Lecture 3
Recurrence Relations and finding the median!
Last week

• We saw two naïve $O(n^2)$-time algorithms:
  
  `GradeSchoolMultiplication` and `InsertionSort`

• We saw two fast divide-and-conquer algorithms:
  
  `Karatsuba` and `MergeSort`

• Analyzed correctness of iterative + recursive algs
  
  • Using induction

• Analyzed running time of recursive algorithms
  
  • By writing out a tree and adding up all the work done
  • (Aka more induction...)
Today

• Recurrence Relations!
  • Two tricks to solve recurrence really fast: Master Method and Substitution Method

• A new algorithm for finding the median!
  • We’ll combine both tricks to analyze the run time 😊
Karatsuba’s Algorithm

**Multiply**($x, y$):
- If $n=1$:
  - Return $xy$
- Write $x = a \times 10^{n/2} + b$
- Write $y = c \times 10^{n/2} + d$
- Recursively compute $ac, bd, (a + b)(c + d)$:
  - $ac = \text{Multiply}(a, c)$, etc...
- Add them up to get $xy$:
  - $xy = ac \times 10^n + (ad + bc) \times 10^{n/2} + bd$

**Running time:**
$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$
Karatsuba: total running time

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>n/2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>n/4</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>$3^t$</td>
<td>n/2^t</td>
<td>[on white board]</td>
</tr>
</tbody>
</table>

(Size 1)

$(\log(n))$ $3^{\log(n)}$ 1
Karatsuba’s Algorithm

**Multiply**($x, y$):

- **If** $n=1$:
  - *Return* $xy$
- **Write** $x = a \ 10^{\frac{n}{2}} + b$
- **Write** $y = c \ 10^{\frac{n}{2}} + d$
- **Recursively compute** $ac, bd, (a + b)(c + d)$:
  - $ac = \text{Multiply}(a, c)$, etc...
- **Add them up to get** $xy$:
  - $xy = ac \ 10^n + (ad + bc) \ 10^{n/2} + bd$

**Running time:**

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^{\log_3 3})$$
MergeSort Algorithm

**MERGESORT**(A):

- n ← length(A)
- **if** n ≤ 1:
  - return A
- L ← **MERGESORT**(A[0 : n/2])
- R ← **MERGESORT**(A[n/2 : n])
- return **MERGE**(L, R)
MergeSort Algorithm

MERGESORT(A):
• \( n \leftarrow \text{length}(A) \)
• \( \text{if } n \leq 1: \)
  • \( \text{return } A \)
• \( L \leftarrow \text{MERGESORT}(A[0 : n/2]) \)
• \( R \leftarrow \text{MERGESORT}(A[n/2 : n]) \)
• \( \text{return } \text{MERGE}(L,R) \)

Running time:
\[
T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)
\]
## MergeSort: total running time

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>n/2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>n/4</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>$2^t$</td>
<td>n/2&lt;sup&gt;t&lt;/sup&gt;</td>
<td>[on white board]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>log(n)</td>
<td>n</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
MergeSort Algorithm

\textbf{MERGESORT}(A):
\begin{itemize}
  \item n \leftarrow \text{length}(A)
  \item \textbf{if} n \leq 1:
    \begin{itemize}
      \item \textbf{return} A
    \end{itemize}
  \item L \leftarrow \text{MERGESORT}(A[0 : n/2])
  \item R \leftarrow \text{MERGESORT}(A[n/2 : n])
  \item \textbf{return} \text{MERGE}(L,R)
\end{itemize}

Running time:
\[
T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)
\]
\[
= \Theta(n \log n)
\]
Mystery Algorithm

- **Mystery**(A):
  - If len(A) \(\leq 1\):
    - TBD: \(O(1)\) work
  - **Recursion:**
    - TBD: 9 calls of size \(n/10\)
  - **Combine sub-problems:**
    - TBD: \(\Theta(n)\) work

Running time:

\[ T(n) = 9T\left(\frac{n}{10}\right) + \Theta(n) \]
### Mystery: total running time

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>n/10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>81</td>
<td>n/100</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>9^t</td>
<td>n/10^t</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>log_{10} n</td>
<td>&lt;n</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Mystery Algorithm

• **Mystery**\( (A) \):
  • If \( \text{len}(A) \leq 1 \):
    • TBD: \( O(1) \) work

• **Recursion**:
  • TBD: 9 calls of size \( n/10 \)

• **Combine sub-problems**:
  • TBD: \( \Theta(n) \) work

Running time:

\[
T(n) = 9T\left(\frac{n}{10}\right) + \Theta(n) = \Theta(n)
\]
Summary

• Karatsuba integer multiplication
  • $T(n) = 3T(n/2) + \Theta(n)$
  • $T(n) = \Theta(n^{\log_2(3)} \approx n^{1.6})$

• MergeSort
  • $T(n) = 2T(n/2) + \Theta(n)$
  • $T(n) = \Theta(n \log(n))$

• Mystery
  • $T(n) = 9T(n/10) + \Theta(n)$
  • $T(n) = \Theta(n)$
The master theorem

• A formula that solves recurrences when all of the sub-problems are the same size.

• Lot’s of notation coming! Remember the bottom line: there are only 3 cases when summing geometric sequences.

A useful formula it is. Know why it works you should.
Summary

• Karatsuba integer multiplication
  • $T(n) = 3T(n/2) + \Theta(n)$
  • $T(n) = \Theta(n^{\log_2(3)} \approx n^{1.6})$

• MergeSort
  • $T(n) = 2T(n/2) + \Theta(n)$
  • $T(n) = \Theta(n \log(n))$

• Mystery
  • $T(n) = 9T(n/10) + \Theta(n)$
  • $T(n) = \Theta(n)$
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.

We can also take $n/b$ to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$ and the theorem is still true.
Technicalities

Integer division

• If n is odd, I can’t break it up into two problems of size n/2.

\[ T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n) \]

• However (see CLRS, Section 4.6.2), one can show that the Master theorem works fine if you pretend that what you have is:

\[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \]

• From now on we’ll mostly ignore floors and ceilings in recurrence relations.
Examples

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + O(n^d). \]

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

- **Karatsuba integer multiplication**
  - \( T(n) = 3T(n/2) + O(n) \)
  - \( T(n) = O(n^{\log_2(3)} \approx n^{1.6}) \)
  - \( a = 3 \)
  - \( b = 2 \)
  - \( d = 1 \)
  - \( a > b^d \)

- **MergeSort**
  - \( T(n) = 2T(n/2) + O(n) \)
  - \( T(n) = O(n \log(n)) \)
  - \( a = 2 \)
  - \( b = 2 \)
  - \( d = 1 \)
  - \( a = b^d \)

- **Mystery algorithm**
  - \( T(n) = 9T(n/10) + O(n) \)
  - \( T(n) = O(n) \)
  - \( a = 9 \)
  - \( b = 10 \)
  - \( d = 1 \)
  - \( a < b^d \)
Proof of the master theorem

• We’ll do the same recursion tree thing we did for **MergeSort**, but be more careful.

• Suppose that \( T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \).

Hang on! The hypothesis of the Master Theorem was the extra work at each level was \( O(n^d) \). That’s NOT the same as \( work \leq cn^d \) for some constant \( c \).

That’s true ... we’ll actually prove a weaker statement that uses this hypothesis instead of the hypothesis that \( T(n) = a \cdot T \left( \frac{n}{b} \right) + O(n^d) \).

It’s a good exercise to make this proof work rigorously with the \( O() \) notation.
Recursion tree

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td>c \cdot n^d</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>n/b</td>
<td>ac \left( \frac{n}{b} \right)^d</td>
</tr>
<tr>
<td>2</td>
<td>a^2</td>
<td>n/b^2</td>
<td>a^2c \left( \frac{n}{b^2} \right)^d</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>a^t</td>
<td>n/b^t</td>
<td>a^tc \left( \frac{n}{b^t} \right)^d</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

(Size 1)

\[ \log_b(n) = a^{\log_b(n)} \]

\[ a^{\log_b(n)} \cdot c \]
Recursion tree

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td>( c \cdot n^d )</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>n/b</td>
<td>( ac \left( \frac{n}{b} \right)^d )</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
</tbody>
</table>

Total work (derivation on board) is at most:

\[
c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t
\]

\[
= c \cdot n^d \cdot a^{\log_b(n)} \cdot \frac{1}{1 - \frac{a}{b^d}}
\]

(Size 1)
Now let’s check all the cases (on board)

\[ T(n) = \begin{cases} 
O(n^d \log(n)) & \text{if } a = b^d \\
O(n^d) & \text{if } a < b^d \\
O(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases} \]
Even more generally, for $T(n) = aT(n/b) + f(n)$...

**Theorem 3.2** (Master Theorem). Let $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ be a recurrence where $a \geq 1$, $b > 1$. Then,

- If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, $T(n) = \Theta\left(n^{\log_b a}\right)$.
- If $f(n) = \Theta\left(n^{\log_b a}\right)$, $T(n) = \Theta\left(n^{\log_b a \log n}\right)$.
- If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

Figure out how to adapt the proof we gave to prove this more general version!

[From CLRS]

Ollie the Over-Achieving Ostrich
LIFE IS SO GOOD

BUT I WILL NOT BECOME COMPLACENT AND I CAN TOTALLY SOLVE RECURRENCES FROM SCRATCH IF I WANT TO
Today

• Recurrence Relations!
  • Two tricks to solve recurrence really fast: Master Method and Substitution Method

• A new algorithm for finding the median!
  • We’ll combine both tricks to analyze the run time 😊
MergeSort Algorithm

\text{MERGESORT}(A):
\begin{itemize}
  \item \( n \leftarrow \text{length}(A) \)
  \item \textbf{if} \( n \leq 1 \):
    \begin{itemize}
      \item \textbf{return} \( A \)
    \end{itemize}
  \item \( L \leftarrow \text{MERGESORT}(A[0 : n/2]) \)
  \item \( R \leftarrow \text{MERGESORT}(A[n/2 : n]) \)
  \item \textbf{return} \( \text{MERGE}(L,R) \)
\end{itemize}

Running time:
\[ T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \]
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.
• $T(n) = 2 \cdot T \left( \frac{n}{2} \right) + 1 \cdot n$

• $T(n) = 2 \cdot \left( 2 \cdot T \left( \frac{n}{4} \right) + \frac{n}{2} \right) + 1 \cdot n$

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

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

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

1. Guess what the answer is.

29
• $T(n) = 2 \cdot T \left( \frac{n}{2} \right) + 1 \cdot n$
  $= \log(2)$

• $T(n) = 2 \cdot \left( 2 \cdot T \left( \frac{n}{4} \right) + \frac{n}{2} \right) + 1 \cdot n$

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

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

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

• Following the pattern...

• $T(n) \leq n \cdot T(1) + \log(n) \cdot n = n(\log(n) + 1)$

So that is our guess!
• 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:
  • \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \)
    \[
    \leq 2 \left( \frac{n}{2} \left( \log\left(\frac{n}{2}\right) + 1 \right) \right) + n
    \]
    \[
    = 2 \left( \frac{n}{2} \left( \log(n) - 1 + 1 \right) \right) + n
    \]
    \[
    = 2 \left( \frac{n}{2} \log(n) \right) + n
    \]
    \[
    = n(\log(n) + 1)
    \]

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

• Recurrence Relations!
  • Two tricks to solve recurrence really fast: Master Method and Substitution Method

• A new algorithm for finding the median!
  • We’ll combine both tricks to analyze the run time 😊
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) = \text{MIN}(A)
- **SELECT** (A, n/2) = \text{MEDIAN}(A)
- **SELECT** (A, n) = \text{MAX}(A)

For today, assume all arrays have distinct elements.
Use our sorting algorithms!

- **NaïveSelect**(A, k):
  - A = **MergeSort**(A)
  - **return** A[k-1]

- Running time is **O(n log(n))**.
- So that’s the benchmark....

Can we do better?

We’re hoping to get **O(n)**
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: 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”

$L =$ array with things smaller than $A[pivot]$

$R =$ array with things larger than $A[pivot]$

How about this pivot?

This PARTITION step takes time $O(n)$. (Notice that we don’t sort each half).
Idea continued...

Say we want to find \textsc{select}(A, k)

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

This suggests a recursive algorithm

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

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

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

**Base Case**: If the \(\text{len}(A) = O(1)\), 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
Select works!

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

Formally prove the correctness of \texttt{Select}!
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)

Running time:

\[
T(n) \leq \max\{T(|L|), T(|R|)\} + O(n)
\]
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...
  • What do we hope happens?
  • What do we hope doesn’t happen?

Think-Pair-Share!
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)

Running time:

\[ T(n) \leq \max\{T(|L|), T(|R|)\} + O(n) \]
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

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

*Okay, really ideal would be that we always pick the pivot so that \( \text{len}(L) = k-1 \). But say we don’t have control over \( k \), just over how we pick the pivot.
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 $\text{Select}(A, n/2)$
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!

Apply here, the Master Theorem STILL does NOT. (Since we don’t know that we can do this – and if we could how long would it take?).

But at least it gives us a goal!

Jedi master Yoda

Lucky the lackadaisical lemur
Goal

• Pick the pivot so that

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

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

• We can’t solve \( \text{Select}(A, n/2) \) (yet)
• But we can **divide and conquer** and solve \( \text{Select}(B, m/2) \) for smaller values of \( m \) (where \( \text{len}(B) = m \)).
• **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 $m = \left\lceil \frac{n}{5} \right\rceil$ groups, of size $\leq 5$ each.
  - For $i=1, \ldots, m$:
    - Find the median within the $i$'th group, call it $p_i$
    - $p = \text{SELECT}( [p_1, p_2, p_3, \ldots, p_m], \frac{m}{2} )$
  - return $p$

This takes time $O(1)$, for each group, since each group has size 5. So that’s $O(m)=O(n)$ total in the for loop.

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

PARTITION around that 6:

This part is L

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

• Formally, we will prove (later... or at home):

\textbf{Lemma:} If we choose the pivots like this, then

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

and

\[ |R| \leq \frac{7n}{10} + 5 \]
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:
  $$T(n) \leq ?$$
Today

• Recurrence Relations!
  • Two tricks to solve recurrence really fast: Master Method and Substitution Method

• A new algorithm for finding the median!
  • We’ll combine both tricks to analyze the run time 😊
Pseudocode

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

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)

Running time:

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

Oh no! I told you \(\Omega(n)\) times: only subproblems of same size!

Why can't we just add them up?! It's definitely not a proof, but it's also not a bad guess for substitution method…
This sounds like a job for...

**The Substitution Method!**

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

- **Guess (inductive hypothesis):**
  - \( T(k) \leq 10ck \) for all \( 1 \leq k \leq n \)

- **Base case:**
  - \( T(1) \leq c \)

- **Inductive step:**
  - \( T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn \)
    \[ \leq (10c \cdot \frac{n}{5}) + (10c \cdot \frac{7n}{10}) + cn \]
    \[ = 10cn \]

**Conclusion:**
\[ T(n) = O(n) \]
Recap

• Shortcuts for solving recurrences:
  • Master Method – when sub-problems are same size
  • Substitution Method – requires a good guess

• Select(A,k) algorithm
  • Finds the k-th element in an *unsorted* array
  • Divide-and-Conquer (median of sub-medians)
  • $O(n)$ time  --- faster than $O(n\log(n))$ with sorting!
Next time

• Randomized algorithms!
• Turns out it’s an easier way of doing almost the same things 😊
  • And often also faster in practice!

Before next time

• Go to section!
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.
Proof by picture

Say these are our $m = \lceil n/5 \rceil$ sub-arrays of size at most 5.
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

We will show that lots of elements are smaller than the pivot, hence not too many are larger than the pivot.

How many elements are SMALLER than the pivot?

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

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.

How many of those are there?

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

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)

Remember \( m = \left\lfloor \frac{n}{5} \right\rfloor \)
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.