Algorithm Design and Analysis

Introduction & Linear-time Selection Algorithms

Logistics

Who is 15-451 / 15-651?

Daniel Anderson

~~Daniel~~ Danny Sleator

TAs: Abby Li, Claire Jin, David Tang, Efe Cekirge, Jonathan Liu, Nick Grill, Raaid Tanveer, Sophie Liu, Dhruti Kuchibhotla, Summit Wei, Will Gay, Yoseph Mak

Grading and policies

8 Homeworks:	40%
Recitation Attendance	5%
2 Midterm exams	30%
Final exam	25%

See the full policy list: https://www.cs.cmu.edu/~15451-f23/policies.html

Homework

- **Written Homework:** Each has 3-4 problems. Solutions must be typeset, not handwritten!
- **Oral Homework:** Collaborate in groups of three and present your solutions to a TA
- **Programming Problems:** Zero or one of these on each homework. Submitted via Autolab.
  - Officially recommended/supported languages are **C++** and **Python**
  - We will also try to accept C, Java, OCaml, Rust, SML

Homework submission

- Submit to Gradescope/Autolab by 11:59pm on the due date
- **Grace days:** 2 for written, 2 for programming (separate)
- Homework 1 will be released on Thursday

Recitation

- Please review the lecture notes and read the problems beforehand
- Only 50-minutes long so please show up on time!
- 5% of your grade from attendance
- If you can’t make it to your section on a particular week, you may attend a different section for your attendance credit (please email both your section TAs and the TAs who run the section you will attend instead)
- If you can’t make it to any sections in a particular week and have a valid excuse, email your section TAs and they will waive the requirement

Office hours

- See the course calendar! Usually regular, but changes may happen.
- Locations are TBD
- Queue here: **http://jonathansliu.com/**
- Some office hours may be dedicated to a specific purpose (e.g., written questions only, programming only, non-homework questions only, TBD)

Midterm exams

- **New this semester:** Out-of-class midterms!

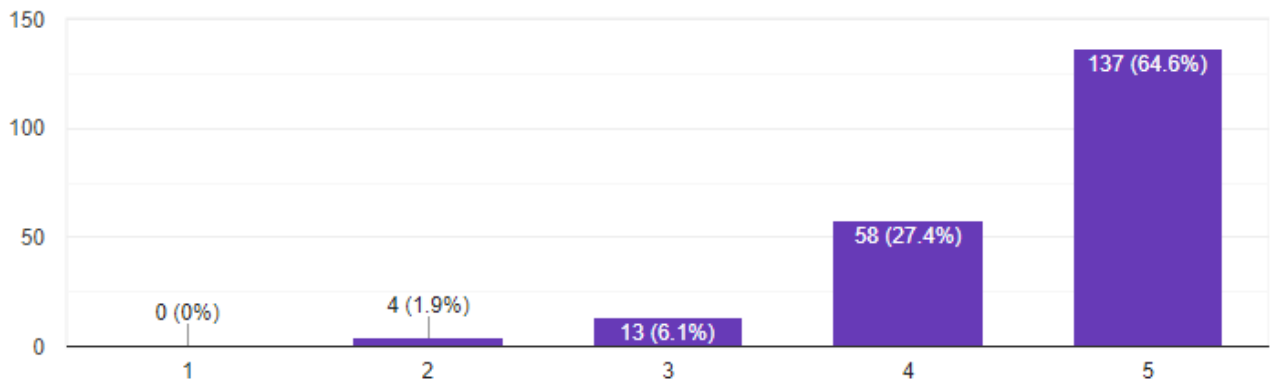
- ***Midterm one: Tuesday September 26th (Week 5) at 7:00pm***
- ***Midterm two: Thursday November 9th (Week 10) at 7:00pm***

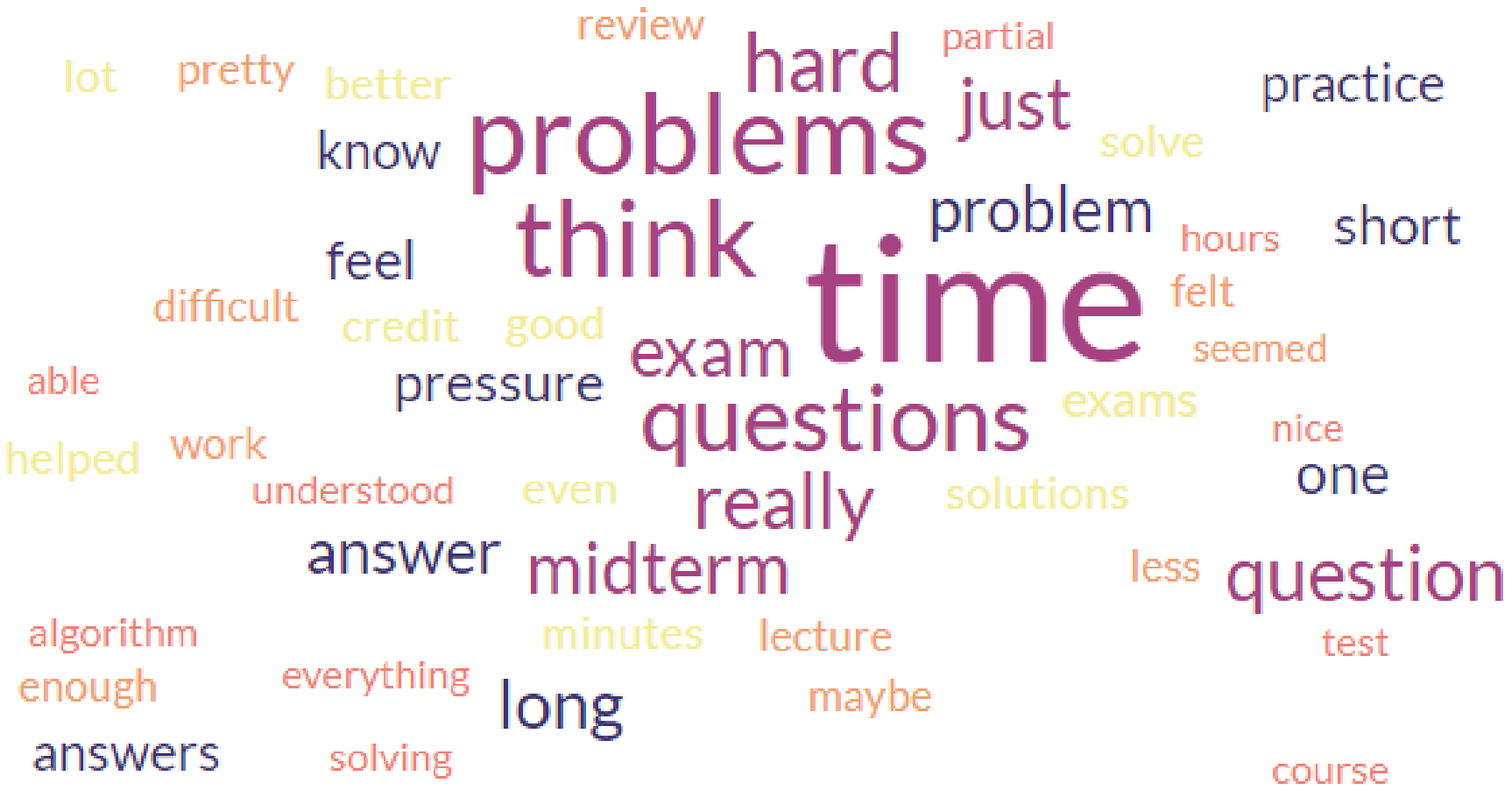
- The midterms were previously 80-minutes long (in class time)

How did you feel about the time pressure on midterm 1? (3 = mild time pressure)

Copy

212 responses




review
partial
lot
pretty
better
hard
practice
just
know
problems
solve
problem
think
short
hours
feel
time
felt
difficult
credit
good
exam
seemed
able
pressure
questions
exams
nice
helped
work
one
understood
even
really
solutions
answer
midterm
less
question
algorithm
minutes
lecture
test
enough
everything
long
maybe
answers
solving
course

Midterm timing

- The midterms will be designed to be similar in difficulty to the previous semesters (i.e., challenging to complete in 80 minutes…)
- However, the exam is now scheduled for **7:00pm – 9:30pm**, more than double time!
- Lastly, we plan to **not** put a hard stop at 9:30pm. If you need slightly longer, you're welcome to keep going a bit more until we need to go home for bed :)

Goals of the course

Learn how to design algorithms and formally analyze them in several different models / scenarios. Also practice implementing some!

- **Algorithm design techniques:** Dynamic programming, hashing and data structures, randomization, network flows, linear programming
- **Analysis:** Recurrences, probabilistic analysis, amortized analysis
- **Models**: Online algorithms, approximation algorithms, lower bounds

Now onto the algorithms!

Formal analysis of algorithms

- We want **provable guarantees** about the properties of algorithms
  - E.g., **prove** that it runs in a certain amount of **time**
  - E.g., **prove** that it outputs the correct **answer**

- **Important question:** How exactly do we measure time?
  - **Answer**: It depends :)
  - More discussion about this in the coming lectures

- ***Today: The comparison model.*** Given an algorithm whose input is an array of elements from some totally ordered set, we count the number of comparisons required to produce the output.

Today's problem: Median / k^{th} smallest

Problem (Median) Given a range of distinct numbers $a_1, a_2, \ldots, a_n$, output the median.

Definition (Median) The median is the element such that exactly $\lfloor n/2 \rfloor$ elements are larger

- More generally, we can try to solve the "k^{th} smallest" problem. Given a range of distinct elements and an integer k, we want to find the element such that there are exactly k smaller elements

Algorithm design strategy

Algorithm design idea: Start with a simple but inefficient algorithm, then optimize and remove unnecessary steps.

Simple algorithm (k^{th} smallest): Sort the array and output element k

- This takes $O(n \log n)$ comparisons using MergeSort, QuickSort, or HeapSort to do the sorting step.
- **Redundancy**: We are finding the k^{th} *smallest* for **every** k

Idea (Partial Sorting): Rearrange the numbers such that the k^{th} number is in position k, and all elements less than the k^{th} number occur before position k

Recall: Randomized QuickSort

function quicksort($a[0 \ldots n-1]$**) {**
  select a random pivot element p = a_i for a random i
  let LESS = [a_j such that $a_j < p$]
  let GREATER = [a_j such that $a_j > p$]
  return quicksort(LESS) + [p] + **quicksort**(GREATER)
}

***Question*: If we only want the k^{th} number, what is wasteful here?**

return quicksort(LESS) + [p] + **quicksort**(GREATER)

The answer is either in here *Or the answer is in here*

The result: Randomized Quickselect

function quickselect($a[0 \ldots n-1],\ k$**) {**

select a random pivot element p = a_i for a random i

let LESS = [a_j such that $a_j < p$]

let GREATER = [a_j such that $a_j > p$]

if _____ **then return** _____

else if _____ **then return** _____

else return _____

}

Now the analysis

Theorem: The *expected* number of comparisons performed by Quickselect on an input of size n *is at most* $8n$

IMPORTANT NOTES:

Note: When analyzing randomized algorithms, we are usually interested in the ***expected value*** *over the random choices* to process a ***worst-case user input***

- i.e., we are **not** assuming that our random-number generator gives us the worst possible random numbers, and we are **not** analyzing the algorithm for a randomly chosen input.

Proof of Theorem

Warning: The proof is subtle because it uses probability. We must be careful to not make false assumptions about how probability and randomness work…

Let $T(n) =$ the ***expected*** number of comparisons performed by Quickselect on a ***worst-case input*** of size n

First attempt: Almost-correct analysis

Note: This proof is nearly, but not quite correct. It does, however, provide some useful insight that gets us closer to a correct proof.

Question: What is a (good) upper bound on the expected size of the recursive subproblem?

So, we might try the recurrence…

$$T(n) \leq$$

A better proof

Question: Let's be more precise. How often is the recursive subproblem size at most $3n/4$?

So, a better recurrence relation is

$$T(n) \leq$$

Validating the recurrence relation

$$T(n) \leq 2(n-1) + T(3n/4)$$

Summary of randomized Quickselect

function quickselect($a[0 \ldots n-1],\ k$**) {**
  select a random pivot element p = a_i for a random i
  let LESS = [a_j such that $a_j < p$]
  let GREATER = [a_j such that $a_j > p$]

  if $|\text{LESS}| \geq k$ **then return quickselect**(LESS, k)
  else if $|\text{LESS}| = k$ **then return** p
  else return quickselect(RIGHT, $k - |\text{LESS}| - 1$)
}

- Runs in $O(n)$ **expected** time in the **comparison model**.
- More tightly, uses at most $8n$ comparisons in expectation.
- An as exercise, the analysis can be improved to $4n$ comparisons.

Question break

How about a deterministic algorithm?

- Where was the randomness in Randomized QuickSelect? How can we get rid of it?
- What if we could deterministically find the optimal pivot? What would that be? The median! Oh…

What we need: In $O(n)$ comparisons, we need to find a "good" pivot. A good pivot would leave us with cn elements in the recursive call, for some fraction $c < 1$, e.g., $3n/4$ elements is good.

Picking a good pivot

- Picking the median as the pivot is too much to ask for, so we want some kind of "approximate median"

Idea (doesn't quite work, but very close): Pick the median of a smaller subset of the input (faster to find) then hope that it is a good approximation to the true median.

Question: What if we find the median of half of the elements?

Median of half

If we pivot on the median of half of the elements, the number of comparisons will be

$$T(n) \leq$$

Exercise: Show that picking any constant-fraction sized subset (e.g., a quarter, one tenth) and taking the median doesn't work.

We need to go deeper!

Note: This idea is extremely subtle. It took four Turing Award winners to figure it out. We don't expect that you would produce this algorithm on your own.

- Finding the median of a smaller set **almost** worked, but it was just a bit too much work since the "approximate median" wasn't good enough.

Huge idea (median of medians): Find the medians of several small subsets of the input, then find the **median of those medians**.

Median of medians algorithm

function DeterministicSelect($a[0 \ldots n-1],\ k$**) {**
  group the array into $n/5$ groups of size 5, find the median of each group
  recursively find the median of these medians, call it p

  // Below is the same as Randomized Quickselect
  let LESS = [a_j such that $a_j < p$]
  let GREATER = [a_j such that $a_j > p$]
  if $|\text{LESS}| \geq k$ **then return DeterministicSelect**(LESS, k)
  else if $|\text{LESS}| = k$ **then return** p
  else return DeterministicSelect(RIGHT, $k - |\text{LESS}| - 1$)
}

How good is the median of medians?

Theorem: The median of medians is larger than at least $3/10^{\text{ths}}$ of the input, and smaller than at least $3/10^{\text{ths}}$ of the input

Analysis of DeterministicSelect

Theorem: The number of comparisons performed by DeterministicSelect on an input of size n *is* $O(n)$

1. Find the median of $n/5$ groups of size 5
2. Recursively find the median of medians
3. Split the input into LESS and GREATER
4. Recurse on the appropriate piece

$$T(n) \leq$$

Solving the recurrence

$$T(n) \leq cn + T(n/5) + T(7n/10)$$

So, the total running time is…

$$T(n) \leq cn(1 + (9/10) + (9/10)^2 + (9/10)^3 + \cdots)$$

Summary of DeterministicSelect

function DeterministicSelect($a[0 \ldots n-1],\ k$**) {**
  group the array into $n/5$ groups of size 5,
    find the median of each group
  recursively find the median of these medians, call it p

  // Below is the same as Randomized Quickselect
  let LESS = [a_j such that $a_j < p$]
  let GREATER = [a_j such that $a_j > p$]
  if $|\text{LESS}| \geq k$ **then return DeterministicSelect**(LESS, k)
  else if $|\text{LESS}| = k$ **then return** p
  else return DeterministicSelect(RIGHT, $k - |\text{LESS}| - 1$)
}

- The median of medians is the key ingredient for getting a deterministic algorithm
- To analyze the recurrence, we used the "stack of bricks" method.
- We could also prove it by induction, but this requires us to know the runtime already

Take-home messages for today

- Recursion is powerful, randomization is powerful.
- Analyzing ***randomized recursive algorithms*** is tricky. Be careful with expected values!!
- Analyzing runtime via ***recurrence relations*** is very useful. We can do it with several different techniques:
  - Guess the answer and verify via induction
  - Expand it out / “stack of bricks” if we don’t already know the answer
  - You probably learned even more techniques in your prerequisite classes