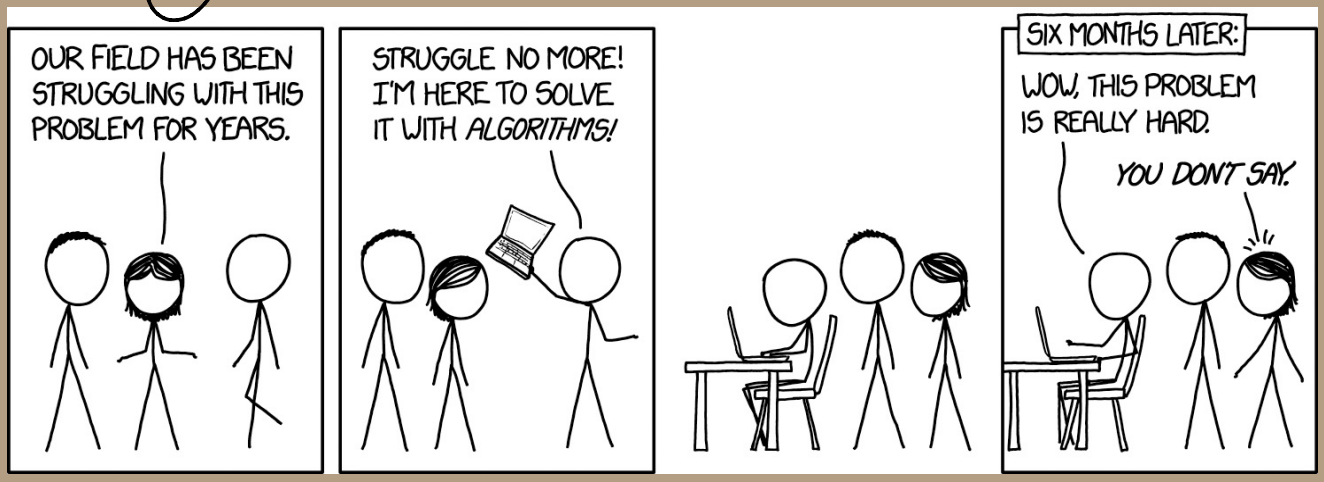
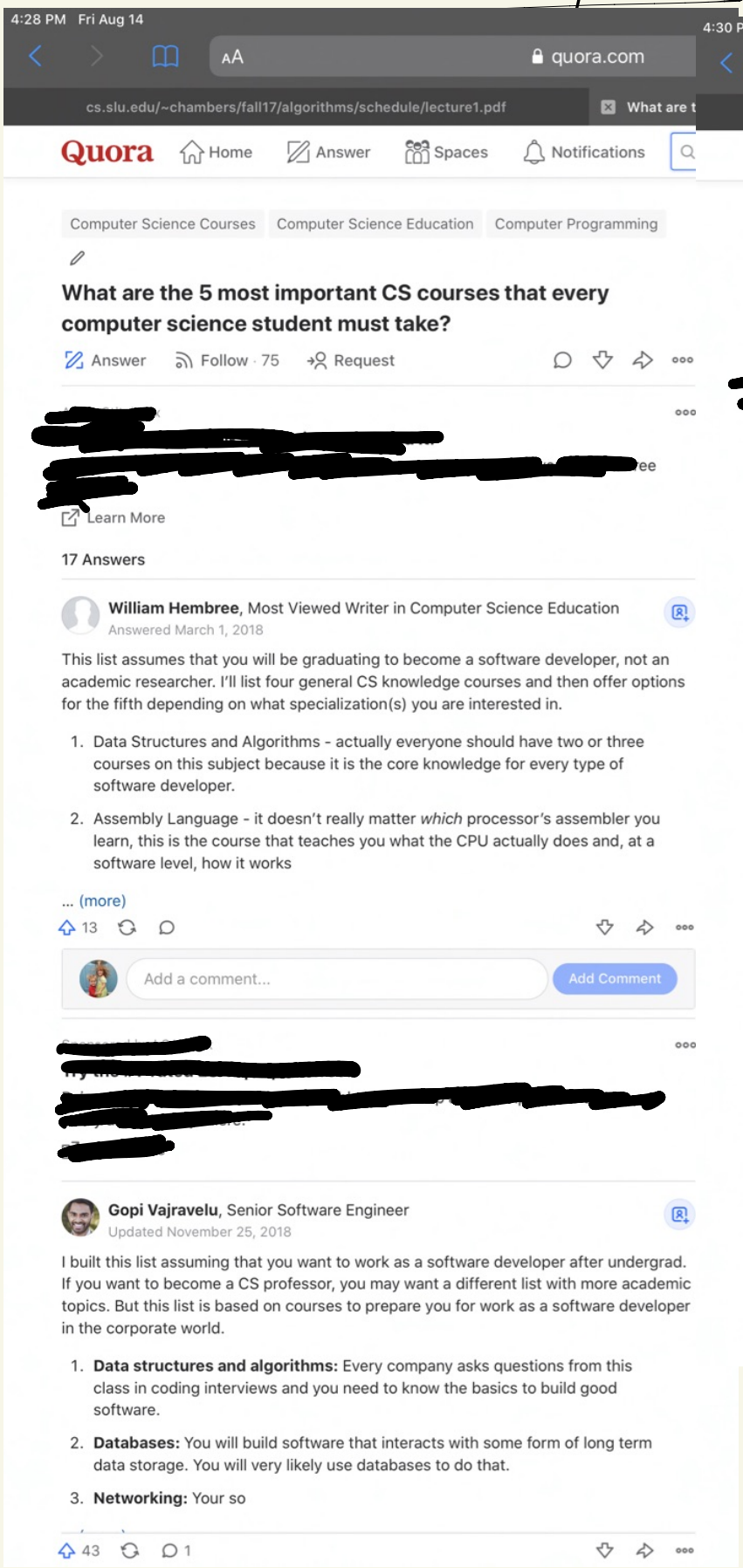
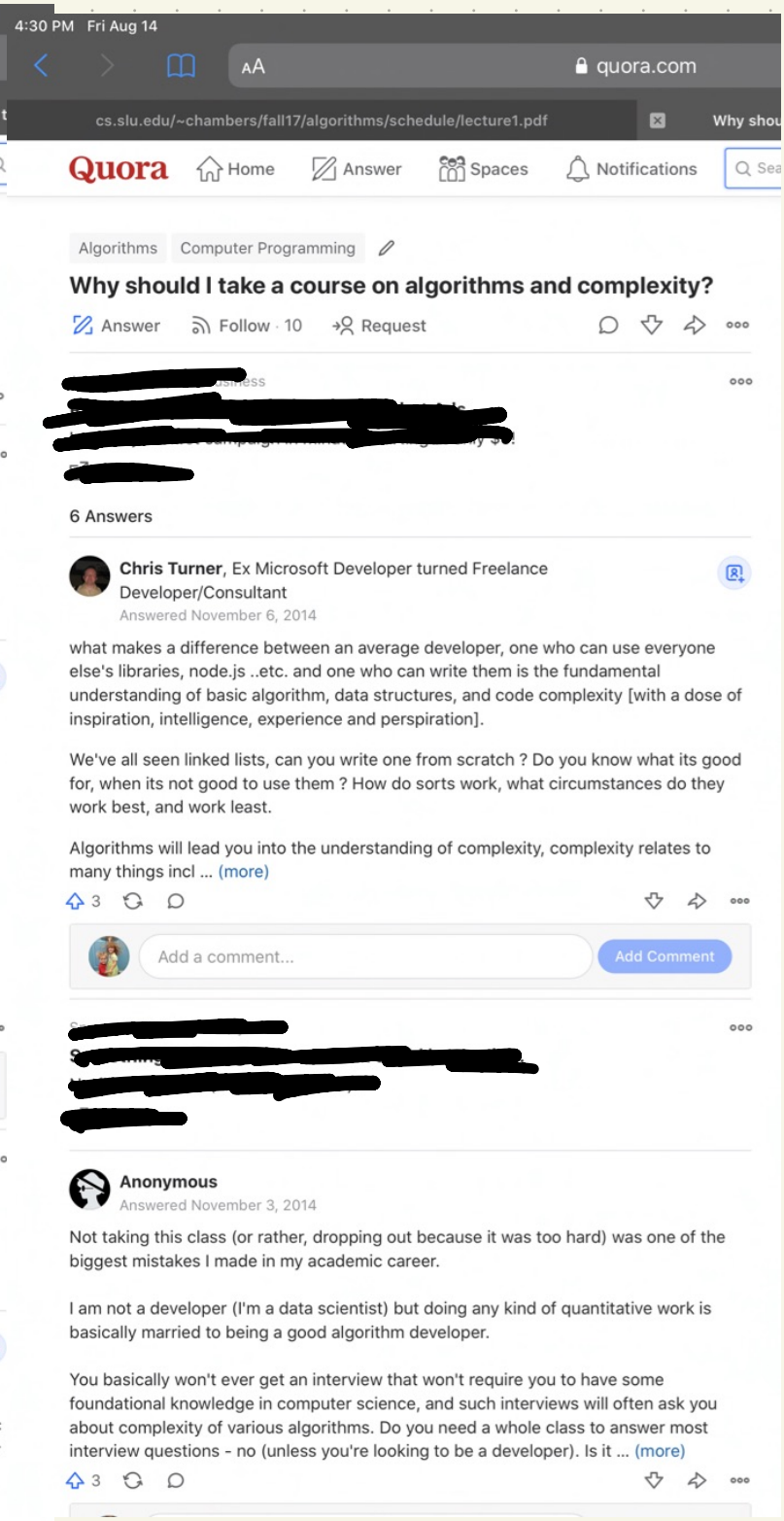
Algorithms – Spring '25



Introduction + Recap

First & Why?


4:28 PM Fri Aug 14
AA
quora.com
cs.slu.edu/~chambers/fall17/algorithms/schedule/lecture1.pdf
What are t
Quora
Home
Answer
Spaces
Notifications
Computer Science Courses
Computer Science Education
Computer Programming
What are the 5 most important CS courses that every computer science student must take?
Answer
Follow · 75
Request
Learn More
17 Answers
William Hembree, Most Viewed Writer in Computer Science Education
Answered March 1, 2018
This list assumes that you will be graduating to become a software developer, not an academic researcher. I'll list four general CS knowledge courses and then offer options for the fifth depending on what specialization(s) you are interested in.
1. Data Structures and Algorithms - actually everyone should have two or three courses on this subject because it is the core knowledge for every type of software developer.
2. Assembly Language - it doesn't really matter which processor's assembler you learn, this is the course that teaches you what the CPU actually does and, at a software level, how it works
... (more)
13
Add a comment...
Add Comment
Gopi Vajravelu, Senior Software Engineer
Updated November 25, 2018
I built this list assuming that you want to work as a software developer after undergrad. If you want to become a CS professor, you may want a different list with more academic topics. But this list is based on courses to prepare you for work as a software developer in the corporate world.
1. Data structures and algorithms: Every company asks questions from this class in coding interviews and you need to know the basics to build good software.
2. Databases: You will build software that interacts with some form of long term data storage. You will very likely use databases to do that.
3. Networking: Your so
43
1


4:30 PM Fri Aug 14
AA
quora.com
cs.slu.edu/~chambers/fall17/algorithms/schedule/lecture1.pdf
Why shou
Quora
Home
Answer
Spaces
Notifications
Algorithms
Computer Programming
Why should I take a course on algorithms and complexity?
Answer
Follow · 10
Request
6 Answers
Chris Turner, Ex Microsoft Developer turned Freelance Developer/Consultant
Answered November 6, 2014
what makes a difference between an average developer, one who can use everyone else's libraries, node.js ..etc. and one who can write them is the fundamental understanding of basic algorithm, data structures, and code complexity [with a dose of inspiration, intelligence, experience and perspiration].
We've all seen linked lists, can you write one from scratch ? Do you know what its good for, when its not good to use them ? How do sorts work, what circumstances do they work best, and work least.
Algorithms will lead you into the understanding of complexity, complexity relates to many things incl ... (more)
3
Add a comment...
Add Comment
Anonymous
Answered November 3, 2014
Not taking this class (or rather, dropping out because it was too hard) was one of the biggest mistakes I made in my academic career.
I am not a developer (I'm a data scientist) but doing any kind of quantitative work is basically married to being a good algorithm developer.
You basically won't ever get an interview that won't require you to have some foundational knowledge in computer science, and such interviews will often ask you about complexity of various algorithms. Do you need a whole class to answer most interview questions - no (unless you're looking to be a developer). Is it ... (more)
3

Mindset (& main goals):

- Interview: how to concisely explain your answer to a technical question, complete with trade-offs
- Design meeting: same!
- Developer: What known approaches / issues are there with regards to your problem / code?
- Modeling: If already developed, which solution fits your domain area best.

(Note: all of these are relevant for data structures, too!)

Q: What IS an algorithm?

algorithm

noun

Word used by programmers when they do not want to explain what they did.

Related: What is a program?
(+how is it different?)

Goal: Give high level idea of your solution.

- Not code!

  Why?

- Not English prose!

  Use data structures, loops, functions, etc.

In here, 3 parts to every algorithm:

1.

2.

3.

Preregs:

① Discrete math:
Why?

② Data structures:
Why?

Goal of HW0:

- Re-locate references to these 2 classes.
- Re-familiarize your self with core concepts.
- Practice pseudo code. (It will be hard!)
- Meet some fellow students & form working relationships
- Figure out Canvas, while tackling known topics! (Hopefully, you'll already have seen everything on HW0...)

Course logistics:

Canvas site for everything.

Outline:

- Canvas: HW submission, lecture notes
- Perusall: every Sunday + Thurs.
- Homework: Oral + written

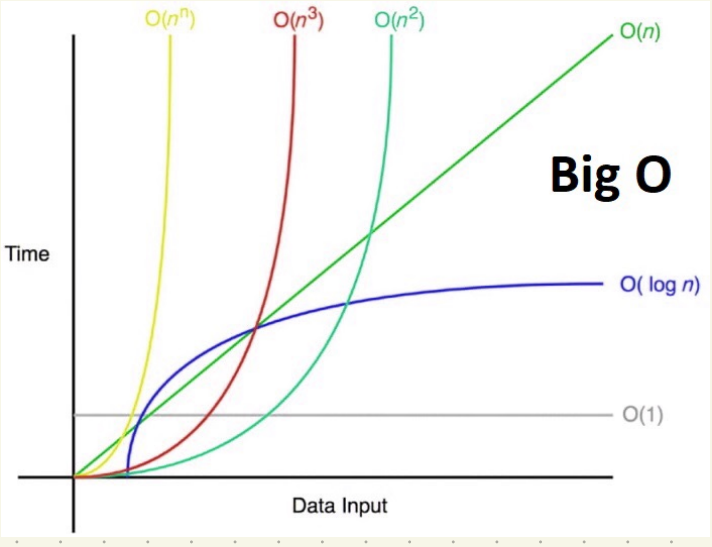
Policies to note:

- Integrity
- Group work
- Missing work policies

Some background reminders

① Big-O: ignore constants!
Also, smaller terms "disappear":

- $5 \cdot 2^n + 6n^3 + 3$
- $\frac{5}{3}n^2 - 33n + 1$
- $1{,}634n^2 + 10n + 1{,}000{,}000$



More formally: $f(n)$ is $O(g(n))$ if

Prove: $5n^2 + 11n - 6$ is $O(n^2)$:

(2) logarithms: useful identities!

Find it in your discrete math reference, ie →

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$

$$\log_b(x^y) = y \log_b(x)$$

$$\log_b(\sqrt[y]{x}) = \frac{\log_b(x)}{y}$$

Why? logarithms are all about exponents!

what is $2^a \cdot 2^b$?

Another:

$$\log_a b = \frac{\log_x b}{\log_x a}$$ with any base x

Question: Is $\log_{10} n$ big-O of $\log_2 n$?

$\log_{10} n =$

Ex: what about $\log_a(a^x)$?

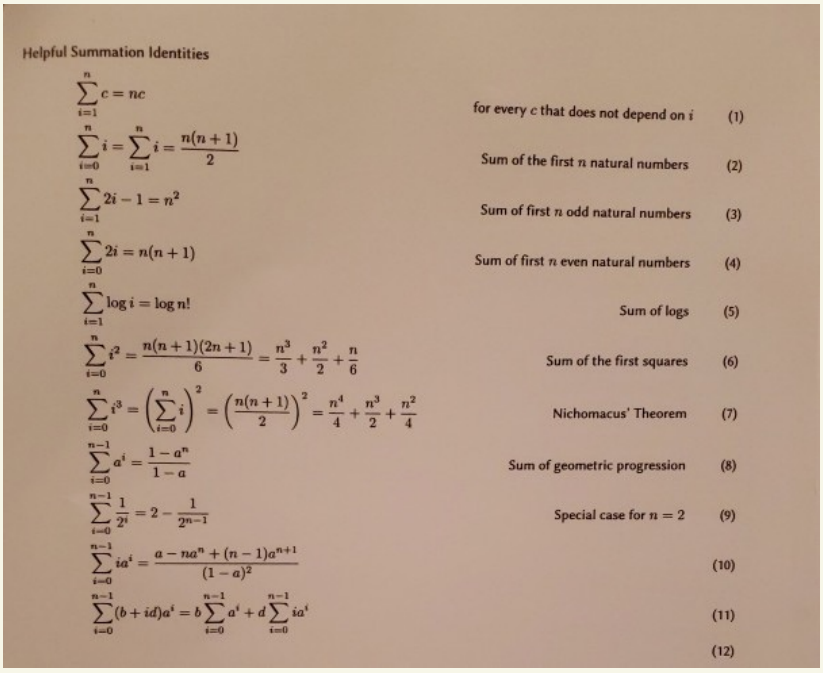
or: $\log_2(n^2)$

③ Summations:

again, your discrete math book has a table.

Find it.

Ex:



Helpful Summation Identities

$$\sum_{i=1}^{n} c = nc \qquad \text{for every } c \text{ that does not depend on } i \qquad (1)$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \text{Sum of the first } n \text{ natural numbers} \qquad (2)$$

$$\sum_{i=1}^{n} 2i - 1 = n^2 \qquad \text{Sum of first } n \text{ odd natural numbers} \qquad (3)$$

$$\sum_{i=0}^{n} 2i = n(n+1) \qquad \text{Sum of first } n \text{ even natural numbers} \qquad (4)$$

$$\sum_{i=1}^{n} \log i = \log n! \qquad \text{Sum of logs} \qquad (5)$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \qquad \text{Sum of the first squares} \qquad (6)$$

$$\sum_{i=0}^{n} i^3 = \left(\sum_{i=0}^{n} i\right)^2 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} \qquad \text{Nichomacus' Theorem} \qquad (7)$$

$$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a} \qquad \text{Sum of geometric progression} \qquad (8)$$

$$\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}} \qquad \text{Special case for } n = 2 \qquad (9)$$

$$\sum_{i=0}^{n-1} i a^i = \frac{a - na^n + (n-1)a^{n+1}}{(1-a)^2} \qquad (10)$$

$$\sum_{i=0}^{n-1} (b + id)a^i = b\sum_{i=0}^{n-1} a^i + d\sum_{i=0}^{n-1} i a^i \qquad (11)$$

(12)

Notation: $\sum_{i=1}^{n} f(i)$

Ex: $\sum_{i=1}^{n} 1$, or $\sum_{i=1}^{n} i$

(4) Induction: There is a template!

Base case:

Ind hypothesis:

Ind. step:

Think of this as "automating" a proof.

Learn it, use it, love it!

Example:

EXAMPLE 3 Use mathematical induction to show that

$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$$

"Structural" induction:

Let n be a positive integer. Show that every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes, where these pieces cover three squares at a time, as shown in Figure 4.

5 Pseudo code & runtime: Discrete math examples

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** $location := i$
else $location := 0$
return $location$ {$location$ is the subscript of the term that equals x, or is 0 if x is not found}

Pseudocode conventions here:

Variable assignment:

Boolean comparison:

```
if (x = 5)
```

Arrays: A[0..n-1]

- each A[i]

Loops:

```
for i ← 1 to 100
    A[i] ← i
```

Again, takes practice! Read intro chapter + then give HW0/HW1 a try.

Pseudocode format:

In a pinch, pretend you're in Python / Ruby.

High level & readable.

I realize this is not a "definition" - that is the point!

It's about effective communication.

Initially:

- lots of examples
- lots of practice
- reach out if you have questions
- in a pinch - peer evaluation!

Example (& tie to runtimes):
Initializing arrays.

```
FIBONACCIMULTIPLY(X[0..m−1], Y[0..n−1]):
  hold ← 0
  for k ← 0 to n + m − 1
      for all i and j such that i + j = k
          hold ← hold + X[i] · Y[j]
      Z[k] ← hold mod 10
      hold ← ⌊hold/10⌋
  return Z[0..m + n − 1]
```

Space & runtime:

Q: What if all won't get filled?

More complex: recursion!

Algorithm 1 Quicksort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q = PARTITION(A, p, r)
4:         QUICKSORT(A, p, q − 1)
5:         QUICKSORT(A, q + 1, r)
6:     end if
7: end procedure
8: procedure PARTITION(A, p, r)
9:     x = A[r]
10:    i = p − 1
11:    for j = p to r − 1 do
12:        if A[j] < x then
13:            i = i + 1
14:            exchange A[i] with A[j]
15:        end if
16:        exchange A[i] with A[r]
17:    end for
18: end procedure
```

QuickSort Pseudocode Example

Recursive Algorithms : Chapter 1

1st half:

Setup, plus (hopefully) familiar examples:

- Towers of Hanoi
- Merge sort

2nd half:

- Recap of recurrences & "Master theorem"
- Linear time selection
- Multiplication (again)
- Exponentiation