Administrivia

Instructor: Lawrence Teitelman

Email: Lawrence.Teitelman@qc.cuny.edu LT.CS320@yahoo.com

Class Time: Wednesdays, 6:30 PM to 9:20 PM

Class location: SB D133

Course website: https://groups.yahoo.com/group/qc-algorithms/ (join this group)

Textbook: https://people.cs.vt.edu/~shaffer/Book/ (free)

This lecture note: https://docs.google.com/document/d/1qgrXDDrj8L70rmftjGJ_kpq5aOJLrfnuJ7U327NOZDs/

Requirements

Lecture notes: Do not replace class but it can be a good transcript of the class. Prefer typed over handwritten. Prefer PDF over Microsoft Word.

Homework: We will usually go over homework together in class. As such, we should try to do the homework before the class. Questions on the exam may(?) follow the pattern from homework questions.

Projects: Comparing different algorithms to solve problems. A lot of math to see if it matches with what we see empirically. (Empirical: based on, concerned with, or verifiable by observation or experience rather than theory or pure logic[1]) A project is an opportunity to see what happens in reality.

Exams: Fifth week, eleventh week, finals week. Older exams on Yahoo! group

Office hours: After class or get in touch over email.

Suggestion: Print the table of contents from the textbook and check off or highlight topics we have covered as we go.

[1] https://www.google.com/search?q=define+empirical

Grades: Class participation 10% Lecture notes 05% Homework 15% Project 15% Exam 1 15% Exam 2 15% Final Exam 25%

No	Date	Topic	Text chapter
1	February 3	introduction to algorithms analysis; mathematical foundations	2, 3
2	10	basic data structures	4
3	17	sorting algorithms	7
4	24	sorting algorithms	7
5	March 2	binary and non-binary trees, **exam 1**	5, 6
6	9	searching	9
7	16	advanced tree structures and indexing	10, 13
8	30	graph algorithms	11
9	April 6	graph algorithms	11
10	13	analysis techniques	14
11	20	lower bounds; **exam 2**	15
12	May 4	advanced design techniques	16
13	11	limits to computation, np completeness	17
14	18	selected topics and general review	TBD
15	25	**final examination**	n/a

Algorithms

What is the definition of algorithm?[2] An algorithm is defined. An algorithm has to follow a sequence. It is a step by step solution of a problem. It should be a general solution. Etymology: the word algorithm comes from the name of a person.

We will learn how to solve problems such as sorting, searching, geometry matching and so on. We will learn a lot of different algorithms.

Design and analysis of algorithms

What is analysis? Analysis tries to answer questions such as the following:

1. Is the algorithm correct?
2. Does it solve the problem?
3. If it solves the problem, how well does it do the job?

Talking about efficiency, we can talk about two things:

1. Time complexity
2. Space complexity

Time complexity: We measure time complexity by answering the question “how many iterations does it take?” and NOT how long it takes on a clock because the latter is implementation specific.

Space complexity: We usually talk about internal memory. Sometimes, we can talk about external memory. It is important to distinguish between space required for data as opposed to auxiliary storage. Auxiliary storage is how much extra space it takes for our algorithm. For example, how much space does my tree take if I decide to use a tree? The storage space used by the tree is a part of the auxiliary storage.

Do space and time go hand in hand or are they independent? Sometimes time complexity and space complexity work hand in hand while sometimes they work opposite. In graph algorithms, adjacency matrix takes n^2 space and adjacency list takes less space and also less time.

What is design?[3] We can classify our algorithms based on their design into several categories. For example,

1. Brute force
2. Divide and conquer

[2] The words 'algorithm' and 'algorism' come from the name al-Khwārizmī. Al-Khwārizmī (Persian: خوارزمي, c. 780-850) was a Persian mathematician, astronomer, geographer, and scholar. https://en.wikipedia.org/wiki/Algorithm

[3] We are not doing chapter 1 in class. Read about design patterns section. Read the design patterns book if you like. This is not the design we talk about in this class. Perhaps it is this gang of four book? http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

3. Backtracking / decrease and conquer
4. Greedy algorithm
5. Dynamic programming
6. Randomized algorithms

(not an exclusive list)
If you look at the list above, these are not algorithms that solve sorting. These are designs of algorithms. Some books focus on algorithms. They will look at a problem and list many algorithms to tackle the problem. Other books focus on analysis and how quickly an algorithm works.

Cormen and three others wrote a book called Algorithms[4]. Levitan's book[5] is organized differently. It is organized by types such as divide and conquer, greedy, … and so on. Sedgewick[6] has two sets of books. One is on algorithms[7] while the other is on analysis[8]. The latter is perhaps too mathematical for our purposes.

There are many ways to teach the course. We take a hybrid route.

Critical to this class is the data structures concept. We will learn advanced data structures. We will not only write programs but also associated data structures.

7:15 PM We will illustrate different ways of solving a problem. E.g., Registrar sees you are registered for this class but not paid. They might want to send mail or catch you and so on. How would a security guard go about this?

Brute force: Search for the person in the whole campus. This is the brute force way.

How can we measure complexity in this case? Visiting every student or getting to the place is probably expensive, walking or driving and it is not always clear which is the dominant operation. If we are writing a graph, it is usually not the comparison that is dominant.

Let's say we have n students that didn't pay the bill. We can find one student and do the same operation n times for n students. This would be the brute force way.

We are going to each student if it is about walking to a room then we can make a list.

Difference between one student vs n students: Maybe there is no difference in a brute force algorithm because we just do n * x for n students.

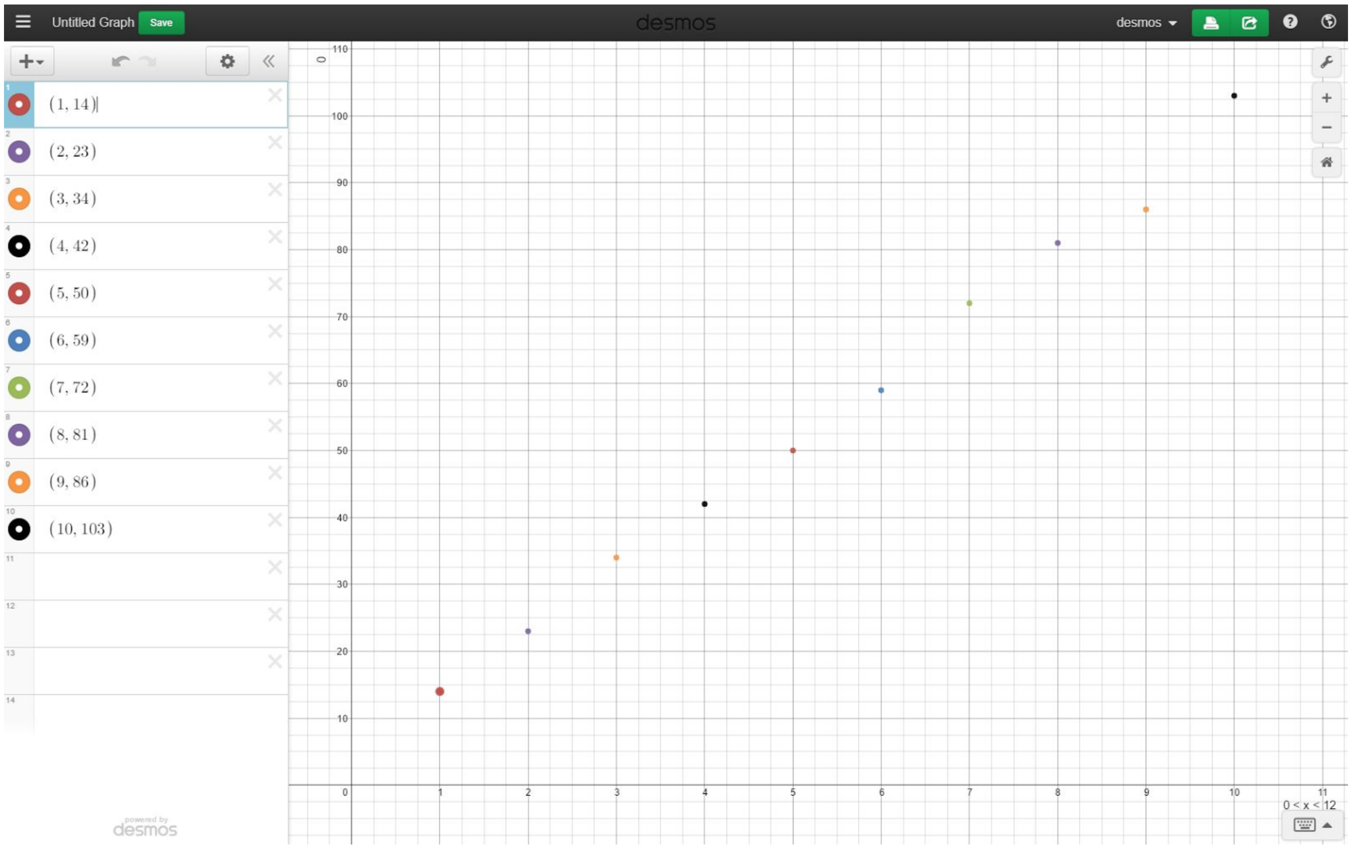
[4] http://www.mif.vu.lt/~valdas/ALGORITMAI/LITERATURA/Cormen/Cormen.pdf
http://bayanbox.ir/view/4177858657730907268/introduction-to-algorithms-3rd-edition.pdf It is a big book so it is in the reference section.
[5] http://www.vgloop.com/_files/1394454921-126688.pdf
[6] Robert Sedgewick is a professor at Princeton University and a member of the board of directors at Adobe Systems.
[7] https://drive.google.com/file/d/0B2uJazaRVsV1RWxkNDJ5RWZPSFE/view
[8] https://drive.google.com/file/d/0B2uJazaRVsV1bTlJaFFaaVh1c3M/view

Mathematical preliminaries

Sequences: A sequence of elements from $a_0, a_1, a_2, \ldots, a_n \rightarrow a$ If we are given a finite sequence of numbers, we can make it fit in more than one mathematical model. One example of a sequence is a Fibonacci sequence which is defined as follows:

1. f(0) = 0 , f(1) = 1
2. f(n) = f(n - 1) + f(n - 2)

Contrary to popular belief, not all sequences have a particular reason. For example, 14, 23, 34, 42, 50, 59, 72, 81, 86, 103 is the sequence of stops on the A train[9]. However, we could come up with more than one mathematical model that fits this sequence.



(1, 14)
(2, 23)
(3, 34)
(4, 42)
(5, 50)
(6, 59)
(7, 72)
(8, 81)
(9, 86)
(10, 103)

The point is that there can be more than one “correct” answer to what the next term in the sequence should be.

9 https://en.wikipedia.org/wiki/A_(New_York_City_Subway_service)

Summations: Summations are important in algorithms because algorithms are usually an iterative process and often work in loops. An algorithm may take a certain number of operations per iteration. We would need to add the operations in all the iterations.

For example, what is the summation of 1 + 2 + 3 + … + n? The story goes that Gauss was a kid when his teacher asked him to add all integers from one to a hundred or a thousand to keep him occupied. Gauss figured out that you can list the numbers in descending order to get the same sum (because addition is commutative. If we put them as follows, we will see something interesting when we add the first row and second row in the same column.

1	2	3	4	5	6	7	...	99	100
100	99	98	97	96	95	94	...	2	1
101	101	101	101	101	101	101	101	101	101

1	2	3	4	5	6	7	...	n - 1	n
n	n - 1	n - 2	n - 3	n - 4	n - 5	n - 6	...	2	1
n + 1	n + 1	n + 1	n + 1	n + 1	n + 1	n + 1	n + 1	n + 1	n + 1

We will see that (1 + 2 + … + n) + (n + … + 2 + 1) = n * (n + 1)

$$\sum_{k=1}^{n} k = \frac{1}{2}(n)(n+1)$$

There are various ways to prove this is the case including geometric proof, proof by mathematical induction/weak induction, …

For proof by mathematical induction, we need a base case and an inductive hypothesis. Think of an infinite ladder. How do we climb an infinite ladder? We start on the first rung. Then, we find a general way to get on the next rung, given we are on a given rung. Usually (though not always) the base case is zero and one. Our proposition $\mathcal{P}$(n) is to prove that if $\mathcal{P}$(k) is true then $\mathcal{P}$(k + 1) is true as well.

In case of summation, we have our base case covered pretty easily. We need to show

$$1 = \frac{1}{2}(1)(1+1)\ (?) \Rightarrow 1 = 1$$

which is true. Thus the base case holds for $n = 1$

Now, if $\mathcal{P}$(k) holds true, we will show that $\mathcal{P}$(k + 1) holds true as well. When we say $\mathcal{P}$(k) holds true, we mean

$$1 + 2 + ... + k = \frac{(k)(k+1)}{2}$$

Let us add $(k + 1)$ to both sides. We now have,

$$1 + 2 + ... + k + k + 1 = \frac{(k)(k+1)}{2} + (k+1)$$

$$\text{or, } 1 + 2 + ... + k + k + 1 = \frac{(k+1)(k+2)}{2}$$

Thus, the proposition holds true for $k + 1$

We might talk about proof by strong induction later in the semester.

7:47 PM break

A lower bound and an upper bound[10] of the summation of all positive integers from one to a given positive integer n. Since n is a part of this summation, the least that the sum can be is n. This is our lower bound. Moreover, since everything in the series is either equal to or less than n, the greatest the sum could possibly be is n^2. In other words,

$$1 + 2 + 3 + ... + n \geq n$$

$$1 + 2 + 3 + ... + n \leq n^2$$

We know that 1 + 2 + … + n is not to exceed degree of two.

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... \text{ is } an^2 + bn + c$$

When n = 0, we have $c = 0$

When n = 1, we have $a + b + c = 1$

when n = 2, we have $4a + 2b + c = 3$

Solving this system of simultaneous equations, we get $a = \frac{1}{2}$ and $b = \frac{1}{2}$

Using these values in $an^2 + bn + c$, we get $\frac{1}{2}(n)(n + 1)$.

This is a more algorithmic method of arriving at the answer.

Homework problem: Try to find a formula for $1^2 + 2^2 + 3^2 + ... + n^2$ Use the polynomial based approach to find the formula. Prove it works with induction.

Another proof technique is proof by contradiction. We assume to the contrary that the hypothesis is incorrect. If it leads to a contradiction then the original hypothesis is true. We define our set of natural numbers as $\{0, 1, 2, ...\}$ Integers as $\{... -3, -2, -1, 0, 1, 2, 3, ...\}$, rational numbers as $\{\frac{a}{b} \text{ where } a \varepsilon \mathbb{Z}, b \varepsilon \mathbb{Z} - \{0\}\}$. We prove that $\sqrt{2}$ is irrational using proof by contradiction.

[10] A self-help bookcalled how to stop worrying and start living talks about thinking about the worst case scenario. Usually, the worst case scenario isn't that bad in the bigger scheme of things. The book says Ask yourself, "What is the worst that can possibly happen?"

Exponent and logarithm

$$2^3 = 8 \Leftrightarrow log_2 8 = 3$$

We are usually interested in log_2 in computer science as opposed to log_e or log_{10}. In a calculator, we can do

$$log_2 x = \frac{log_{10} x}{log_{10} 2}$$

Asymptotic notations

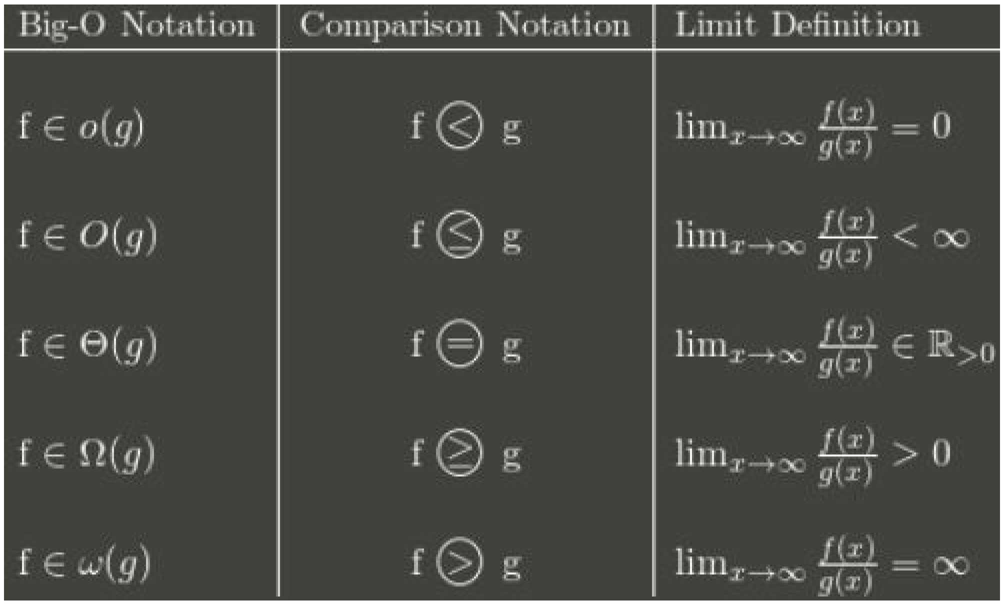
1. Big O
2. Little O
3. Big Omega Ω
4. Little omega ω
5. Theta Θ

Why do we not care about negative numbers in this context?

We cannot do a problem is less than zero operations and we cannot store values in less than zero space in memory.

Why is there no big and little theta Θ when there are big O and little O and there are big omega and little omega?

There are no big and little theta because theta means they are the same asymptotic behavior. big means they are bounded above.



Big-O Notation	Comparison Notation	Limit Definition
$f \in o(g)$	$f \textcircled{$<$} g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$
$f \in O(g)$	$f \textcircled{$\leq$} g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} < \infty$
$f \in \Theta(g)$	$f \textcircled{$=$} g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} \in \mathbb{R}_{>0}$
$f \in \Omega(g)$	$f \textcircled{$\geq$} g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} > 0$
$f \in \omega(g)$	$f \textcircled{$>$} g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty$

[11]

Here the limit is a limit superior.

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ We want to know the relationship between $f(n)$ and $g(n)$.

One way to do this is with L'Hôpital's rule.

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ has three possibilities in our context.

1. ∞
2. 0
3. $c\ \varepsilon \Re^+$

for 1. $f(n) = \omega(g(n))$ and $f(n) = \Omega(g(n))$

[11] https://stackoverflow.com/a/1364582/227646

for 2. $f(n) = o(g(n))$ and $f(n) = O(g(n))$

for 3. $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ and $f(n) = \Theta(g(n))$

For example, let $f(n) = 3n^2 + 5n + 7$ and $g(n) = n^3 + 100n + 250$

With L'Hôpital's rule, we have $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{6n+5}{3n^2+100}$

Applying again, we have $\frac{6}{6n}$

Thus, $f(n)$ is $o(g(n))$. $f(n)$ is also $O(g(n))$. However, we prefer to say f(n) is little o of g of n because little o is stronger than big O.

One of the things we do is rank functions. Let's take an extreme case $f(n) = 1000n^2$ and $g(n) = 2n^3$ Which is bigger? The answer depends on the answer to the question for which n?

For our purposes, the coefficient of the function is not important. If we consider $f(n) = 1000n^2$ and $g(n) = 2n^2$ we can say that $f(n) = \Theta(g(n))$. In other words, we put f and g in the same complexity class.

It is important to note that being in the same complexity class does not mean that we are talking about exact values or < or >. They are Θ of each other. They are not equal but rather we group them together.

8:50 PM Consider $n!$ and 2^n. Sometimes, it is useful to compare the log of two functions. Sometimes, it is useful to compare the exponents of two functions.

$n! = (n)(n-1)(n-2)\ldots$ $\qquad\qquad\qquad\qquad$ 2^n

We know, $log\, a^b = b\, log\,(a)$ and $log\,(a\, b) = log\, a + log\, b$

$log_2(n!) = log_2 n + log_2(n-1) + \ldots + log_2 1$

$log_2(n!) > n$

$log_2(n!) < n\, log_2 n$

This is a technique you can use so we can use e and log.

Recap: Take a look at chapter 1 for background information. We covered all of chapter 2 but recurrences. We touched most of chapter 3 except definition of asymptotic groups.