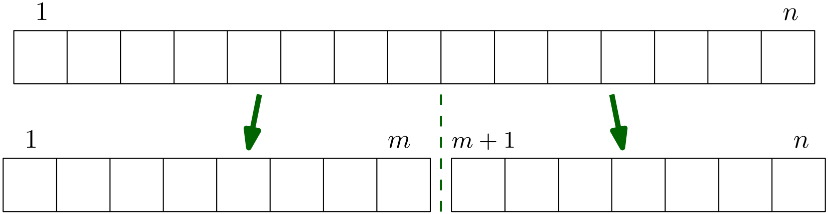
Brute-force approach to maximum subarray

procedure MAXIMUM-SUBARRAY$(A, 1, n)$
  $\max \leftarrow -\infty$
  for $i \leftarrow 1, n$ **do**
    for $j \leftarrow i, n$ **do**
      $\mathrm{sum} \leftarrow 0$
      for $k \leftarrow i, j$ **do**
        $\mathrm{sum} \leftarrow \mathrm{sum} + A[k]$
      if $\mathrm{sum} > \max$ **then**
        $\max \leftarrow \mathrm{sum}$, $p \leftarrow i$, $q \leftarrow j$
  return p, q, max

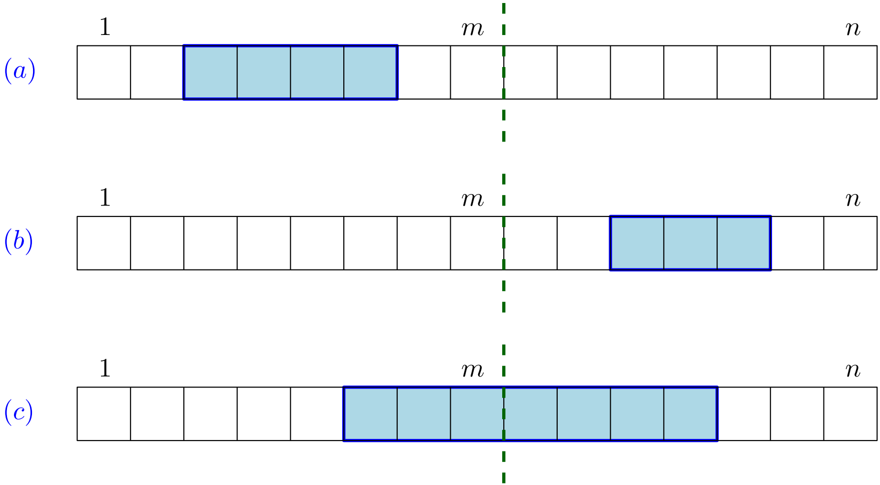
- Running time? $\Theta(n^3)$.
- Proof done in class. See lecture notes.

Divide-and-Conquer Approach

- This is not very good, because our original problem of buying and selling stock could be solved by brute force in time $\Theta(n^2)$.
- We will now give a $\Theta(n \log n)$-time divide and conquer algorithm.
- (It is possible to solve MAXIMUM SUBARRAY in linear time. See Wikipedia.)
- Divide & conquer approach: *Divide* $A[1 \dots n]$ into $A[1 \dots m], A[m+1 \dots n]$ where $m = \lceil n/2 \rceil$.



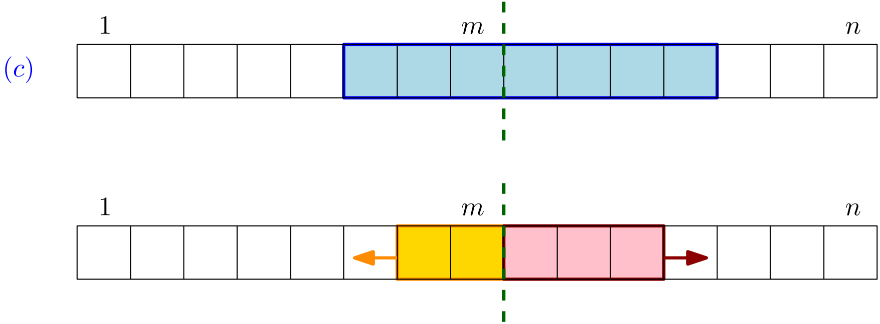
The MAXIMUM SUBARRAY Problem



The Maximum Subarray Problem

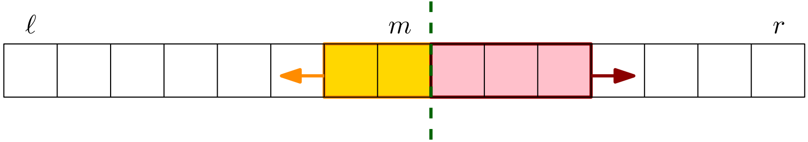
- Three cases: The maximum subarray
  (a) is contained in $A[1 \dots m]$,
  (b) is contained in $A[m+1 \dots n]$, or
  (c) crosses the boundary between the two subarrays.
- Cases (a) and (b) are handled by recursing on $A[1 \dots m]$ and $A[m+1 \dots n]$, respectively. This is the *conquer* step.
- Case (c) is non-trivial. We will give an efficient algorithm for it, which is the main part of the *combine* step.

Maximum Crossing Subarray



- Idea: we can maximize the left part and the right part separately, and then combine the two maxima.

Maximum Crossing Subarray



- Similarly as we did for MERGE SORT, our algorithm solves this problem for the subarray $A[\ell \ldots r]$ split at index $m = \lfloor (\ell + r)/2 \rfloor$.

Maximum Crossing Subarray

Pseudocode

procedure MAX-CROSSING-SUBARRAY(A, ℓ, m, r)

 $M_1 \leftarrow -\infty$, $\mathrm{sum} \leftarrow 0$

 for $i \leftarrow m+1, r$ **do**

  $\mathrm{sum} \leftarrow \mathrm{sum} + A[i]$

  if $\mathrm{sum} > M_1$ **then**

   $M_1 \leftarrow \mathrm{sum}$, $q \leftarrow i$

 $M_2 \leftarrow -\infty$, $\mathrm{sum} \leftarrow 0$

 for $i \leftarrow m, \ell$ **do** $\triangleright$ i goes down from m to ℓ

  $\mathrm{sum} \leftarrow \mathrm{sum} + A[i]$

  if $\mathrm{sum} > M_2$ **then**

   $M_2 \leftarrow \mathrm{sum}$, $p \leftarrow i$

 return p, q, $M_1 + M_2$

- The result is $A[p \dots q]$ with sum $M_1 + M_2$.

Maximum Crossing Subarray

- Analysis: runs in *linear time*.
- More precisely, $\Theta(n')$ where $n' = r - \ell + 1$ is the size of the input subarray $A[\ell \dots r]$.

The Maximum Subarray Problem

Pseudocode

procedure MAXIMUM-SUBARRAY(A, ℓ, r)
  if $\ell = r$ **then**
    return ℓ, r, $A[\ell]$ ▷ base case: 1-element array
  $m \leftarrow \lfloor (\ell + r)/2 \rfloor$
  $p_a, q_a, M_a \leftarrow$ MAXIMUM-SUBARRAY(A, ℓ, m) ▷ case (a)
  $p_b, q_b, M_b \leftarrow$ MAXIMUM-SUBARRAY$(A, m+1, r)$ ▷ case (b)
  $p_c, q_c, M_c \leftarrow$ MAX-CROSSING-SUBARRAY(A, ℓ, m, r) ▷ case (c)
  $r = \arg\max_{x \in \{a,b,c\}} M_x$ ▷ i.e. $M_r = \max(M_a, M_b, M_c)$
  return p_r, q_r, M_r

The Maximum Subarray Problem

- **Divide:** Line 4
- **Conquer:** Lines 5–6
- **Combine:** Lines 7–9
- Analysis:
  - Suppose n is a power of 2, and $n \geqslant 2$. ($n = 2^h$ for some integer $h \geqslant 1$.)
  - let $T(n)$ be the running time.
  - Lines 5 and 6 take $T(n/2)$ each.
  - Line 7 takes $\Theta(n)$.
  - Other lines take $\Theta(1)$.
  - Therefore

$$T(n) = 2T(n/2) + \Theta(n)$$

The MAXIMUM SUBARRAY Problem

- Combined with the base case (Line 2–3) we obtain the recurrence relation:

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

- $T(n) = \Theta(n \log n)$ because we found the same relation for MERGE SORT.