

ACOL106: Data Structures and Algorithms

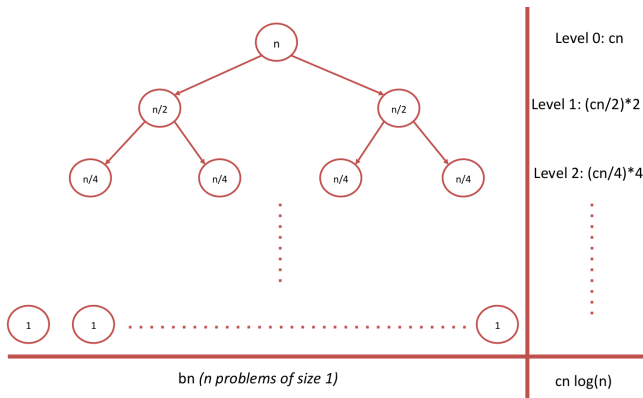
Ragesh Jaiswal, IITD

Introduction

Digression: Binary Search → Solving Recurrence

- Another way of viewing unrolling of the recursion is **Recurrence Trees**.
- For example, consider the following recurrence relation:

$$T(n) \leq 2 \cdot T(n/2) + c \cdot n \text{ for all } n > 1; \quad T(1) \leq b$$



Introduction

Digression: Binary Search $\rightarrow$ Solving Recurrences

- Solve: $T(n) \leq 2 \cdot T(n/2) + cn^2; T(1) \leq c$

Introduction

Digression: Ternary Search

- Solve: $T(n) \leq 2 \cdot T(n/2) + cn^2; T(1) \leq c$
- Solve: $T(n) \leq T(n/3) + c; T(1) \leq b$

Introduction

Digression: Binary Search → Recursive Functions

Problem

Multiplying two n -bit numbers: Given two n -bit numbers, A and B , Design an algorithm to output $A \cdot B$.

Introduction

Digression: Binary Search → Recursive Functions

Problem

Multiplying two n -bit numbers: Given two n -bit numbers, A and B , Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication?

Introduction

Digression: Binary Search → Recursive Functions

Problem

Multiplying two n -bit numbers: Given two n -bit numbers, A and B , Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication? $O(n^2)$
- Is there a faster algorithm?

DIVIDE AND CONQUER

Divide and Conquer

- Break a problem into similar subproblems
- Solve each subproblem recursively
- Combine

Above its weight class

- Divide and conquer is a very simple idea
- But it has far more than its share of the *miraculous algorithms*
- Examples
 - Strassen Matrix Multiplication
 - Karatsuba multiplication
 - Fast Fourier Transform
 - Linear time select

Multiplying Binomials

- if you want to multiply two binomials
- $(ax + b)(cx + d) = acx^2 + adx + bcx + bd$
- It **requires ?** 4 multiplications. ac, ad, bc, bd

Multiplying Binomials

- if you want to multiply two binomials
- $(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd$
- It requires 4 multiplications. ac, ad, bc, bd
- If we assume that addition is cheap (has short runtime.)
Then we can improve this by only doing 3 multiplications:
 $ac, bd, (a + b)(c + d)$

Multiplying Binomials

- Reducing the number of multiplications from 4 to 3 may not seem very impressive when calculating asymptotics.
- If this was only a part of a bigger algorithm, it may be an improvement.

Multiplying Binary numbers

$$\begin{array}{r} \\ \\ \\ \\ + \\ \hline 1 \end{array}$$

(1101 times 1)
(1101 times 1, shifted once)
(1101 times 0, shifted twice)
(1101 times 1, shifted thrice)

(binary 143)

Divide and conquer multiply

- Say we want to multiply 10100110 and 10110011
- How can we divide the problem into sub-problems?
- Remember, we want much smaller sub-problems

Multiplying large binary numbers

- $10100110 = 166 = 1010 * 2^4 + 0110 = 10*16 + 6$
- $10110011 = 179 = 1011*2^4 + 0011 = 11*16 + 3$
- $10100110*10110011 = (10*16+6)(11*16+3) = 110*256 + 6*11*16 + 3*10*16 + 3*6$

Multiplying Binary numbers (DC)

- Suppose we want to multiply two n -bit numbers together where n is a power of 2.
- One way we can do this is by splitting each number into their left and right halves which are each $n/2$ bits long

• $x =$	x_L	x_R
• $y =$	y_L	y_R

Multiplying Binary numbers (DC)

- Suppose we want to multiply two n -bit numbers together where n is a power of 2.
- One way we can do this is by splitting each number into their left and right halves which are each $n/2$ bits long
- $x = 2^{n/2}x_L + x_R$
- $y = 2^{n/2}y_L + y_R$

Multiplying Binary numbers (DC)

$$x = 2^{n/2}x_L + x_R$$

$$y = 2^{n/2}y_L + y_R$$

- $xy = \left(2^{\frac{n}{2}}x_L + x_R\right)\left(2^{\frac{n}{2}}y_L + y_R\right)$
- $xy = 2^n x_L y_L + 2^{\frac{n}{2}}(x_L y_R + x_R y_L) + x_R y_R$

Algorithm multiply

- function **multiply** (x,y):
- *Input*: n-bit integers x and y
- *Output*: the product xy
 - If $n=1$: return xy
 - x_L, x_R and y_L, y_R are the left-most and right-most $n/2$ bits of x and y, respectively.
 - $P_1 = \mathbf{multiply}(x_L, y_L)$
 - $P_2 = \mathbf{multiply}(x_L, y_R)$
 - $P_3 = \mathbf{multiply}(x_R, y_L)$
 - $P_4 = \mathbf{multiply}(x_R, y_R)$
 - return $(P_1 * 2^n + (P_2 + P_3) * 2^{\frac{n}{2}} + P_4)$

Algorithm

- Runtime analysis:
 - Let $T(n)$ be the runtime of the multiply algorithm.
 - Then $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

End