Lecture 5
Substitution method, and randomized algorithms!
Announcements

• HW2 is posted! Due Friday.

• Please send any OAE letters to Luna Frank-Fischer (luna16@stanford.edu) by April 28.

• Lines at office hours: we know they are long.
  • We will convert some office hours to “group style.”
    • some will stay as individual using QueueStatus.
    • keep an eye on the Google calendar.
  • Go to office hours earlier in the week.
  • Go with a “buddy” (who has the same questions).
Thanks for filling out that poll!

- Feedback on pace:

  ![Poll Results]

  - **Just right**: 30 / 38%
  - **Fast, but I like the challenge**: 15 / 19%
  - **Slow, but it’s good: I’m really understanding stuff**: 13 / 16%
  - **Too fast, I’m lost**: 8 / 10%
  - **Too slow, I’m bored.**: 7 / 9%
  - **Slow, but it’s good: I’m getting a chance to catch up on Facebook**: 6 / 8%

- So I’m not going to change the pace of lectures.

- **BUT!!**
If you think lectures are too fast

• You are not alone.
• Read the book and lecture notes *before coming to lecture.*
• Go to *discussion sections.*
• Go to *office hours.*
If you think lectures are too slow

• You are not alone.

• I’ll try to put fun problems on the side of slides for you to think about.
  
  (Also you can find all the typos in my slides and email them to me) 😊

Note: even if you don’t think lectures are too slow, you can go back and look at these problems afterwards!

Are there functions $f(n)$ and $g(n)$ that are both increasing, but so that $f(n)$ is neither $O(g(n))$ nor $Ω(g(n))$?
Other things I will change

• From now on, homework questions will all explicitly say what sort of answer we are expecting.

• I recognize I need to do better with **pacing lectures**.
  • I’ve been getting bogged down with details at the beginning and have to rush at the end.
  • I will try to focus on the high-level points (unless I think the technical details are very important). Please see CLRS, lecture notes, or office hours for omitted technical details.

• I will try to make fewer typos on slides. [sic]

• I will skew slightly toward slides.

• I will post another poll in a few weeks.

**How is the mix between slides and board work?**

78 out of 80 people answered this question

<table>
<thead>
<tr>
<th>Option</th>
<th>Votes</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Just right</td>
<td>39</td>
<td>50%</td>
</tr>
<tr>
<td>2 I’d like more slides</td>
<td>31</td>
<td>40%</td>
</tr>
<tr>
<td>3 I’d like more board</td>
<td>5</td>
<td>6%</td>
</tr>
<tr>
<td>4 Neither are really working</td>
<td>3</td>
<td>4%</td>
</tr>
</tbody>
</table>
Let’s get a move-on...

• **Last time:** we saw a cool (and complex!) recursive algorithm for solving SELECT.

  \[
  \text{A is an array of size } n, \text{ } k \text{ is in } \{1, \ldots, n\}
  \]

  • **SELECT**(A, k):
    • Return the k’th smallest element of A.

• **One idea:** Use MergeSort and take the k’th smallest.
  • Time O(n log(n)). *Can we do better??*

• **Idea:** pick a **pivot** that’s close to the median, and recurse on either side of the pivot.

• **Cool trick:** Use recursion to also pick the pivot!

• **CLAIM:** This runs in time O(n).
Last time we ended up with this:

\[ T(n) \leq c \cdot n + T \left( \frac{n}{5} \right) + T \left( \frac{7n}{10} + 5 \right) \]

• How can we solve this?
• The sub-problems **don’t have the same size.**
  • The master method doesn’t work.
  • Recursion trees get complicated.
• The **substitution method** gives us a way.
  • fancy “guess-and-check”

The cn is the O(n) work done at each level for PARTITION

The T(n/5) is for the recursive call to get the median in FINDPIVOT

The T(7n/10 + 5) is for the recursive call to SELECT for either L or R.

Try solving this using a recursion tree!

Ollie the over-achieving ostrich
The substitution method (by example)

- example: \( T(n) \leq 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right) \),
  - with \( T(n) = 10n \) for \( n < 10 \).
- First, make a guess about the answer.
- Check your guess using induction.
  - Suppose that your guess holds for all \( k < n \).
  - \( T(n) \leq 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right) \)
  - \( T(n) \leq 3n + 10\left(\frac{n}{5}\right) + 10\left(\frac{n}{2}\right) \)
  - \( T(n) \leq 3n + 2n + 5n = 10n. \)
  - This establishes the inductive hypothesis for \( n \).
  - (And the base case is satisfied: \( T(n) \leq 10n \) for \( n < 10 \).)
  - So \( T(n) = O(n) \).
How did we come up with that hypothesis?

• Doesn’t matter for the correctness of the argument, but..
  • Be very lucky.
  • Play around with the recurrence relation to try to get an idea before you start.
  • Start with a hypothesis with a variable in it, and try to solve for that variable at the end.
Example of how to come up with a guess.

• First, make a guess about what the correct term should be: but leave a variable “C” in it, to be determined later.

• example: $T(n) \leq 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$,

  • with $T(n) = 10n$ for $n < 10$.

• Check your guess using induction.

  • Suppose that your guess holds for all $k < n$.

  • $T(n) \leq 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$

  • $T(n) \leq 3n + C\left(\frac{n}{5}\right) + C\left(\frac{n}{2}\right)$

  • $T(n) \leq 3n + \frac{Cn}{5} + \frac{Cn}{2}$.

  • If I want that to be $Cn$, then I can solve for $C$...
Back to SELECT

\[ T(n) \leq c \cdot n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 5\right) \]

The \( c \cdot n \) is the \( O(n) \) work done at each level for PARTITION

The \( T(n/5) \) is for the recursive call to get the median in FINDPIVOT

The \( T(7n/10 + 5) \) is for the recursive call to SELECT for either L or R.

Inductive hypothesis (aka our guess):

\[ T(n) \leq \begin{cases} 
  d \cdot 100 & \text{if } n \leq 100 \\
  d \cdot n & \text{if } n > 100 
\end{cases} \]

(aka, \( T(n) = O(n) \)).

for \( d = 20c \).

How on earth did we come up with this? Try to arrive at this guess on your own.

Ollie the over-achieving ostrich
Finally, let’s prove we can do SELECT in time $O(n)$

(*) $T(k) \leq \begin{cases} 
  d \cdot 100 & \text{if } k \leq 100 \\
  d \cdot k & \text{if } k > 100 
\end{cases}$

for $d = 20c$.

• Base case:
  • If $n \leq 50$, we can assume our alg. takes time $\leq 50d$.
    • (You should justify: WHY IS THIS OKAY?)

• Inductive step: Suppose (*) holds for all sizes $k < n$. Then

$$T(n) \leq c \cdot n + T \left( \frac{n}{5} \right) + T \left( \frac{7n}{10} + 5 \right) \leq c \cdot n + d \cdot \frac{n}{5} + d \cdot \left( \frac{7n}{10} + 5 \right) \leq n \left( c + \frac{d}{5} + \frac{7d}{10} \right) + 5d \leq n \left( c + \frac{20c}{5} + \frac{140c}{10} \right) + 100 \cdot c = (19 \cdot n + 100) \cdot c \leq 20c \cdot n \text{ whenever } n > 100.$$ 

This is pretty pedantic! But it’s worth being careful about the constants when doing inductive arguments. (see: your homework).

Here come some computations: no need to pay too much attention, just know that you can do these computations.
Nearly there!

• By induction, the inductive hypothesis (*) applies for all n.

• **Termination**: Observe that this is exactly what we wanted to show!
  • There exists:
    • a constant $d > 0$ (which depends on the constant $c$ from the running time of PARTITION...)
    • an $n_0$ (aka 101)
  • so that for all $n \geq n_0$, $T(n) \leq d \cdot n$.
  • By definition, $T(n) = O(n)$.
  • Hooray!

• Conclusion:
  
  We can implement SELECT in time $O(n)$.
Quick recap before we move on

• We can do SELECT (in particular, MEDIAN) in time $O(n)$.
• We analyzed this with the substitution method.

Next up:

• Randomized algorithms.
Randomized algorithms

• The algorithm gets to use randomness.
• It should always be correct (for this class).
• But the runtime can be a random variable.

• We’ll see a few randomized algorithms for sorting.
  • BogoSort
  • QuickSort

• BogoSort is a pedagogical tool.
• QuickSort is important to know. (in contrast with BogoSort...)
Example of a randomized sorting algorithm

• **BogoSort(A):**
  • While true:
    • Randomly permute A.
    • Check if A is sorted.
    • If A is sorted, return A.

• **This algorithm is always correct:**
  • If it returns, then it returns a sorted list.

• **Informal Runtime Analysis (and probability refresher):**
  • $E[\text{runtime}] = ?$
  • $\Pr[\text{randomly permuted array is sorted}] = ?$
    • $1/n!$
  • We expect to permute A $n!$ times before it’s sorted.
  • $E[\text{runtime}] = O(n \cdot n!) = \text{BIG}.$
  • Worst-case runtime?
    • Infinity!

Suppose that you can draw a random integer in $\{1,...,n\}$ in time $O(1)$. How would you randomly permute an array in-place in time $O(n)$?

Ollie the over-achieving ostrich

We expect to roll a 6-sided die 6 times before we see a 1. We expect to flip a fair coin twice before we see heads.

Worst case means that an adversary chooses the randomness.
Example of a better randomized algorithm: QuickSort

- Runs in expected time $O(n\log(n))$.
- Worst-case runtime $O(n^2)$.
- Easier to implement than MergeSort, and the constant factors inside the $O()$ are very small.
- In practice often more desirable.
Quicksort

We want to sort this array.

First, pick a "pivot."
Do it at random.

Next, partition the array into "bigger than 5" or "less than 5"

Arrange them like so:

Recurse on L and R:

This PARTITION step takes time $O(n)$. (Notice that we don’t sort each half). [same as in SELECT]
PseudoPseudoCode
for what we just saw

• **QuickSort(A):**
  • **If** len(A) \( \leq 1: \)
    • **return**
  • Pick some \( x = A[i] \) at random. Call this the pivot.
  • **PARTITION** the rest of A into:
    • L (less than x) and
    • R (greater than x)
  • Replace A with \( [L, x, R] \) (that is, rearrange A in this order)
  • **QuickSort(L)**
  • **QuickSort(R)**

See CLRS for more detailed pseudocode.

How would you do all this in-place in time \( O(n) \)?

Ollie the over-achieving ostrich
Example of recursive calls

Pick 5 as a pivot

Partition on either side of 5

Recurse on [3142] and pick 3 as a pivot.

Partition around 3.

Recurse on [12] and pick 2 as a pivot.

Partition around 2.

Recurse on [1] (done).


Recurse on [7], it has size 1 so we’re done.
How long does this take to run?

• We will count the number of comparisons that the algorithm does.
  • This turns out to give us a good idea of the runtime. (Not obvious).

• How many times are any two items compared?

In the example before, everything was compared to 5 once in the first step....and never again.

But not everything was compared to 3. 5 was, and so were 1,2 and 4. But not 6 or 7.
Each pair of items is compared either 0 or 1 times. Which is it?

| 7 | 6 | 3 | 5 | 1 | 2 | 4 |

Let’s assume that the numbers in the array are actually the numbers 1,…,n.

Of course this doesn’t have to be the case! It’s a good exercise to convince yourself that the analysis will still go through without this assumption. (Or see CLRS)

• Whether or not a,b are compared is a random variable, that depends on the choice of pivots. Let’s say

\[ X_{a,b} = \begin{cases} 1 & \text{if } a \text{ and } b \text{ are ever compared} \\ 0 & \text{if } a \text{ and } b \text{ are never compared} \end{cases} \]

• In the previous example \( X_{1,5} = 1 \), because item 1 and item 5 were compared.
• But \( X_{3,6} = 0 \), because item 3 and item 6 were NOT compared.
• Both of these depended on our random choice of pivot!
Counting comparisons

• The number of comparisons total during the algorithm is

\[
\sum_{a=1}^{n} \sum_{b=a+1}^{n} X_{a,b}
\]

• The expected number of comparisons is

\[
E \left[ \sum_{a=1}^{n} \sum_{b=a+1}^{n} X_{a,b} \right] = \sum_{a=1}^{n} \sum_{b=a+1}^{n} E[X_{a,b}] 
\]

using linearity of expectations.
Counting comparisons

- So we just need to figure out $E[X_{a,b}]$
- $E[X_{a,b}] = P(X_{a,b} = 1) \cdot 1 + P(X_{a,b} = 0) \cdot 0 = P(X_{a,b} = 1)$
  - (using definition of expectation)
- So we need to figure out
  $P(X_{a,b} = 1) =$ the probability that $a$ and $b$ are ever compared.

Say that $a = 2$ and $b = 6$. What is the probability that 2 and 6 are ever compared?

This is exactly the probability that either 2 or 6 is first picked to be a pivot out of the highlighted entries.

If, say, 5 were picked first, then 2 and 6 would be separated and never see each other again.

\[
\sum_{a=1}^{n} \sum_{b=a+1}^{n} E[X_{a,b}]
\]
Counting comparisons

\[ P(X_{a,b} = 1) \]

= probability \( a, b \) are ever compared

= probability that one of \( a, b \) are picked first out of all of the \( b - a + 1 \) numbers between them.

\[ = \frac{2}{b - a + 1} \]

2 choices out of \( b-a+1 \)...
All together now...

**Expected number of comparisons**

- $E\left[\sum_{a=1}^{n} \sum_{b=a+1}^{n} X_{a,b}\right]$  
  This is the expected number of comparisons throughout the algorithm

- $= \sum_{a=1}^{n} \sum_{b=a+1}^{n} E\left[X_{a,b}\right]$  
  linearity of expectation

- $= \sum_{a=1}^{n} \sum_{b=a+1}^{n} P(X_{a,b} = 1)$  
  definition of expectation

- $= \sum_{a=1}^{n} \sum_{b=a+1}^{n} \frac{2}{b-a+1}$  
  the reasoning we just did

- This is a big nasty sum, but we can do it.

- We get that this is less than $2n \ln(n)$. 

Do this sum!

Ollie the over-achieving ostrich
Are we done?

• We saw that \( E[ \text{number of comparisons} ] = O(n \log(n)) \)
• Is that the same as \( E[ \text{running time} ] \)?

• In this case, \textbf{yes}.

• We need to argue that the running time is dominated by the time to do comparisons.

• \textbf{QuickSort(A)}:
  • If \( \text{len}(A) \leq 1 \):
    • return
  • Pick some \( x = A[i] \) at random. Call this the \textit{pivot}.
  • PARTITION the rest of \( A \) into:
    • \( L \) (less than \( x \)) and
    • \( R \) (greater than \( x \))
  • Replace \( A \) with \([L, x, R]\) (that is, rearrange \( A \) in this order)
  • QuickSort(L)
  • QuickSort(R)

• (See CLRS for details).
Worst-case running time for QuickSort (if time)

• Suppose that an adversary is choosing the random pivots for you.

• Then the running time might be \( O(n^2) \) [on board]

• In practice, this doesn’t usually happen.

• **Aside**: We worked really hard last week to get a deterministic algorithm for SELECT, by picking the pivot very cleverly.

• What happens if you pick the pivot randomly?

• Turns out this is also usually a good idea.
Recap

• We can do SELECT and MEDIAN in time $O(n)$.
• We already knew how to sort in time $O(n \log(n))$ with MergeSort.

• The randomized algorithm QuickSort also runs in expected time $O(n \log(n))$.
• In practice, QuickSort is often nicer.

• Skills of today:
  • substitution method
  • analysis of randomized algorithms.

Next time

• Could we sort faster than $O(n \log(n))$??