Comp 252 Course Notes

(Gautier) Cole Killian

February 25, 2020

This is McGill's undergraduate Comp 252, instructed by Luc Devroye. The formal name for this class is "Honors Algorithms and Datastructures". You can find this and other course notes here: https://colekillian.com/course-notes

Contents

§1 Lecture 01-07

§1.1 Website

http://luc.devroye.org/251.html

§1.2 Textbook

Cormen, Leisarson, Rivest, Steing, Intro to algorithms, 3rd edition.

Definition 1.1. Precise set of instructions acting on a finite input acting on a finite input and halting in finite time and using a finite amount of resources.
i.e. there is no algorithm for calculating all the digits of pi. It would be wrong to say that "my algorithm" has an infinite loop because then it would not halt in finite time.
Algorithms for everything. Need it to create a good spageti bolognese.
(world wars are usually very good for advancing technology (although we don't need another one))

§1.3 Timeline

1940: Turing and his Tape
1950: IBM Ram Model (Random access memory)
CPU constant time access very questionable
1970: Pointer based machines come into fashion
Difference: Unlimited cells. Address calcluations are not permitted / done. IBM you can go to the "next" address, but you cannot do that with the pointer based machine.

§1.4 Time / Complexity of an algorithm

Note 1.2. i, j, k, l, m, n are reserved for small f, g are for functions r, s, t are sort of grouped. t for time x, y, z o can be confused with 0 e do not use this d used for dimension

If algorithm takes finite time and finite resources, the ouput must also be finite.

$$\underbrace{x_1, \ldots, x_n}_{\text{Input}} \Rightarrow \text{Algorithm} \Rightarrow \underbrace{y_1, \ldots, y_m}_{\text{Output}}$$

Model Dependent

§1.5 Models

1. Ram Model: Every standard operationtakes 1 time unit.
2. Bit Model (competitor model cherished by computer theorists): Cost $= 1$ $\forall$ bit operations. (After class understand the cost of adding two binary numbers). Note that addition takes n time.
3. Oracle Model: Cost $= 1$ per use of the oracle.

Example 1.3

Types of oracles

a) Comparison based oracle. Analogy is a scale.

b) Function oracle: $f(x)$

c) Sorting Chip. 3 inputs one output with 6 possible values (better understand why there are 6 possible values, i would have though 8)

4. Cache, communication complexity. Idea: Computers are orders of magnitudes faster than communication lines. Cost is the number of bits in an exchange between two computers. Inside cost at another computer is 0

5. Modern day computers. Memory interacts slowly with cache memory which interacts very quickly with the cpu. Cost of operation between memory and cache memory is 1. Cost of operation between cache memory and cpu is 0. (I think the memory could be thought as hard drive, and cache memory as ram)

§1.6 Time Complexity

1. Worst Case Time:

$$T_n = \max Time_n(x_1, \dots, x_n)$$

2. Average Case Time:

$$T_n = \frac{1}{\#(x_1, \dots, x_n)} \sum_{x_1,\dots,x_n} T_n(x_1, \dots, x_n)$$

When designing an algorithm with the user in mind the average case time is more important.

§1.7 Lambda Symbols

$$O, \Omega, \Theta, o, w, \sim$$

Let a_n, b_n be positive number sequences.

$$a_n = O(b_n) \text{ if } \exists C, n_0 : a_n \leq C \cdot b_n : (\forall n \geq n_0)$$

Example 1.4

If $T_n = O(n)$, then we are guaranteeing that T_n has no more than linear growth.
If $T_n = \Omega(2^n)$, we are guaranteeing that T_n grows at least exponentially.
If $T_n = \Theta(n^2)$, we are guaranteeing that T_n always grows at n^2. More specific information.
$a_n = o(b_n)$ when

$$\lim(a_n/b_n) = 0$$

$a_n = w(b_n)$ when

$$\lim(a_n/b_n) = \infty$$

$a_n =\sim (b_n)$ when

$$0 < \lim(a_n/b_n) < \infty$$

§1.8 Scale

$$\underbrace{\log(n)}_{\text{polylogorithmic}(\log(n))^{\Theta(1)}}, \underbrace{\sqrt{n}, n, n\log(n), n^2,}_{\text{polynomial}n^{\Theta(1)}} \underbrace{2^n, 3^n}_{\text{Exponential}(\Theta(1))^n}, n!, n^n$$

Example 1.5 (Computing the n-th fibonacci number)

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$
$$\text{Assume } x_n \approx c^n$$
$$\Rightarrow c^n = c^{n-1} + c^{n-2}$$
$$c^2 = c + 1 \Rightarrow c = \{\frac{1+\sqrt{5}}{2}, \frac{1-\sqrt{5}}{2}\}$$

Solution Number 2, recursive program
Solution Number 3, dynamic programming
Solution #4, linear algebra exponentiation. Take $O(\log n)$

§2 Lecture 01-09

Trying to compute $x^n = e^{n\log(x)}$.
exp(x, n).

$$\begin{cases} n = 0, & 1 \\ n = 1, & x \\ n > 1, n\%2 = 0, & (\exp(x, n/2))^2 \\ n > 1, n\%2 = 1, & (\exp(x, n/2))^2 * x \end{cases}$$

Let T_n be the time to compute x^n. Let $T_n = \underbrace{1}_{\text{Overhead}} + T_{n/2}$

Divide and conquer approach. Break down probelm into a series of subproblems of smaller size, and then marry the solutions of the subproblems for the final output.

Example 2.1 (Chip Testing, Corman Exercise 4.5)

Problem. Factories cranking out billions of chips, far too many for humans to check by hand.

We are given a tester which takes two chips as an input. Each of these chips give an opinion of the other chip.

$$A----\text{Tester}----B$$

Good chips do not lie, but bad chips cannot be trusted.

Model: Number of uses of a tester (Oracle)

Given: n chips. g represents the set of good chips and B represents the set of bad chips. $|g|+|B|=n$. It is known that $|g|>|B|$.

Wanted: Identify g using the tester $O(n)$ times.

Solution: If we have one good chip, is that very valuable? Absolutely because then you could pass the rest of the chips ($n-1$ tests) through the machine, knowing that it doesn't lie. So the problem is really to find 1 good chip in $O(n)$.

Procedures

1. Take one chip and test it against all other chips. If the vote is greater than or equal to 50%, then it is good. This procedure takes $n-1$ time. Repeating this procedure until success costs pessimistically $(n-1)+(n-2)+\cdots+(n/2-1)$. A.K.A. worst case is $\Theta(n^2)$. $\geq n/2\times n/2$ and $\leq n/2\times n$.

2. If n is even, then we can make $n/2$ pairings. A percentage of these will be GG, a percentage will be BB, and a percentage will be BG/GB.

   Take all BB, BG, and GB, and throw them out. Then take all the GG pairs, and throw out the right side.
   At least of the two chips in each of the pairs that are thrown out must be bad. Therefore $\geq 50\%$ bad.

   Therefore, in the GG section, the goods are bigger than the bads, especially in this section $|g|\geq|B|+2$ (Pigeon Hole Principle). Having them, and we still have that

$$\frac{|g|}{2}\geq\frac{|B|}{2}+1$$

   T_n is the time to solve the problem.

   Worst case: $T_n\leq n/2+T_{n/2}$, and $T_1=0$.

   In case that n is odd, you have an extra $n-1$ test. You do procedure 1 a single time and decide whether or not a single chip is good or bad. Then $T_n\leq\lfloor(\rfloor n/2)+(n-1)+T_{\lfloor n/2\rfloor}\leq 3n/2+T_{\lfloor n/2\rfloor}$

$$T_n\leq\frac{3n}{2}+T_{n/2}\leq\frac{3n}{2}+\frac{3}{2}(\frac{n}{2})+6T_{n/4}\leq\frac{3n}{2}\underbrace{(1+\frac{1}{2}+\frac{1}{4}+\dots)}_{2}+T_1=3n$$

Example 2.2 (Multiplying two n-bit integers)
n bits. (if one of them has less than n bits you just add zeros)

$$\begin{array}{r} 1\,1\,0\,1 \\ \times \quad 1\,0\,1 \\ \hline 1\,1\,0\,1 \\ 1\,1\,0\,1\,\cdot \\ \hline 1\,1\,1\,2\,0\,1 \end{array}$$

Addition can be seen as a $2n \times n$ matrix. Time $= \Theta(n^2)$. n rows, $2n$ wide.

Kanatsuba method.

$$a = \underbrace{a_1}_{n/2 \text{ bits}} \times 2^{n/2} + \underbrace{a_2}_{n/2 \text{ bits}}$$

$$b = \underbrace{b_1}_{n/2 \text{ bits}} \times 2^{n/2} + \underbrace{b_2}_{n/2 \text{ bits}}$$

$$ab = a_1 b_1 \times 2^n + (a_1 b_2 + a_2 b_1) 2^{n/2} + a_2 b_2$$

$$T_n = \underbrace{4T_{n/2}}_{a_1b_1, a_1b_2, a_2b_1, a_2b_2} + \underbrace{\frac{3n}{2}}_{*} + \underbrace{12n}_{+}$$

$$a_1 b_2 + a_2 b_1 = (a_1 - a_2)(b_2 - b_1) + a_2 b_2 + a_1 b_1$$

We already computed $a_2 b_2$ and $a_1 b_1$. Now $T_n = 3T_{n/2} + an$. Constant in front of n will go up, but the $4 \to 3$ makes a huge huge difference. Now

$$T_n = \Theta(n^{\log_2 3})$$

Toom-cook method further improves on this. The game is on after this first break of $\Theta(n^2)$. You can get $T_n \leq 5T_{n/3} + cn$. It involves breaking into chunks of $n/3$ instead of $n/2$.
Best today: $O(n \log n)$. Published last year still being reviewed.

Example 2.3 (Mergesort)
Input is an array A, $A[1], \dots, A[n]$.

$$A' = A[1 - n/2]$$
$$A'' = A[n/2 + 1 - n]$$
$$B' = \text{MERGESORT}(A') \quad \text{Cost is } T_{n/2} B'' = \text{MERGESORT}(A'') \quad \text{Cost is } T_{n/2} return(\text{merge of } B' \text{ and }$$
$$T_n \leq T_{n/2} + T_{n/2} + n - 1 \approx 2T_{n/2} + n - 1$$

Note 2.4. Earlier we saw that $xT_{n/y}$ implies $T_n = \Theta(n^{\log_y 5})$ but this is not always true.

§2.1 Convex Hull

Can think of it as tightening a string around a set of points in $\mathbb{R}^2$. From any point in convex hull you can drop line where are points are on one side.

Convincing you that finding convex hull is the same as merge sort.

First find left and right hull. Then find upper and lower hull.

§3 Lecture 01-14

§3.1 Recurrences

Mergesort: $T_n = 2T_{n/2} + n$. Roughly because emitting the floor function.
Binary Search: $T_n = 2T_{n/2} + 1$. Roughly because emitting the floor function.
Chip Testing: $T_n \leq T_{n/2} + \frac{3n}{2}$.
Karatsuba Multiplication: $T_n = 3T_{n/2} + n$.
Matrix Multiplication Strassen: $T_n = 7T_{n/2} + n^2$
Halfspace Counting: $T_n = 3T_{n/4} + 1$

§3.2 Solving Recurrences

Methods include:

1. Exact Methods
   a) Substitution. Summing a series.
   b) Induction. Assume $T_n = f(n)$.
2. Order of magnitude
   a) Master theorem: Gives $O(), \Theta(), \Sigma()$ result. Order of magnitude.
   b) Recursion tree: to get insight.

Example 3.1 (Recursion Tree)

$$T_n = 3T_{\frac{n}{4}} + n$$
$$T_1 = 1 \text{ or } 0, T_0 = 0$$
$$k \text{ (height) } = \log_4(n)$$
$$\frac{n}{4^k} = 1$$
$$n, \frac{3}{4}n, (\frac{3}{4})^2 n, (\frac{3}{4})^3 n, \ldots Sum = n(1 + \frac{3}{4} + (\frac{3}{4})^2 + \cdots + (\frac{3}{4})^k)$$
$$\approx= n(1 + \frac{3}{4} + (\frac{3}{4})^2 + \cdots) = \frac{1}{1 - \frac{3}{4}} = 4n \leq 4n \leq 4n + o(n)$$
$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$
$$\Rightarrow T_n = \Theta(n)$$

Most of the time is spent at the top level in this problem

Changing the problem

$$T_n = 4T_{\frac{n}{4}} + n$$
$$n, n, n, n, n, \ldots$$
$$\Rightarrow T_n = \Theta(n \log n)$$

The same amount of time is spent at each level in this problem

Changing one more time

$$T_n = 5T_{\frac{n}{4}} + n$$
$$n, (\frac{5}{4})n, (\frac{5}{4})^2 n, \ldots$$
$$Sum = n(1 + \frac{5}{4} + (\frac{5}{4})^2 + \cdots + (\frac{5}{4})^k)$$
$$Sum = n(\frac{5}{4})^k(1 + \frac{4}{5} + (\frac{4}{5})^2 + \cdots + (\frac{4}{5})^k) \approx \frac{1}{(1 - \frac{4}{5})} = 5$$
$$\approx n(\frac{5}{4})^k * 5 = 5 * 5^k = 5 * 5^{\log_4(n)} = 5n^{\log_4(5)}$$

More time is spent at the bottom level in this problem

Note that the coefficient in front of n wouldn't affect the magnitude.

§3.3 Master Theorem

$$T_n = aT_{n/b} + f(n) \quad (T_n = constant, n \leq \square)$$

Largest of $f(n), n^{\log_b(a)}$

1. $$\frac{n^{\log_b(a)}}{f(n)} \geq n^{\epsilon} \text{ for some } \epsilon > 0 : T_n = \Theta(n^{\log_b(a)})$$

2. $$\frac{f(n)}{n^{\log_b(a)}} \geq n^{\epsilon} .... \quad T_n = \Theta(f(n))$$

3. $$f(n) = \Theta(n^{\log_b(a)}) : T_n = \Theta(f(n) \times \log(n))$$

Technical Condition

$$\limsup_{n \to \infty} \frac{a f(\frac{n}{b})}{f(n)} < 1$$

If $T_n \leq aT_{n/b} + f(n) \quad (T_n = constant, n \leq \square)$, then $O()$ instead of $\Theta()$. If $\geq$, $\Sigma()$. Other cases (exercises to think about):

$$T_n = T_{n/2} + \log n$$
$$T_n = T_{n/2} + T_{n/3} + n$$

§3.4 Strassen's Matrix Multiplication

For any n, there exists a power of $2 \leq 2n$. So assume that n is a power of 2 without loss of generality. Strassen gets rid of a multiplication.

$$T_n = 8T_{\frac{n}{2}} + n^2 = \Theta(n^3)$$
$$\text{Became}$$
$$T_n = 7T_{\frac{n}{2}} + n^2 = \Theta(n^{2.8})$$

Best known to date: $\Theta(n^{2.374})$. Goal: $\Theta(n^2 \log n)$.

§3.5 Halfspace Counting

Given: $x_1, \ldots, x_n \in \mathbb{R}^2$. Users query: How many points on one side of H (hyperplane).
Fact: There exists a partition of $x_1, \ldots, x_n$ into 4 equal sets, using only 2 lines.

Theorem 3.2 (Pancake Theorem)
Take a shape. There is always a cut that cuts the pancake into two equal sets.

Also works with overlapping pancakes. Can cut both in have with a single line.

§4 Lecture 01-16

§4.1 Selection Problem

Given: $x_1, \dots, x_n \in \mathbb{R}$ (pairwise distinct)

Want: Find the kth smallest.

Sorting leads to $\Theta_(n \log n)$ complexity. But we can do better.

Algorithm of Blum et al.

If $n \leq 5$, sort and exit. Cost ≤ 7. Try it at home. Usually it can be done in 8, but with an extra trick you can do it in 7.

Else

1. Divide all elements into groups of 5 and find the median. Cost $\leq n/5 \times x$ where x is the time to find the median of a group of 5 elements. It isn't obvious but it can be done in 6.
2. Find the median of M, recursively. Cost $\leq T_{n/5}$
3. Compare all items with m, forming sets L, R. Cost $= n - 1$.
4. Case $k \leq |L|$, then find the kth smallest in L. Cost is $T_{|L|} \leq T_{7n/10}$
   Case $k = |L| + 1$, then return m. This has cost of 0.
   Case $k > |L| + 1$, find the $k - |L| - 1$ smallest in R. Cost is $T_{|R|} \leq T_{7n/10}$

$$T_N \leq \begin{cases} 7, & n \leq 5 \\ T_{n/5} + T_{7n/10} + \frac{11}{5}n & n > 5 \end{cases}$$

Now to show inductively that $T_n \leq Cn$.

Proof. Case where $n \leq 5$, then $7 \leq C$, so $C \geq 7$.

Now assume that $T_n \leq Cn$ up to $n - 1$, where $n > 5$.

$$T_{n/5} + T_{7n/10} + \frac{11}{5}n \leq C(n/5 + 7n/10) + \frac{11}{5}n = C\frac{9}{10}n + \frac{11}{5}n \leq C_n$$
$$\Rightarrow \frac{11}{5} \leq \frac{C}{10} \Rightarrow C \geq 22$$

An improvement can be made by return L and R, so step 3 only has to check $\frac{4}{10}n - 1$ elements.

$$C\frac{9}{10}n + \frac{8}{5}n \leq C_n$$
$$\Rightarrow \frac{8}{5} \leq \frac{C}{10} \Rightarrow C \geq 16$$

□

There we go that is the linear time algorithm.

Tips for guessing the order of an algorithm when preparing to do induction: The subscripts on T summed together are less than 1. So when building the recursion you'll see that the cost at each level decreases by $\frac{1}{10}$. So it's reasonable to assume that it might be linear.

Note 4.1. T_n = maximal cost on any input of size $\leq n$. This implies monotone.

§4.2 Algorithm "FIND" (Hoare)

Very similar to the above algorithm.

Worst case of quick sort. $n + (n-1) + (n-2) + \cdots + (n - (n-1)) = \Theta(n^2)$

$E\{T_n\} \leq \frac{1}{2}(ET_n + E_T 3n/4) + n$.
So $ET_n \leq E_{8n/4} + 2n \leq Cn \Rightarrow \frac{3n}{4} \Rightarrow$

§4.3 Mergesort Induction

$$T_n \leq T_{\lfloor (n/2) \rfloor} + T_{\lceil (n/2) \rceil} + n - 1$$

Claim: $T_n \leq Cn \log_2(n)$

Proof.

$$\begin{cases} T_n \leq T_{\lfloor (n/2) \rfloor} + T_{\lceil (n/2) \rceil} + n - 1 \\ T_0 = 0, T_1 = 0 \end{cases}$$

If $n = 1$, ✓.

If n is even. $T_n \leq 2T_{n/2} + n - 1 \leq 2C(n/2)\log_2(\frac{n}{2}) + n - 1$

$$= cn\log_2 n - cn + n - 1 < n \log_2 n \quad \forall C \geq 1$$

If n is odd.

$$\begin{aligned} T_n &\leq T_{\frac{n+1}{2}} + T_{\frac{n-1}{2}} + n - 1 \\ &\leq C\frac{n+1}{2}\log_2\frac{n+1}{2} + C\frac{n-1}{2}\log_2\frac{n-1}{2} + n - 1 \\ &= C\frac{n}{2}\log_2\frac{n^2-1}{4} + \frac{C}{2}\log_2\frac{n+1}{n-1} + n - 1 \\ &= Cn\log_2 n - Cn + \frac{C}{2} + n - 1 \\ &< n\log_2 n \end{aligned}$$

□

§4.4 Binary Search

Given: Sorted array $x_1, \ldots, x_n$.

Find: x. Return either. 1. $x = x_i$ or 2. $x_i < x < x_{i+1}$

Model: Ternary Oracle. Two inputs, three possible outputs (<, =, >).
Binary Oracle: $x \leq y$ or $x > y$.

BinarySearch(x, i, j)

Cases

1. $i = j$. Ternary Oracle(x, x_i). Exit with either $x = x_i$ or x is not present.
2. $i > j$. This doesn't especially make sense. Exit with "x not present"
3. $i < j$. Let $k = \left\lfloor (\frac{i+j}{2}) \right\rfloor$. Ternary oracle (x, x_k).

$$\begin{cases} \text{Return BinSearch } (x, i, k-1) & x < x_k \\ \text{Return "}x = x_k\text{"} & x = x_k \\ \text{Return BinSearch } (x, k+1, j) & x > x_k \end{cases}$$

$$T_n \leq \begin{cases} T_{n/2} + 1 & \text{n even} \\ T_{(n-1)/2} + 1 & \text{n odd} \end{cases}$$
$$T_1 = 1$$
$$T_0 = 0$$

§5 Augmented Data Structures

1. Red-black tree + Linked List
2. Order statistics.
3. Interval Trees
4. Fast browsing
5. K-d trees.

§5.1 Order Statistics ADT

Operations:

1. Dictionary:
    a) Insert
    b) Delete
    c) Search
2. Order:

a) Select(k, D) Give you rank and ask for the element.

b) Rank(x, D) returns the rank of item ”x”. The smallest has rank 1. The largest has rank n.

§5.2 Order Statistics Tree

A red black tree with cells with following attributes: Color, rank, key, left, right. And the addition of size.

§5.3 Interval Trees

Operations:

1. Insert, Delete.

2. Overlap. Returns true if there exists overlap between two intervals.

§5.4 Sweepline

A strategy that makes a D dimensional problem into a $D-1$ dimensional problem by converting one of the dimensions to time.

Data structure for y-intervals. Birth: insert a new node. Death: Delete an interval.

§5.5 Fast Browsing

Insert, Delete, Search, k-Next browset
Implementation: Level-linked red-black trees.