csce350 — Data Structures and Algorithms
Fall 2019 — Lecture Notes: Divide and Conquer

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with occasional supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

5

Divide and conquer algorithms divide the problem into several smaller instances, solve those instances recursively, and combine the results.

2 Aside: A shortcut for solving recurrences

Theorem ("The Master Theorem") If there are constants a, b, and d, for which $T(n) = aT(n/b) + f(n)$ and $f(n) \in \Theta(n^d)$, then:

- If $a < b^d$, $T(n) \in \Theta(n^d)$.
- If $a = b^d$, $T(n) \in \Theta(n^d \log n)$.
- If $a > b^d$, $T(n) \in \Theta(n^{\log_b a})$.

3 Master theorem examples

$$T(n) = 2T(n/2) + n^2$$

$$T(n) = 4T(n/2) + n^2$$

4 Master theorem examples

$$T(n) = 400T(n/7) + n^3 + n + 1$$

$$T(n) = 2T(n/2)$$

5 Master theorem examples

$$T(n) = 2T(n-1) + 1$$

6 Merge sort

Idea: Sort the first half and the second half of the array separately. Then *merge* the results.

7 Merge sort

```
MERGSORT(A[0, ..., n − 1])
  if n ≤ 1 then
    return
  end if
  B ← copy of A[0, ..., ⌊n/2⌋]
  C ← copy of A[⌊n/2⌋ + 1, ..., n]
  MERGESORT(B)
  MERGESORT(C)
  MERGE(B, C, A)
```

8 Merge

```
MERGE(B[0, ..., p − 1], C[0, ..., q − 1], A[0, ..., p + q − 1])
  i ← 0
  j ← 0
  k ← 0
  while i < p and j < q do
    if B[i] < C[j] then
      A[k] ← B[i]
      i ← i + 1
    else
      A[k] ← C[j]
      j ← j + 1
    end if
    k ← k + 1
  end while
  if i ¡ p then
    Copy B[i, ..., p − 1] to A[k, ..., p + q − 1]
  else
    Copy C[i, ..., q − 1] to A[k, ..., p + q − 1]
  end if
```

9 Quicksort

Idea: Partition the array. This puts the pivot element it its final home. Then recursively sort the left and right sides.

```
QUICKSORT(A[0, ..., n − 1])
  if n > 1 then
    s ← PARTITION(A[0, ..., n − 1])
    QUICKSORT(A[0, ..., s − 1])
    QUICKSORT(A[s + 1, ..., n − 1])
  end if
```

10 Quicksort with boundary indices

A typical quicksort implementation keeps track of the left and right **boundary indices**, rather than forming subarrays:

```
QUICKSORT(A[0, ..., n − 1], ℓ, r)
  if r − ℓ > 0 then
    s ← PARTITION(A, ℓ, r)
    QUICKSORT(A, ℓ, s − 1)
    QUICKSORT(A, s + 1, r)
  end if
```

11 QuickSort analysis

Best case:

$$C_{\text{best}}(n) = C(n/2) + C(n/2) + n - 1 \in \Theta(n \log n)$$

Worst case:

$$C_{\text{worst}}(n) = C(0) + C(n-1) + n - 1 \in \Theta(n^2)$$

Average case, with input permutations equally likely:

$$C_{\text{avg}}(n) \approx 1.38n \log n \in \Theta(n \log n)$$

12 Multiplication of large numbers

Observation: An `int` variable typically stores values only up to 2 billion or so.

Question: What if we need to do exact arithmetic on numbers that are too big to fit into an integer variable?

One answer: Represent big numbers as arrays of base-10 digits.

(or base-16, or base-256, or if you are extremely careful, base-`INT_MAX`, or...)

13 Arrays of digits

Some things are easy to do with this representation.

Can you **add** two arrays of digits?

Can you **subtract** two arrays of digits?

Can you **multiply** an array of digits by a **power of ten**?

14 Back to multiplication: Grade school algorithm

Brute force: To compute $a \times b$, multiply each digit of b times a, and then add the partial products.

$$\begin{array}{cccccccc} & & & 1 & 2 & 3 & 4 \\ & & \times & 5 & 6 & 7 & 8 \\ \hline & & & 9 & 8 & 7 & 2 \\ & & 8 & 6 & 3 & 8 & 0 \\ & 7 & 4 & 0 & 4 & 0 & 0 \\ 6 & 1 & 7 & 0 & 0 & 0 & 0 \\ \hline 7 & 0 & 0 & 6 & 6 & 5 & 2 \end{array}$$

15 Can we do better?

Suppose we have two n-digit numbers a and b.

Idea 1: Split each number into two equal chunks.

$$m = \lfloor n/2 \rfloor$$

$$a = 10^m a_1 + a_0$$

$$b = 10^m b_1 + b_0$$

16 Simplify?

Idea 2: Do some algebra.

$$\begin{aligned} ab &= (10^m a_1 + a_0)(10^m b_1 + b_0) \\ &= 10^{2m} a_1 b_1 + 10^m a_0 b_1 + 10^m a_1 b_0 + a_0 b_0 \\ &= 10^{2m} a_1 b_1 + 10^m (a_0 b_1 + a_1 b_0) + a_0 b_0 \\ &= 10^{2m} a_1 b_1 + 10^m (a_0 b_1 + a_1 b_0 + a_1 b_1 + a_0 b_0 - a_1 b_1 - a_0 b_0) + a_0 b_0 \\ &= 10^{2m} a_1 b_1 + 10^m (a_0 (b_0 + b_1) + a_1 (b_0 + b_1) - a_1 b_1 - a_0 b_0) + a_0 b_0 \\ &= 10^{2m} a_1 b_1 + 10^m ((a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0) + a_0 b_0 \end{aligned}$$

17 Success!

Idea 3: Notice the repetition. Let

$$\begin{aligned} c_2 &= a_1 b_1 \\ c_0 &= a_0 b_0 \\ c_1 &= (a_0 + a_1)(b_0 + b_1) - c_2 - c_0 \end{aligned}$$

so that

$$ab = 10^{2m} c_2 + 10^m c_1 + c_0.$$

18 Karatsuba multiplication

```
KARATSUBA(a, b)
  if a and b are small then
    return a · b
  else
    m ← ⌊max(digits in a, digits in b)/2⌋
    a_0 ← last m digits in a
    a_1 ← remaining digits in a
    b_0 ← last m digits in b
    b_1 ← remaining digits in b
    c_2 ← KARATSUBA(a_1, b_1)
    c_0 ← KARATSUBA(a_0, b_0)
    c_1 ← KARATSUBA(a_1 + a_0, b_1 + b_0) − c_2 − c_0
    return 10^{2m} c_2 + 10^m c_1 + c_0
  end if
```

19 Karatsuba analysis

Three recursive calls, each on problems of size $n/2$.

All of the other work (additions, subtractions, multiplications by powers of 10) takes $\Theta(n)$ time.

Run time recurrence:

$$M(n) = 3M(n/2) + \Theta(n)$$

Solve via the Master Theorem:

$$M(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.58}).$$