CSE331 Introduction to Algorithms
Lecture 6: More Divide & Conquer Algorithms

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

- In this lecture, I will present two more divide and conquer algorithms: binary search and Karatsuba's algorithm for integer multiplication.
- (Binary search is not always considered to be a divide and conquer algorithm.)
- **Reference** for this lecture:
  - Section 5.5 of Algorithm Design by Kleinberg and Tardos.

Searching in a Sorted Array

Problem (Searching in a sorted array)

Given a sorted array $A[1 \ldots n]$ of numbers and a query number x, find the position of x in A. In particular, if there exists i such that $x = A[i]$, return i, and otherwise, return $\textsc{NotFound}$.

- Can be solved in $\Theta(n)$ by checking whether $x = A[i]$ for all i.
- This is called *brute force search*, because all possible answers are checked.
- It is also called *linear search*.
- Next slides: We introduce *binary search*, which runs in $\Theta(\log n)$ time.

Binary Search

- Example 1: $x = 4$ and $A[1 \ldots 10] = [1, 2, 4, 6, 7, 9, 12, 13, 15, 19]$

1	2	4	6	7	9	12	13	15	19

- Compare x with the middle element $A[5] = 7$.

1	2	4	6	7	9	12	13	15	19

- As $x < A[5]$, recurse on $A[1 \ldots 4]$

1	2	4	6	7	9	12	13	15	19

Binary Search

- Compare x with the middle element $A[2] = 2$.

1	2	4	6	7	9	12	13	15	19

- As $x > A[2]$, recurse on $A[3 \ldots 4]$.

1	2	4	6	7	9	12	13	15	19

- Compare x with the middle element $A[3] = 4$.

1	2	4	6	7	9	12	13	15	19

- As $x = A[3] = 4$, we return 3.

Binary Search

- Example 2: $x = 10$ and $A[1 \dots 10] = [1, 2, 4, 6, 7, 9, 12, 13, 15, 19]$

1	2	4	6	7	9	12	13	15	19

- Compare x with the middle element $A[5] = 7$.

1	2	4	6	7	9	12	13	15	19

- As $x > A[5]$, recurse on $A[6 \dots 10]$

1	2	4	6	7	9	12	13	15	19

- Compare x with the middle element $A[8] = 13$.

1	2	4	6	7	9	12	13	15	19

Binary Search

- As $x < A[8]$, recurse on $A[6 \ldots 7]$.

1	2	4	6	7	9	12	13	15	19

- Compare x with the middle element $A[6] = 9$.

1	2	4	6	7	9	12	13	15	19

- As $x > A[6]$, recurse on $A[7]$.

1	2	4	6	7	9	12	13	15	19

- As $x \neq A[7]$, return $\textsc{NotFound}$.

Pseudocode

Pseudocode of Binary Search

procedure $\textsc{BinarySearch}(A, \ell, r, x)$ ▷ search for x in $A[\ell \dots r]$

  if $\ell > r$ **then** ▷ test if array is empty

    return $\textsc{NotFound}$

  $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$

  if $x = A[m]$ **then**

    return m

  if $x < A[m]$ **then**

    return $\textsc{BinarySearch}(A, \ell, m-1, x)$

  else

    return $\textsc{BinarySearch}(A, m+1, r, x)$

Analysis

- The recursion tree is a single path.
- At each level, we spend time c for some constant c.
- Intuition:
  - The recursion tree is similar to a single path to a leaf in the recursion tree of MERGE SORT.
  - So there are about $\log n$ levels.
  - So the running time is $\Theta(\log n)$.
- More careful analysis:
  - The size of the current subarray is at most half the size of its parent.
  - So it takes at most $1 + \lceil \log n \rceil$ steps to reach an empty subarray.
  - So the worst-case running time is $\Theta(\log n)$.

Iterative Binary Search

- Binary search also has a simple iterative version, i.e. non-recursive.

Pseudocode of Iterative Binary Search

```
procedure BinarySearch(A[1...n], x)        ▷ search for x in array A
    ℓ ← 1, r ← n                           ▷ search interval bounds
    while ℓ ⩽ r do
        m ← ⌊(ℓ+r)/2⌋
        if A[m] > x then
            r ← m − 1                      ▷ search on the left
        else if A[m] < x then
            ℓ ← m + 1                      ▷ search on the right
        else
            return m                       ▷ x = A[m]
    return NotFound
```

Iterative Binary Search

- How does it compare with the recursive version?
- How to prove correctness?

See lecture notes.

Integer Multiplication

$$
\begin{array}{cccc}
 & & 1 & 2 \\
\times & 1 & 3 & \\
\hline
 & & 3 & 6 \\
 & 1 & 2 & \\
\hline
 & 1 & 5 & 6
\end{array}
\qquad\qquad
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 0 \\
\times & & & & 1 & 1 & 0 & 1 \\
\hline
 & & & & 1 & 1 & 0 & 0 \\
 & & & 0 & 0 & 0 & 0 & \\
 & & 1 & 1 & 0 & 0 & & \\
 & 1 & 1 & 0 & 0 & & & \\
\hline
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0
\end{array}
$$

- The *long multiplication* algorithm taught in primary school.
- Also works in binary.
- Running time: For two n-digits (or n-bits) numbers, takes $\Theta(n^2)$ time.

Integer Multiplication

- We try to improve it using divide & conquer.
- We use binary numbers, and assume that n is a power of 2.
- Idea: $10011011_2 = 2^4 \times 1001_2 + 1011_2$
- Any n-bits number x can be split into two $n/2$ bits numbers x_1,x_0

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
x_1 &= x \text{ div } 2^{n/2} \\
x_0 &= x \text{ mod } 2^{n/2}
\end{aligned}
$$

- Let $y = 2^{n/2} \cdot y_1 + y_0$ be another n-bit number split in the same way.

Integer Multiplication

$$
\begin{aligned}
xy &= (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) \\
&= 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + x_0 y_1) + x_0 y_0
\end{aligned}
$$

- We reduce the product of two n-digits numbers to computing 4 products of two $n/2$-digit numbers, and two additions.

Integer Multiplication

Pseudocode

```
1: procedure MULTIPLY(x, y)                                ▷ n-bit integers
2:     if n = 1 then                                       ▷ base case
3:         return xy
4:     write x = 2^{n/2} · x_1 + x_0 and y = 2^{n/2} · y_1 + y_0
5:     z_2 ← MULTIPLY(x_1, y_1)
6:     z_1 ← MULTIPLY(x_1, y_0)+MULTIPLY(x_0, y_1)
7:     z_0 ← MULTIPLY(x_0, y_0)
8:     return 2^n z_2 + 2^{n/2} z_1 + z_0
```

- Analysis:
  - Line 4: $\Theta(n)$
  - Line 5,7: $T(n/2)$
  - Line 6: $2T(n/2) + \Theta(n)$ because an addition takes $\Theta(n)$.
  - Line 8: $\Theta(n)$

Integer Multiplication

- Therefore
$$T(n) = 4T(n/2) + \Theta(n).$$
- $T(n) = \Theta(n^2)$.
- Proof: See lecture notes.
- Intuition:
  - Every pair of bits a b from x and y, respectively, appears in a product $x_i y_j$.
  - Example: $1\underline{1}00\,1101 \times 1001\,1\underline{0}11$
  - The pair $\underline{1}, \underline{0}$ appears in the product $x_1 y_0$
  - So in the end, $\underline{1}, \underline{0}$ will appear as a base case of our recursive computation, i.e. a leaf of the recursion tree.
  - $\Rightarrow$ there are $\Omega(n^2)$ steps in this calculation.
- $\Theta(n^2)$ is not a good time bound because it is the same as the standard long multiplication algorithm.
- Conclusion: Divide & conquer *does not always help*.

Integer Multiplication

- The relation $T(n) = 4T(n/2) + \Theta(n)$ comes from the fact that we make 4 recursive calls to problems of size $n/2$.
- Can we do better, i.e. make only 3 recursive calls?
- Yes: remember

$$\begin{aligned} x &= 2^{n/2} \cdot x_1 + x_0 \\ y &= 2^{n/2} \cdot y_1 + y_0 \\ xy &= 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + x_0 y_1) + x_0 y_0 \end{aligned}$$

- Let $p = (x_1 + x_0)(y_1 + y_0)$, then

$$x_1 y_0 + x_0 y_1 = p - x_1 y_1 - x_0 y_0.$$

- So we need only 3 recursive multiplications: p, $x_1 y_1$ and $x_0 y_0$.

Integer Multiplication

Karatsuba's algorithm for integer multiplication

procedure $\textsc{Multiply}(x, y)$ ▷ *n*-bit integers
  if $n = 1$ **then** ▷ base case
    return xy
  write $x = 2^{n/2} \cdot x_1 + x_0$ and $y = 2^{n/2} \cdot y_1 + y_0$
  $z_2 \leftarrow \textsc{Multiply}(x_1, y_1)$
  $z_0 \leftarrow \textsc{Multiply}(x_0, y_0)$
  $p \leftarrow \textsc{Multiply}(x_1 + x_0, y_1 + y_0)$
  $z_1 \leftarrow p - z_0 - z_2$
  return $2^n z_2 + 2^{n/2} z_1 + z_0$

Integer Multiplication

- Analysis of Karatsuba's algorithms: as an addition or subtraction of two n-bit numbers takes $\Theta(n)$ time, and there are 3 recursive computations on problems of size $n/2$,

$$T(n) = 3T(n/2) + \Theta(n).$$

- As we will see in next Lecture, it implies:

Theorem

Karatsuba's algorithm for multiplying two n-bit integers takes $\Theta(n^{\log_2 3})$ time.

- $\log_2 3 \approx 1.59$, so $T(n) = o(n^2)$. This algorithm is *subquadratic*.

Integer Multiplication: Minor Issues

What if n is not a power of 2?

Let m be the smallest power of 2 larger than n, i.e. $m = 2^{\lceil \log n \rceil}$. Then insert $m - n$ zeroes at the beginning of the binary expansion of n. Example:

$$1\,1010_2 = 0001\,1010_2$$

As $n \leqslant m < 2n$, we still have $T(m) = \Theta(m^{\log_2 3}) = \Theta(n^{\log_2 3})$.
This technique is called zero-padding.

Line 7: $x_1 + x_0$ and $y_1 + y_0$ may have $n/2 + 1$ bits, not $n/2$

It can be solved in several ways. (See lecture notes or wikipedia.)