

# ECS 122A: Algorithm Design and Analysis

## Week 4 Discussion

© Ji Wang

Apr 21, 2021

# Outline

- ▶ Binary Integer Multiplication (Homework 3 Problem 7)
- ▶ Variant of Maximum-Subarray Problem: Stock Investment

# Binary Integer Multiplication

Recall how we multiply two integers of equal length =  $n$

	1100
	<u>× 1101</u>
	1100
	0000
	1100
	<u>1100</u>
	10011100
(a)	(b)

# Binary Integer Multiplication

Recall how we multiply two integers of equal length =  $n$

	1100
	$\times 1101$
	<hr/>
	1100
	0000
	1100
	<hr/>
	10011100
	<b>(b)</b>
12	
$\times 13$	
<hr/>	
36	
12	
<hr/>	
156	
<b>(a)</b>	

**Time complexity:**  $O(n^2)$

1. bit multiplication:  $n^2$
2. bit addition:  $O(n)$

# Binary Integer Multiplication: Speedup

First, how do we represent a decimal integer digit-wise?

$$1024 = 10^3 \times 1 + 10^2 \times 0 + 10^1 \times 2 + 10^0 \times 4$$

# Binary Integer Multiplication: Speedup

First, how do we represent a decimal integer digit-wise?

$$1024 = 10^3 \times 1 + 10^2 \times 0 + 10^1 \times 2 + 10^0 \times 4$$

**Or,**  $1024 = \boxed{10} \boxed{24} = 10^2 \times 10 + 24$

# Binary Integer Multiplication: Speedup

First, how do we represent a decimal integer digit-wise?

$$1024 = 10^3 \times 1 + 10^2 \times 0 + 10^1 \times 2 + 10^0 \times 4$$

**Or,**  $1024 = \boxed{10} \boxed{24} = 10^2 \times 10 + 24$

Now, how about two  $n$ -bit binary integers?

$$x = \boxed{a} \boxed{b} = 2^{\frac{n}{2}} a + b$$

$$y = \boxed{c} \boxed{d} = 2^{\frac{n}{2}} c + d$$

For instance,  $10110110 = \boxed{1011} \boxed{0110} = 2^4 \times 1011 + 0110$

$$\begin{aligned} xy &= (2^{\frac{n}{2}} a + b)(2^{\frac{n}{2}} c + d) \\ &= 2^n ac + 2^{\frac{n}{2}} (ad + bc) + bd \end{aligned}$$

# Binary Integer Multiplication: Speedup

First, how do we represent a decimal integer digit-wise?

$$1024 = 10^3 \times 1 + 10^2 \times 0 + 10^1 \times 2 + 10^0 \times 4$$

**Or,**  $1024 = \boxed{10} \boxed{24} = 10^2 \times 10 + 24$

Now, how about two  $n$ -bit binary integers?

$$x = \boxed{a} \boxed{b} = 2^{\frac{n}{2}} a + b$$

$$y = \boxed{c} \boxed{d} = 2^{\frac{n}{2}} c + d$$

For instance,  $10110110 = \boxed{1011} \boxed{0110} = 2^4 \times 1011 + 0110$

$$\begin{aligned} xy &= (2^{\frac{n}{2}} a + b)(2^{\frac{n}{2}} c + d) \\ &= 2^n ac + 2^{\frac{n}{2}} (ad + bc) + bd \end{aligned}$$

**Time complexity:** <sup>1</sup>

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + O(n) + \Theta(1) \\ &= O(n^2) \quad \rightarrow \text{No improvement!} \end{aligned}$$

---

<sup>1</sup> $2^n x$  is done by left shift with a constant time cost



# Binary Integer Multiplication

Previously, we wrote:

$$\begin{aligned}xy &= (2^{\frac{n}{2}}a + b)(2^{\frac{n}{2}}c + d) \\ &= 2^n \underline{ac} + 2^{\frac{n}{2}}(ad + bc) + \underline{bd}\end{aligned}$$

If we rewrite

$$ad + bc = \underline{(a + b)(c + d)} - ac - bd,$$

then the entire  $xy$  will only involve three multiplications, namely the underlined parts.

# Binary Integer Multiplication

Previously, we wrote:

$$\begin{aligned}xy &= (2^{\frac{n}{2}}a + b)(2^{\frac{n}{2}}c + d) \\ &= 2^n \underline{ac} + 2^{\frac{n}{2}}(ad + bc) + \underline{bd}\end{aligned}$$

If we rewrite

$$ad + bc = \underline{(a + b)(c + d)} - ac - bd,$$

then the entire  $xy$  will only involve three multiplications, namely the underlined parts.

**Time complexity:**

$$\begin{aligned}T(n) &= 3T\left(\frac{n}{2}\right) + O(n) + \Theta(1) \\ &= O(n^{\log 3})\end{aligned}$$

# Binary Integer Multiplication: Pseudocode

MULTIPLY( $x, y$ )

```
1  //  $x, y$  are positive integers of  $n$ -bit, assuming  $n$  is even
2  if  $n = 1$ 
3      return  $xy$ 
4  else
5       $a, b =$  leftmost  $n/2$ , rightmost  $n/2$  bits of  $x$ 
6       $c, d =$  leftmost  $n/2$ , rightmost  $n/2$  bits of  $y$ 
7       $p_1 =$  MULTIPLY( $a, b$ )
8       $p_2 =$  MULTIPLY( $c, d$ )
9       $p_3 =$  MULTIPLY( $a + b, c + d$ )
10     return  $p_1 \times 2^n + (p_3 - p_1 - p_2) \times 2^{\frac{n}{2}} + p_2$ 
```

# Stock Investment

## Problem Statement:

We're doing a simulation in which we look at  $n$  consecutive days of a given stock, at some point in the past. Let's number the days  $i = 1, 2, \dots, n$  and  $p(i)$  is the price per share for the stock on that day. We want to know: When should we have bought and sold in order to have made as much money as possible? (If there was no way to make money during the  $n$  days, we should report this instead.)

For example,  $n = 4$ ,  $p(1) = 9$ ,  $p(2) = 1$ ,  $p(3) = 5$ ,  $p(4) = 3$ . Then we should answer “buy on day 2, sell on day 3”.

# Stock Investment

## Problem Statement:

We're doing a simulation in which we look at  $n$  consecutive days of a given stock, at some point in the past. Let's number the days  $i = 1, 2, \dots, n$  and  $p(i)$  is the price per share for the stock on that day. We want to know: When should we have bought and sold in order to have made as much money as possible? (If there was no way to make money during the  $n$  days, we should report this instead.)

For example,  $n = 4$ ,  $p(1) = 9$ ,  $p(2) = 1$ ,  $p(3) = 5$ ,  $p(4) = 3$ . Then we should answer “buy on day 2, sell on day 3”.

## Rephrase the problem:

**Input:** array  $p$  of length  $n$

# Stock Investment

## Problem Statement:

We're doing a simulation in which we look at  $n$  consecutive days of a given stock, at some point in the past. Let's number the days  $i = 1, 2, \dots, n$  and  $p(i)$  is the price per share for the stock on that day. We want to know: When should we have bought and sold in order to have made as much money as possible? (If there was no way to make money during the  $n$  days, we should report this instead.)

For example,  $n = 4$ ,  $p(1) = 9$ ,  $p(2) = 1$ ,  $p(3) = 5$ ,  $p(4) = 3$ . Then we should answer "buy on day 2, sell on day 3".

## Rephrase the problem:

**Input:** array  $p$  of length  $n$

**Output:**  $\operatorname{argmax}\{p(j) - p(i)\}$  where  $i \leq j$

# Stock Investment

## Revisit maximum subarray problem:

1. Divide  $A[low \cdots high]$  into two subarrays of as equal size as possible by finding the midpoint  $mid$
2. Conquer:
  - a. finding maximum subarrays of  $A[low \cdots mid]$  and  $A[mid + 1 \cdots high]$
  - b. finding a max-subarray that crosses the midpoint
3. Combine: returning the max of the three

# Stock Investment

## Revisit maximum subarray problem:

1. Divide  $A[low \cdots high]$  into two subarrays of as equal size as possible by finding the midpoint  $mid$
2. Conquer:
  - a. finding maximum subarrays of  $A[low \cdots mid]$  and  $A[mid + 1 \cdots high]$
  - b. finding a max-subarray that crosses the midpoint
3. Combine: returning the max of the three

Can we apply the same strategy on this problem?



# Stock Investment

## How does the conquer part work?

1. The optimal solution to  $p[low \cdots mid]$
2. The optimal solution to  $p[mid + 1 \cdots high]$
3.  $\operatorname{argmax}\{p(j) - p(i)\}$  where  $low \leq i \leq mid$  and  $mid + 1 \leq j \leq high$

# Stock Investment

## How does the conquer part work?

1. The optimal solution to  $p[low \cdots mid]$
2. The optimal solution to  $p[mid + 1 \cdots high]$
3.  $\operatorname{argmax}\{p(j) - p(i)\}$  where  $low \leq i \leq mid$  and  $mid + 1 \leq j \leq high$  (**equivalent to** finding the index of  $\min$  of  $p[low \cdots mid]$  and that of  $\max$  of  $p[mid + 1 \cdots high]$ )

# Stock Investment

## How does the conquer part work?

1. The optimal solution to  $p[low \cdots mid]$
2. The optimal solution to  $p[mid + 1 \cdots high]$
3.  $\operatorname{argmax}\{p(j) - p(i)\}$  where  $low \leq i \leq mid$  and  $mid + 1 \leq j \leq high$  (**equivalent to** finding the index of  $\min$  of  $p[low \cdots mid]$  and that of  $\max$  of  $p[mid + 1 \cdots high]$ )

## How to describe the design of an algorithm in English

- ▶ First point out what strategy/method used, e.g. divide-and-conquer, binary search.
- ▶ Separate paragraphs if necessary, e.g. branches.
- ▶ Bullet-point format is also a good practice.

# Stock Investment: Pseudocode

STOCK-INVESTMENT( $p, low, high$ )

```
1  // Base case: only one element
2  if  $low == high$ 
3      return  $low, high, 0$ 
4  else  $mid = low + \lfloor (high - low) / 2 \rfloor$ 
5       $leftBuy, leftSell, leftGain = \text{STOCK-INVESTMENT}(p, low, mid)$ 
6       $rightBuy, rightSell, rightGain = \text{STOCK-INVESTMENT}(p, mid, high)$ 
7      // Find the index of  $\min(\text{leftArray})$ 
8       $crossBuy = \text{MIN-INDEX}(p, low, mid)$ 
9      // Find the index of  $\max(\text{rightArray})$ 
10      $crossSell = \text{MAX-INDEX}(p, mid + 1, high)$ 
11      $crossGain = p[crossSell] - p[crossBuy]$ 
12     if  $\max(leftGain, rightGain, crossGain) \leq 0$ 
13         return "no gain"
14     elseif  $leftGain \geq rightGain$  and  $leftGain \geq crossGain$ 
15         return  $leftBuy, leftSell, leftGain$ 
16     elseif  $rightGain \geq leftGain$  and  $rightGain \geq crossGain$ 
17         return  $rightBuy, rightSell, rightGain$ 
18     else return  $crossBuy, crossSell, crossGain$ 
```

# Pseudocode Conventions <sup>2</sup>

1. Pseudocode should be self-explanatory, only a few comments might be needed.
2. Indentation/end indicates block structure (curly braces are not good practice).
3. Array index starts at 1 and ends at  $n$ , i.e.  $A[i]$  is the  $i$ th element of the array  $A$ .
4. Use subroutine if the main procedure is too long, e.g. merge, MaxXingSubarray

---

<sup>2</sup>See pp.[20-22] in the textbook for pseudocode conventions

## Stock Investment: Time complexity

$$\begin{aligned}T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) \\&= \Theta(n \log n)\end{aligned}$$

1.  $2T(\frac{n}{2})$ : the first two optimal solutions
2.  $\Theta(n)$ : the third solution is equivalent to finding the `min` and `max`, e.g. linear scan
3.  $\Theta(1)$ : compare the results of three solutions