Lecture 4

The Substitution Method and Median and Selection
Announcements!

- HW1 due Friday.
  - (And HW2 also posted Friday).
Last Time: **The Master Theorem**

- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} 
O(n^d \log(n)) & \text{if } a = b^d \\
O(n^d) & \text{if } a < b^d \\
O(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases}$$

Three parameters:
- $a$ : number of subproblems
- $b$ : factor by which input size shrinks
- $d$ : need to do $n^d$ work to create all the subproblems and combine their solutions.
Today
more recursion, beyond the Master Theorem.

• The Master Theorem only works when all sub-problems are the same size.
• That’s not always the case.
• Today we’ll see an example where the Master Theorem won’t work.
• We’ll use something called the substitution method instead.

I can handle all the recurrence relations that look like
\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + O(n^d). \]
Before this theorem I was but the learner. Now I am the master.

*More precisely, only a master of same-size sub-problems...still pretty handy, actually.
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
4. Return of the **Substitution Method**.
A non-tree method

Here’s another way to solve:

- \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \)
- \( T(0) = 0, \ T(1) = 1 \)

1. Guess what the answer is.
2. Formally prove that that’s what the answer is.

For most of this lecture, division is integer division: \( \frac{n}{2} \) means \( \left\lfloor \frac{n}{2} \right\rfloor \). We’ll be pretty sloppy about the difference.

On your HW, you’ll prove why most of the time this doesn’t matter.

You did this for your pre-lecture exercise! Let’s go through it now quickly to make sure we are all on the same page.
• \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \)

• \( T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \)

• \( T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot n \)

• \( T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2 \cdot n \)

• \( T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 3 \cdot n \)

• Following the pattern...
  \[ T(n) = n \cdot T(1) + \log(n) \cdot n = n(\log(n) + 1) \]

So that is our guess!
2. Formally prove that that’s what the answer is

• Inductive hypothesis:
  • $T(k) \leq k(\log(k) + 1)$ for all $1 \leq k \leq n$

• Base case:
  • $T(1) = 1 = 1(\log(1) + 1)$

• Inductive step:
  • $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$
    \[
    = 2 \left(\frac{n}{2} \left(\log\left(\frac{n}{2}\right) + 1\right)\right) + n \\
    = 2 \left(\frac{n}{2} \left(\log(n) - 1 + 1\right)\right) + n \\
    = 2 \left(\frac{n}{2} \log(n)\right) + n \\
    = n(\log(n) + 1)
    \]

• Conclusion:
  • By induction, $T(n) = n(\log(n) + 1)$ for all $n > 0$. 

What happened between these two lines?
That’s called the substitution method

• So far, just seems like a different way of doing the same thing.

• But consider this!

\[ T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right) \]

\[ T(n) = 10n \text{ when } 1 \leq n \leq 10 \]
Gross!

- Let’s try the same unwinding thing to get a feel for it.
  
  - [On board]

- Okay, that gets gross fast. We can also just try it out.
  
  - [iPython Notebook]

- What else do we know?:
  
  - \( T(n) \leq 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right) \)
    
    \[ \leq 3n + 2 \cdot T\left(\frac{n}{2}\right) \]
    
    \[ = O(n \log(n)) \]
  
  - \( T(n) \geq 3n \)
  
  - So the right answer is somewhere between \( O(n) \) and \( O(n \log(n)) \)...
Let’s guess \( \Theta(n) \)
And try to prove it!

- **Inductive Hypothesis:** \( T(k) \leq Ck \) for all \( k < n \).
- **Base case:** \( T(k) \leq Ck \) for all \( k \leq 10 \)
- **Inductive step:**
  - \( T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right) \)
  - \( \leq 3n + C\left(\frac{n}{5}\right) + C\left(\frac{n}{2}\right) \)
  - \( \leq 3n + \frac{c}{5}n + \frac{c}{2}n \)
  - \( \leq Cn \) ??

- **Conclusion:**
  - There is some \( C \) so that for all \( n \geq 1, T(n) \leq Cn \)
  - Aka, \( T(n) = \Theta(n) \). 

\[ T(n) = 10n \text{ when } 1 \leq n \leq 10 \]
Now pretend like we knew it all along.

**Theorem:** \( T(n) = O(n) \)

**Proof:**

- Inductive Hypothesis: \( T(k) \leq 10k \) for all \( k < n \).
- Base case: \( T(k) \leq 10k \) for all \( k \leq 10 \).
- Inductive step:
  - \( T(n) = 3n + T \left( \frac{n}{5} \right) + T \left( \frac{n}{2} \right) \)
  - \( T(n) \leq 3n + 10 \left( \frac{n}{5} \right) + 10 \left( \frac{n}{2} \right) \)
  - \( T(n) \leq 3n + 2n + 5n = 10n. \)
- Conclusion:
  - For all \( n \geq 1, T(n) \leq 10n \), aka \( T(n) = O(n) \).
What have we learned?

• The substitution method can work when the master theorem doesn’t.
  • For example with different-sized sub-problems.

• Step 1: generate a guess
  • Throw the kitchen sink at it.

• Step 2: try to prove that your guess is correct
  • You may have to leave some constants unspecified till the end – then see what they need to be for the proof to work!!

• Step 3: profit
  • Pretend you didn’t do Steps 1 and 2 and write down a nice proof.
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
4. Return of the **Substitution Method**.
The problem we will solve

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

- **SELECT** \( (A, k) \):
  - Return the \( k \)'th smallest element of \( A \).

- **SELECT** \( (A, 1) \) = 1
- **SELECT** \( (A, 2) \) = 3
- **SELECT** \( (A, 3) \) = 4
- **SELECT** \( (A, 8) \) = 14

- **SELECT** \( (A, 1) \) = MIN(A)
- **SELECT** \( (A, n/2) \) = MEDIAN(A)
- **SELECT** \( (A, n) \) = MAX(A)

For today all arrays have distinct elements.

Being sloppy about floors and ceilings!
We’re gonna do it in time $O(n)$

• Let’s start with $\text{MIN}(A)$ aka $\text{SELECT}(A, 1)$.

• $\text{MIN}(A)$:
  • $\text{ret} = \infty$
  • For $i = 0, \ldots, n-1$:
    • If $A[i] < \text{ret}$:
      • $\text{ret} = A[i]$
  • Return $\text{ret}$

• Time $O(n)$. Yay!
How about SELECT(A, 2)?

• SELECT2(A):
  • ret = ∞
  • minSoFar = ∞
  • For i=0, .., n-1:
    • If A[i] < ret and A[i] < minSoFar:
      • ret = minSoFar
      • minSoFar = A[i]
    • Else if A[i] < ret and A[i] >= minSoFar:
      • ret = A[i]
  • Return ret

(The actual algorithm here is not very important because this won’t end up being a very good idea...)

Still O(n)
SO FAR SO GOOD.
SELECT(A, n/2) aka MEDIAN(A)?

- MEDIAN(A):
  - ret = ∞
  - minSoFar = ∞
  - secondMinSoFar = ∞
  - thirdMinSoFar = ∞
  - fourthMinSoFar = ∞
  - ....

- This is not a good idea for large k (like n/2 or n).

- Basically this is just going to turn into something like INSERTIONSORT...and that was O(n²).
A much better idea for large k

• **SELECT**(A, k):
  • A = **MergeSort**(A)
  • **return** A[k]

• Running time is \(O(n \log(n))\).
• So that’s the benchmark....

Can we do better?

We’re hoping to get \(O(n)\)
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
4. Return of the **Substitution Method**.
Idea: divide and conquer!

Say we want to find $\text{SELECT}(A, k)$

First, pick a "pivot." We’ll see how to do this later.

Next, partition the array into “bigger than 6” or “less than 6”

$L = \text{array with things smaller than } A[\text{pivot}]$

$R = \text{array with things larger than } A[\text{pivot}]$

How about this pivot?

This PARTITION step takes time $O(n)$. (Notice that we don’t sort each half).
Idea: divide and conquer!

Say we want to find $\text{SELECT}(A, k)$

First, pick a “pivot.” We’ll see how to do this later.

Next, partition the array into “bigger than 6” or “less than 6”

$L = \text{array with things smaller than } A\{\text{pivot}\}$

$R = \text{array with things larger than } A\{\text{pivot}\}$
Idea continued...

Say we want to find \( \text{SELECT}(A, k) \)

L = array with things smaller than \( A[pivot] \)

R = array with things larger than \( A[pivot] \)

- If \( k = 5 = \text{len}(L) + 1 \):
  - We should return \( A[pivot] \)
- If \( k < 5 \):
  - We should return \( \text{SELECT}(L, k) \)
- If \( k > 5 \):
  - We should return \( \text{SELECT}(R, k - 5) \)

This suggests a recursive algorithm

(Still need to figure out how to pick the pivot...)
Pseudocode

• **Select**(A,k):
  • If len(A) <= 50:
    • A = **MergeSort**(A)
    • Return A[k-1]
  • p = **getPivot**(A)
  • L, pivotVal, R = **Partition**(A,p)
  • if len(L) == k-1:
    • return pivotVal
  • Else if len(L) > k-1:
    • return **Select**(L, k)
  • Else if len(L) < k-1:
    • return **Select**(R, k – len(L) – 1)

**Base Case:** If the len(A) = O(1), then any sorting algorithm runs in time O(1).

**Case 1:** We got lucky and found exactly the k’th smallest value!

**Case 2:** The k’th smallest value is in the first part of the list

**Case 3:** The k’th smallest value is in the second part of the list

• **getPivot**(A) returns some pivot for us.
  • How?? We’ll see later…

• **Partition**(A,p) splits up A into L, A[p], R.
  • See Lecture 4 notebook for code
Let’s make sure it works

• [iPython Notebook for Lecture 4]
Now we should be convinced

- No matter what procedure we use for $\text{getPivot}(A)$, $\text{Select}(A,k)$ returns a correct answer.

Formally prove the correctness of $\text{Select}$!
What is the running time?

\[ T(n) = \begin{cases} 
T(\text{len}(L)) + O(n) & \text{len}(L) < k - 1 \\
T(\text{len}(R)) + O(n) & \text{len}(L) > k - 1 \\
O(n) & \text{len}(L) = k - 1 
\end{cases} \]

• What are \text{len}(L) and \text{len}(R)?
  • That depends on how we pick the pivot...
  • \textbf{[On board]} What’s the best case? What’s the worst case?
In an ideal world...

- We split the input in half:
  - \( \text{len}(L) = \text{len}(R) = (n-1)/2 \)

- Let’s use the **Master Theorem**!
  - \( T(n) \leq T\left(\frac{n}{2}\right) + O(n) \)
  - So \( a = 1, \ b = 2, \ d = 1 \)
  - \( T(n) \leq O(n^d) = O(n) \)

- Suppose \( T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d) \). Then

\[
T(n) = \begin{cases} 
  O(n^d \log(n)) & \text{if } a = b^d \\
  O(n^d) & \text{if } a < b^d \\
  O(n^\log_b(a)) & \text{if } a > b^d 
\end{cases}
\]
But the world is not ideal.

- Suppose we choose a pivot uniformly at random, but then a bad guy who knows what pivots we will choose gets to come up with $A$.
- [Discussion on board]
The distinction matters!

See Lecture 4 iPython notebook for code that generated this picture.
Aside:
actually, the world is pretty ideal most of the time.

• In most situations, a bad guy doesn’t pick the input.
• And even if a bad guy does pick the input, they might not know what pivots you’ll choose in advance.
• In those situations, picking a random pivot works just fine!
But for today

• Let’s assume there’s this bad guy.
• We’ll get a **stronger guarantee**
• We’ll get to see a **really clever algorithm**
• And we’ll get more practice with the **substitution method**.
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise

2. The **SELECT** problem.

3. The **SELECT** solution.
   a) The outline of the algorithm.
   b) How to pick the pivot.

4. Return of the **Substitution Method**.
How should we pick the pivot?

- We’d like to live in the ideal world.
- Pick the pivot to divide the input in half!
- Aka, pick the median!
- Aka, pick $\text{Select}(n, n/2)$
How should we pick the pivot?

• We’d like to **approximate** the ideal world.

• Pick the pivot to divide the input **about** in half!
• Maybe this is easier!
In an ideal world...

- We split the input not quite in half:
  - \(3n/10 < \text{len}(L) < 7n/10\)
  - \(3n/10 < \text{len}(R) < 7n/10\)

- If we could do that, the **Master Theorem** would say:
  - \(T(n) \leq T\left(\frac{7n}{10}\right) + O(n)\)
  - So \(a = 1\), \(b = 10/7\), \(d = 1\)
  - \(T(n) \leq O(n^d) = O(n)\)

STILL GOOD!
Goal

• Pick the pivot so that

$L = \text{array with things smaller than } A[\text{pivot}]$

$R = \text{array with things larger than } A[\text{pivot}]$

$\frac{3n}{10} < \text{len}(L) < \frac{7n}{10}$

$\frac{3n}{10} < \text{len}(R) < \frac{7n}{10}$
Another divide-and-conquer alg!

• We can’t solve `Select(n, n/2)` (yet)
• But we can *divide and conquer* and solve `Select(m, m/2)` for smaller values of `m`. (recursively)

• **Lemma**: The median of sub-medians is close to the median.

*we will make this a bit more precise.*
How to pick the pivot

• **CHOOSEPIVOT(A):**
  • Split A into $m = \lceil \frac{n}{5} \rceil$ groups, of size $\leq 5$ each.
  • For $i=1, \ldots, m$:
    • Find the median within the $i$’th group, call it $p_i$
    • $p = \text{SELECT}(\ [p_1, p_2, p_3, \ldots, p_m], \ m/2 )$
  • return $p$

This takes time $O(1)$, since each group has size 5

Pivot is $\text{SELECT}(\ [8, 4, 5, 6, 12], \ 3 \ ) = 6$:

PARTITION around that 5:

This part is L

This part is R: it’s almost the same size as L.
CLAIM: this works divides the array *approximately* in half

- Empirically (see Lecture 4 iPython Notebook):
CLAIM: this works
divides the array *approximately* in half

- Formally, we will prove (later):

  **Lemma:** If we choose the pivots like this, then

  \[ |L| \leq \frac{7n}{10} + 5 \]

  and

  \[ |R| \leq \frac{7n}{10} + 5 \]
Does that seem true?

\[ |L| \leq \frac{7n}{10} + 5 \text{ and } |R| \leq \frac{7n}{10} + 5 \]

Actually in practice (on randomly chosen arrays) it looks even better!

But this is a worst-case bound.
How about the running time?

• Suppose the Lemma is true. (It is).
  • $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$

• Recurrence relation:
  $$T(n) \leq ?$$
Pseudocode

- **getPivot( A )** returns some pivot for us.
  - How?? We’ll see later...
- **Partition( A, p )** splits up A into L, A[p], R.
  - See Lecture 4 notebook for code

- **Select( A, k ):**
  - **If** len( A ) <= 50:
    - A = **MergeSort( A )**
    - **Return** A[k-1]
  - p = getPivot( A )
  - L, pivotVal, R = **Partition( A, p )**
  - **if** len( L ) == k-1:
    - return pivotVal
  - **Else if** len( L ) > k-1:
    - return **Select( L, k )**
  - **Else if** len( L ) < k-1:
    - return **Select( R, k - len( L ) - 1 )**

**Base Case:** If the len(A) = O(1), then any sorting algorithm runs in time O(1).

**Case 1:** We got lucky and found exactly the k’th smallest value!

**Case 2:** The k’th smallest value is in the first part of the list

**Case 3:** The k’th smallest value is in the second part of the list
How about the running time?

• Suppose the Lemma is true. (It is).
  • $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$

• Recurrence relation:

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise

2. The **SELECT** problem.

3. The **SELECT** solution.
   a) The outline of the algorithm.
   b) How to pick the pivot.

4. Return of the **Substitution Method**.
This sounds like a job for... 

**The Substitution Method!**

Step 1: generate a guess  
Step 2: try to prove that your guess is correct  
Step 3: profit

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

[On board]

Conclusion: \( T(n) = O(n) \)
• Base case:
  • if \( n \leq 50 \), our algorithm was: run MergeSort on \( \leq 50 \) things.
  • By maybe allowing \( c \) to be a bit bigger, that takes time at most \( 50d \).
  
  \( \text{WHY IS THIS OKAY?} \)

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

\[
T(n) \leq c \cdot n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 5\right)
\]
\[
\leq c \cdot n + d \cdot \frac{n}{5} + d \cdot \left(\frac{7n}{10} + 5\right)
\]
\[
\leq n \left( c + \frac{d}{5} + \frac{7d}{10} \right) + 5d
\]
\[
\leq n \left( c + \frac{10c}{5} + \frac{70c}{10} \right) + 50c
\]
\[
= (9n + 50) c
\]
\[
\leq 10c \cdot n = d \cdot n \quad \text{whenever } n > 50.
\]

\[\text{Inductive Hypothesis:}\]
\[
(*) \quad T(n) \leq \begin{cases} 
  d \cdot 50 & \text{if } n \leq 50 \\
  d \cdot n & \text{if } n > 50 
\end{cases}
\]

for \( d = 10c \).
• By induction, this shows that the inductive hypothesis (*) applies for all $n$.

• **Conclusion:**
  • There exists a constant $d$ *(which depends on the constant $c$ from the running time of PARTITION...)* and an $n_0$ (aka 50) so that for all $n > n_0$, $T(n) \leq d \cdot n$.
  • By definition, $T(n) = O(n)$.
  • Hooray!
  • Therefore: We can implement SELECT (and in particular, MEDIAN) in time $O(n)$.
In practice?

- With my dumb implementation, our fancy version of **Select** is worse than **MergeSort-based Select**. 😞
  - But $O(n)$ is better than $O(n \log(n))$! How can that be?
  - What’s the constant in front of the $n$ in our proof? 20? 30?

- On non-adversarial inputs, random pivot choice is **MUCH** better.

**Moral:**
Just pick a random pivot if you don’t expect nefarious arrays.

Optimize the implementation of **Select** (with the fancy pivot). Can you beat MergeSort?

Siggi the Studious Stork
What have we learned?

Pending the Lemma

• It is possible to solve SELECT in time $O(n)$.
  • Divide and conquer!

• If you expect that a bad guy will be picking the list, 
  choose a pivot cleverly.
  • More divide and conquer!

• If you don’t expect that a bad guy will be picking the list, in practice it’s better just to pick a random pivot.
  • (Or, if the bad guy picks the list before you pick your pivots...why?).
If time, back to the Lemma

• **Lemma**: If $L$ and $R$ are as in the algorithm SELECT given above, then

$$|L| \leq \frac{7n}{10} + 5$$

and

$$|R| \leq \frac{7n}{10} + 5$$

• We will see a proof by picture.
• See CLRS for proof by proof.
Say these are our $m = \lceil n/5 \rceil$ sub-arrays of size at most 5.
Proof by picture

In our head, let’s sort them.
Then find medians.
Then let’s sort them by the median
The median of the medians is 7. That’s our pivot!
Proof by picture

How many elements are SMALLER than the pivot?
Proof by picture

At least these ones: everything above and to the left.
Proof by picture

How many of those are there?

at least \(3 \cdot \left(\left\lfloor \frac{m}{2} \right\rfloor - 2\right)\)

\[3 \cdot \left(\left\lfloor \frac{m}{2} \right\rfloor - 1\right)\] of these, but then one of them could have been the “leftovers” group.
So how many are LARGER than the pivot? At most

\[ n - 1 - 3 \left( \left\lfloor \frac{m}{2} \right\rfloor - 2 \right) \leq \frac{7n}{10} + 5 \]

(derivation on board)
That was one part of the lemma

- **Lemma**: If $L$ and $R$ are as in the algorithm SELECT given above, then

  $$|L| \leq \frac{7n}{10} + 5$$

  and

  $$|R| \leq \frac{7n}{10} + 5$$

The other part is exactly the same.
The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise

2. The **SELECT** problem.

3. The **SELECT** solution.
   a) The outline of the algorithm.
   b) How to pick the pivot.

4. Return of the **Substitution Method**.
Recap

• The substitution method is another way to solve recurrence relations.
  • Can work when the master theorem doesn’t!

• One place we needed it was for SELECT.
  • Which we can do in time $O(n)$!
Next time

• Randomized algorithms and QuickSort!

BEFORE next time

• Pre-Lecture Exercise 5
  • Remember *probability theory*?
  • The pre-lecture exercise will jog your memory.