CS 6505 (Fall 2017)

Computability & Algorithms

Prof. M. Mihail
Georgia Institute of Technology

LaTeXer: W. KONG
http://wwkong.github.io

Last Revision: September 13, 2017

Table of Contents

These notes are currently a work in progress, and as such may be incomplete or contain errors.

Acknowledgments:

Special thanks to *Michael Baker* and his LaTeX formatted notes. They were the inspiration for the structure of these notes.

Abstract

The purpose of these notes is to provide the reader with a secondary reference to the material covered in CS 6505.

Administrative

Instructor: Milena Mihail (mihail@cc.gatech.edu)
Content: $\frac{2}{3}$ Algorithm Design, $\frac{1}{3}$ Hardness
Textbook(s): [1] Algorithms by DasGupta, Papadimitriou, Vazirani [2] Algorithms by Cormen, Leiserson, Rivest, Stein [3] Algorithms by Kleinberg, Tardos
Marking Scheme: Final 25%, 3 Quizzes 20% each, 15% Homework
Timing: Quiz 1 is Sept. 27, Quiz 2 is Oct. 25, Drop date is Oct. 27, Quiz 3 is Nov. 20, Last day of class is Dec. 4/5, Final Exam is Dec. 11.

1 Introduction

Overview of the course:

- Divide and Conquer
- Dynamic Programming
- Graph Algorithms
  - DFS, connectivity, bi-connectivity, strong connectivity
  - Minimum spanning trees
  - Shortest paths
  - Max flow /min cut problem
- $\mathcal{NP}$-complete problems
  - Approximation methods (e.g. TSP)
  - Primal-dual methods
- Special Topics
  - Number theoretic algorithms
  - Page rank (information retrieval, spectral methods)

1.1 Recursion Theory

Sorting Algorithms

Algorithm 1 Bubblesort($a_1, ..., a_n$)?

```
1: if $n \geq 1$ then
2:     max := $a_1$
3:     for $i := 1$ to $(n-1)$ do
4:         if $a_i > a_{i+1}$ then swap($a_i, a_{i+1}$)
5:         end if
6:     end for
7: end if
```

This has $\mathcal{O}(n^2)$ complexity.

Recursive Thinking

e.g. Consider $T(n) = 2T\left(\frac{n}{2}\right) + n$ and $T(1) = 1$. The call of $T(n)$ produces a number which is $\mathcal{O}(n \log n)$. This is the basis of MergeSort.

Solution to Basic Master-Slave Recurrence

Algorithm 2 RecursiveSort$(a_1, ..., a_n)$ or MergeSort$(a_1, ..., a_n)$

```
1: RecursiveSort($a_1, ..., a_{\frac{n}{2}}$)
2: RecursiveSort($a_{\frac{n}{2}}, a_n$)
3: Combine($a_1, ..., a_n$)
```

Consider the relationship

$$T(n) = aT\left(\frac{n}{b}\right) + cn$$

for positive integers a, b, c. Assume n is a power of b, say $n = b^{\log_b n}$. Let us say $T(1) = 1$.

Theorem 1.1. *(Master Theorem) We have*

$$T(n) = \begin{cases} \mathcal{O}(n \log n), & \text{if } a = b \\ \mathcal{O}(n), & \text{if } a < b \\ \mathcal{O}\left(n^{\log_b a}\right), & \text{if } a > b. \end{cases}$$

Proof. Solve by repeated substitution. We have

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + cn \\ &= a\left(aT\left(\frac{n}{b^2}\right) + \frac{cn}{b}\right) + cn \\ &= a^2 T\left(\frac{n}{b^2}\right) + \left(1 + \frac{a}{b}\right) cn \\ &= a^2\left(aT\left(\frac{n}{b^3}\right) + \frac{cn}{b^2}\right) + \left(1 + \frac{a}{b}\right) cn \\ &= a^3 T\left(\frac{n}{b^3}\right) + \left(1 + \frac{a}{b} + \frac{a^2}{b^2}\right) cn \\ &\vdots \\ &= a^k T\left(\frac{n}{b^k}\right) + \left(1 + \frac{a}{b} + \frac{a^2}{b^2} + ... + \frac{a^k}{b^k}\right) cn \end{aligned}$$

and at termination,

$$T(n) = a^m + cn \sum_{i=0}^{m-1} \left(\frac{a}{b}\right)^i, \text{ where } m = \log_b n.$$

If $a = b$ then

$$T(n) = a^{\log_a n} + cn \log_a n = n + cn \log_a n = \mathcal{O}(n \log n).$$

If $a > b$ then

$$\begin{aligned} T(n) &= a^{\log_b n} + cn\left(\frac{(a/b)^{\log_b n} - 1}{(a/b) - 1}\right) \\ &= a^{\log_b n} + bcn\left(\frac{a^{\log_b n} - b^{\log_b n}}{b^{\log_b n}[a-b]}\right) \\ &= a^{\log_b n} + \frac{bc}{a-b}\left(a^{\log_b n} - n\right) \\ &= a^{\log_b n}\left(1 + \frac{bc}{a-b}\right) - \frac{bcn}{a-b} \\ &= b^{\log_b a \cdot \log_b n}\left(1 + \frac{bc}{a-b}\right) - \frac{bcn}{a-b} \\ &= n^{\log_b a}\left(1 + \frac{bc}{a-b}\right) - \frac{bcn}{a-b} \\ &= \mathcal{O}\left(n^{\log_b a}\right). \end{aligned}$$

The last case, $a < b$, uses the same computations, but we have

$$\begin{aligned}
T(n) &= a^{\log_b n} + cn\left(\frac{1-(a/b)^{\log_b n}}{1-(a/b)}\right)\\
&= a^{\log_b n} - bcn\left(\frac{a^{\log_b n} - b^{\log_b n}}{b^{\log_b n}[a-b]}\right)\\
&= a^{\log_b n} - \frac{bc}{a-b}\left(a^{\log_b n} - n\right)\\
&= a^{\log_b n}\left(1 - \frac{bc}{a-b}\right) + \frac{bcn}{a-b}\\
&= b^{\log_b a \cdot \log_b n}\left(1 - \frac{bc}{a-b}\right) + \frac{bcn}{a-b}\\
&= n^{\log_b a}\left(1 - \frac{bc}{a-b}\right) + \frac{bcn}{a-b}\\
&= \mathcal{O}(n).
\end{aligned}$$

Integer Arithmetic

Summary: □

- Addition and subtraction are $\mathcal{O}(n)$ which is obvious.
- Naive Multiplication is $\mathcal{O}(n^2)$

Recall that bubble sort takes $\mathcal{O}(n^2)$ time, so let us try to do multiplication recursively with this in mind. Suppose that x has binary representation

$$x = \underbrace{a_n a_{n-1} \dots a_{\frac{n}{2}}}_{A}\,\underbrace{a_{\frac{n}{2}-1} \dots a_0}_{B} = A \cdot 2^{\frac{n}{2}} + B$$

where both A and B have $\frac{n}{2}$ bits, and y has binary representation

$$y = \underbrace{b_n b_{n-1} \dots b_{\frac{n}{2}}}_{C}\,\underbrace{b_{\frac{n}{2}-1} \dots b_0}_{D} = C \cdot 2^{\frac{n}{2}} + D$$

where both C and D have $\frac{n}{2}$ bits. Observe that

$$x \cdot y = AC \cdot 2^n + (AD + BC)2^{\frac{n}{2}} + BD$$

Now note that

$$(A + B) \times (C + D) - AC - BD = AD + BC$$

and hence we may come up with the algorithm:

Algorithm 3 Multiply(x, y)

```
1: $z_1 :=$ multiply$(A, C)$
2: $z_2 :=$ multiply$(B, D)$
3: $z_3 :=$ multiply$(A + B, B + D)$
4: $z_4 := z_3 - z_1 - z_2$
5: $z := z_1 2^n + z_4 2^{\frac{n}{2}} + z_2$
6: **return** $z$
```

This has the recursion equation

$$T(n) = 3T\left(\frac{n}{2}\right) + cn$$

which has complexity $\mathcal{O}(n^{\log_3 2})$. It is possible to reduce this to a power of $\log_6 11$ (Toom-Cook)

Tower of Hanoi

(0) With m discs (from smallest to largest going down) on one plot (plot 1), and three plots, you must move discs from plot 1 to plot 3.

(1) You can only move one disc at a time.

(2) The only disk that can be moved is the top one.

(3) You can only move smaller disks on top of larger ones.

Firstly, can we solve this problem?

Proof by Recursion

* Base case of $m = 1$ discs is trivial.

* Suppose we can solve it with $(n-1)$ discs. Move the $(n-1)$-unit tower to the second plot and move the base unit to the third plot. Move the $(n-1)$ unit tower to the third plot.

Using this scheme, we have the recurrence

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

which doesn't fit our Master Theorem exactly, but we can solve it via **substitution**:

$$\begin{aligned} T(n) &= 2\left[2T(n-2)+1\right]+1 \\ &= 4T(n-2)+2+1 \\ &= 2^2T(n-2)+2^1+2^0 \\ &\vdots \\ &= \sum_{i=0}^{n-1} 2^i \\ &= \frac{2^n-1}{2-1} \\ &= 2^n - 1. \end{aligned}$$

So we are left with a $\mathcal{O}(2^n)$ algorithm, which is really slow but we don't have any other algorithm. Note that

$$T(64) = 2^{64} - 1 \approx 584 \text{ billion seconds}$$

but the age of the universe is 14 billion years.

Binary Search

This has the recurrence equation

$$\begin{aligned} T(m) &= 1 + T\left(\frac{m}{2}\right) \\ &= 1 + 1 + T\left(\frac{m}{4}\right) \\ &\vdots \\ &= \log_2(m) + T\left(\frac{m}{2^{\log m}}\right) \\ &= \log_2 m + T(1) \\ &= \mathcal{O}(\log_2 n). \end{aligned}$$

Max and Min

Naively, we can calculate the max and min separately (each taking $m-1$ comparisons) to get $2m-2$ comparisons in total.

A better scheme is to compare call max/min via a divide and conquer strategy for a total of $\frac{3m}{2} - 2$ comparisons which is better than $\sim 2m$.

Toom-Cook Multiplication

N is a power of 3, and a and b are N-bit binary numbers. We want $c = a \cdot b$.

Step 1. Define

$$a = a_2 \cdot 2^{2N/3} + a_1 \cdot 2^{N/3} + a_0$$
$$b = b_2 \cdot 2^{2N/3} + b_1 \cdot 2^{N/3} + b_0$$

where a_i, b_i are $\frac{N}{3}$ bit binary numbers.

Step 2.Consider the polynomials

$$a(x) = a_2 x^2 + a_1 x + a_0$$
$$b(x) = b_2 x^2 + b_1 x + b_0$$

where we want $c(x) = a(x)b(x)$, i.e.

$$c(x) = c_4 x^4 + c_3 x^3 + ... + c_1 x + c_0.$$

Step 3. Every polynomial of degree n can be determined by $(n+1)$ sample points. Solving the equations

$$c(k) = a(k)b(k)$$

for $k = -2, -1, 0, 1, 2$, gives a recurrence of

$$T(N) = 5T\left(\frac{N}{3}\right) + cN = \mathcal{O}\left(N^{\log_3 5}\right).$$

* We could also use higher degree polynomials, i.e.

$$a(x) = a_5 x^5 + ... + a_1 x + a_0$$
$$b(x) = b_5 x^5 + ... + b_1 x + b_0$$

where the coefficients are $\frac{N}{5}$ bit binary numbers. This scheme gives

$$T(N) = 11T\left(\frac{N}{6}\right) + cN = \mathcal{O}\left(N^{\log_6 11}\right).$$

Fourier Transforms

a is a number with n bits:

$$a = a_{n-1} x^{n-1} + ... + a_1 x + a_0.$$

Transform it to a degree $(n-1)$ polynomial with coefficients 0 or 1. Take a long number and break it down to single terms instead of playing this game.

1.2 Recursive Algorithms

MergeSort

Suppose that we want an $\mathcal{O}(n \log n)$ sorting algorithm and we do not know where to start. We know that

$$T(n) = 2T\left(\frac{n}{2}\right) + cn = \mathcal{O}(n \log n).$$

Approach

Try to reduce sorting an array of size n to sorting two arrays of size $\frac{n}{2}$ and some linear work to combine. This is exactly MergeSort (from before)! Hence, the complexity of MergeSort is $\mathcal{O}(n \log n)$.

Finding the Median

Consider the following algorithm:

KSelect

Input: n distinct unsorted $a_1, a_2, ..., a_n$ and integer k where $1 \leq k \leq n$

Output: the k^{th} smallest element a_i

Define:

$$X = \{a_j : a_j \leq a_i\}$$
$$Y = \{a_j : a_j > a_i\}$$

where $|X| = k$ and $|Y| = n - k$.

Here is the algorithm:

Algorithm 4 KSelect$(a_1, ..., a_n, k)$

```
 1: Find $s \in a_1, ..., a_n$ [See below]
 2: $X := a_j : a_j \leq s$
 3: if $|X| = k$ then
 4:     return$(s)$
 5: else
 6:     $Y = a_j : a_j > s$
 7: end if
 8: if $|X| > k$ then
 9:     KSelect$(X, k)$
10: else
11:     KSelect$(Y, k - |X|)$
12: end if
```

To get s, which we call a splitter, we want to use $\mathcal{O}(n)$ comparisons so that

$$|X| \leq \frac{3n}{4} \qquad \text{and} \qquad |Y| \leq \frac{3n}{4}$$

and find a way to show

$$T(n) \leq T\left(\frac{3n}{4}\right) + cn$$

with $a = 1, b = \frac{4}{3}$, and $a < b$ which is an algorithm with $\mathcal{O}(n)$ running time.

Finding a good splitter

1. Input a sequence of 5-tuples

2. Sort $\frac{n}{5}$ 5-tuples [this is $\mathcal{O}(n) \sim \bar{c}n$]

3. $s :=$KSelect$(a_3, a_8, a_{13}, ..., n_{n-2}, \frac{n}{10})$

** Note that in the sorted array:

$$\begin{array}{ccccccc}
\hat{a}_1 & \hat{a}_6 & & & & & \hat{a}_{n-4} \\
\vdots & \vdots & & & & & \vdots \\
\hat{a}_3 & \hat{a}_8 & \cdots & s & \cdots & & \hat{a}_{n-2} \\
\vdots & \vdots & & & & & \vdots \\
\hat{a}_5 & \hat{a}_{10} & & & & & \hat{a}_n
\end{array}$$

where all elements are increasing left to right and up to down, let A be the elements (smaller than s) in the top left quadrant (pivoted by s) and B be the elements in the bottom right quadrant (larger than s). So both A and B contain $\frac{1}{4}$ of the elements.

Now $|Y|$ does not contain A and $|X|$ does not contain B. Hence,

$$|X| \leq \frac{3n}{4} \qquad \text{and} \qquad |Y| \leq \frac{3n}{4}.$$

** Find a good splitter will take $T\left(\frac{n}{5}\right) + \bar{c}n$ steps and hence KSelect($\cdot$) will have complexity

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + cn.$$

Claim 1.1. There exists $c_0 > 0$ such that $T(n) \leq c_0 n$ for all n where $T(n)$ the complexity of KSelect($\cdot$).

Proof. Inductive, using strong induction. The base case is trivial. Suppose that $\exists c_0 > 0$ s.t. $T(n') \leq cn'$ for all $n' < n$. We have

$$\begin{aligned} T(n) &= T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + cn \\ &\leq \frac{c_0 n}{5} + \frac{3c_0 n}{4} + cn \\ &= \left(\frac{19c_0}{20} + c\right) n. \end{aligned}$$

Now, we want c_0 such that

$$\frac{19c_0}{20} + c \leq c_0 \iff 20c \leq c_0$$

so we can pick $c_0 = 20c$ and the claim is proved. □

Corollary 1.1. *Apply the above claim to* $T\left(\frac{n}{5}\right)$ *to get*

$$T(n) \leq T\left(\frac{3n}{4}\right) + c_1 n$$

for some $c_1 > 0$.

2 Dynamic Programming

Longest Increasing Subsequence

Input: integers $a_1, ..., a_n$ (not sorted)

Define: a subsequence is $a_{i_1}, a_{i_2}, ..., a_{i_k}$ where $1 \leq i_1 < ... < i_k \leq n$; an increasing subsequence is when $a_{i_1} < ... < a_{i_k}$.

Output: an increasing subsequence of maximal length

(Example)

Input: 5, 2, 8, 6, 3, 9, 7

Recursive form of the solution

Define:

$$\begin{aligned} L(j) &= \text{length of the largest increasing subsequence ending at } a_j \\ &= 1 + \max\{L(i) : i < j, a_i < a_j\} \end{aligned}$$

The search space for $L(j)$ will explode exponentially! No thank you.

Memoization

Bottom-up evaluation. Define

$$\Pi(j) = \text{index prior to } j \text{ in a longest increasing subsequence ending at } a_j$$

Initialization:

```
for $i = 1$ to $n$
$L(i) := 1$
$\Pi(i) :=$ nil
```

Memoization:

```
for $j = 2$ to $n$
for $i = 1$ to $j - 1$
if $a_i < a_j$ then
    if $L(i) + 1 \geq L(j)$ then $\begin{cases} L(j) := L(i) + 1 \\ \Pi(j) = i \end{cases}$
```

Running Time: $\mathcal{O}(n^2)$ which is determined by the double loop of the memoization part. The initialization part is $\mathcal{O}(n)$.

Shortest and Longest Paths in DAGs (directed acyclic graphs)

Note that every DAG has an ordering of vertices $v_1, ..., v_n$ such that all edges $v_i \to v_j \implies i < j$. From the source vertex S to the ending vertex E, finding the shortest path in G is clearly $\mathcal{O}(|V| + |E|)$.

It also turns out that the longest path in G can also be found in $\mathcal{O}(|V| + |E|)$.

ROD-CUTTING

Suppose we have a rod with total length n and lengths $\{l_i\}_{i=1}^n$ with corresponding prices $\{p_i\}_{i=1}^n$ from which we can cut the rod. Where/ how to cut the rod so as to maximize the sum of the resulting pieces?

Recursive Form of Solution

Let

$$\mathbb{P}(k) = \text{the max sum for a piece of length } k.$$

We want $\mathbb{P}(n)$ given that $\mathbb{P}(0) = p_0 = 0$ and $\mathbb{P}(1) = p_1$. Now, for $n, k > 1$,

$$\mathbb{P}(n) = \max_{0 \leq j \leq n-1} \{\mathbb{P}(j) + p_{n-j}\}$$
$$\mathbb{P}(k) = \max_{0 \leq j \leq k-1} \{\mathbb{P}(k-j) + p_{n-j}\}.$$

This will blow up.

Example 2.1. Consider the set up

length l_i	1	2	3	4	5	6	7	8	9	10
price p_i	1	7	8	9	10	12	17	20	32	30

Recursion blows up, so let's try another approach.

Initialization

```
for $j = 0$ to $n$
$\mathbb{P}(j) = p_j$
```

Memoization

```
for $k = 1$ to $n$
for $j = 0$ to $k - 1$
```

$\mathbb{P}(k) = \max\{\mathbb{P}(k), \mathbb{P}(j) + p_{k-j}\}$

<u>End</u>

Return $\mathbb{P}(n)$

and so this scheme is $\mathcal{O}(n^2)$.

<u>Maximum Contiguous Subsequence</u>

Given a sequence $\{a_i\}_{i=1}^n$, we want to find the contiguous subsequence with the largest sum S_k ending in some a_k.

<u>Recursive form of the solution</u>

$S_1 := a_1$ and

$S_k := \max\{S_{k-1} + a_k, a_k\}$

<u>Motel Application</u>

We have stops at $\{a_i\}_{i=1}^n$. If in one day you travel $x \neq 200$, you pay $(x-200)^2$. Define C_i as the minimum cost starting at 0 and ending at i. We want C_n.

<u>Recursive form of the solution</u>

We want C_n. We have $C_0 = 0$ and

$$C_k = \min_{0 \le i \le k-1} \left\{ C_i + ((a_k - a_i) - 200)^2 \right\}$$

<u>Making Change</u>

We have a value v and and unlimited supply of coins with denominations $x_1 < x_2 < ... < x_n$. We want to know if we can make v.

Let $P(k)$ be the the proposition that checks if we can make change for value k. We want $P(k)$.

<u>Initialize</u>: $P(0)$ =YES, $P(x_k)$ =YES for $k = 1, 2, ..., n$ and for all other k, $P(k)$ =NO

<u>Recursive form of the solution</u>:

We have

$$P(k) = \text{YES} \iff \exists 0 \le i < k \text{ s.t. } k - i = x_j$$

for some coin j and $P(i)$ =YES.

<u>K-Select</u>

* Know that you can solve KSelect in $\mathcal{O}(n)$ time.

* <u>Wiggly sorting</u>: given $a_1, ..., a_n$ and n is odd, we want some ordering

$$a_1 < a_2 > a_3 < a_4 > ... > a_n.$$

We can call

$$S = (a_1, ..., a_n, k = \lceil \frac{n}{2} \rceil)$$

which is linear and set X as all elements $< S$ and Y as all elements $\geq S$.

<u>Strong Majority</u>

Use KSelect. *Hint*: In $\mathcal{O}(n)$, you can find the median.

<u>Simple Inversions</u>

Look at Piazza.

<u>Significant Inversions</u>

Hint: Assume $C_1 < C_2 < ... < C_{n/2}$ and $b_1 < b_2 < ... < b_{n/2}$. Then continue with assuming $2C_1 < 2C_2 < ... < 2C_{n/2}$ and $b_1 < b_2 < ... < b_{n/2}$.