Last Time

- Heaps
Homework 4 stats

- Mean: 93.92
- Stdev: 4.40
- Median: 95
Homework problem 1d.

- Solve the recurrence using the Master Theorem
- \( T(n) = 2T(\sqrt{n}) + n \)
Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the non negative integers by the recurrence $T(n) = aT(n/b) + f(n)$. Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

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

Review of the Master Theorem
Homework problem 1d.

- Solve the recurrence using the Master Theorem
- \( T(n) = 2T(\sqrt{n}) + n \)
- Step 1: identify \( a, b, \) and \( f(n) \)
Solve the recurrence using the Master Theorem

\[ T(n) = 2T(\sqrt{n}) + n \]

Step 1: identify \( a, b, \) and \( f(n) \)

- \( b \) is not a constant \( > 1 \).
- The master theorem only applies to functions of the form:

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]
Solve the recurrence using the Master Theorem

- $T(n) = 2T(\sqrt{n}) + n$
- Step 1: identify $a$, $b$, and $f(n)$
- $b$ is not a constant $> 1$.
- The master theorem only applies to functions of the form:
- $T(n) = aT(\frac{n}{b}) + f(n)$
- Manipulate the form of the function.
Homework problem 1d.

- \( T(n) = 2T(\sqrt{n}) + n \)
- \( T(n) = 2T(n^{1/2}) + n \)
- Introduce a new variable, \( m = \log n \), \( n = 2^m \)
Homework problem 1d.

- \( T(n) = 2T(\sqrt{n}) + n \)
- \( T(n) = 2T(n^{1/2}) + n \)
- Introduce a new variable, \( m = \log n \), \( n = 2^m \)
- \( T(2^m) = 2T(2^{m^{1/2}}) + 2^m \)
- \( T(2^m) = 2T(2^{m/2}) + 2^m \)
Homework problem 1d.

- $T(n) = 2T(\sqrt{n}) + n$
- $T(n) = 2T(n^{1/2}) + n$
- Introduce a new variable, $m = \log n$, $n = 2^m$
- $T(2^m) = 2T(2^{m^{1/2}}) + 2^m$
- $T(2^m) = 2T(2^{m/2}) + 2^m$
- Introduce a new function $S(m) = T(2^m)$
- $S(m) = 2S(m/2) + 2^m$
Homework problem 1d.

- $T(n) = 2T(n^{1/2}) + n$
- $S(m) = 2S(m/2) + 2^m$
- Now the Master Theorem can be applied to $S(m)$
- $a = 2$, $b = 2$, $f(m) = 2^m$
- Compare $2^m$ to $m^{\log_a b}$
Homework problem 1d.

- \( T(n) = 2T(n^{1/2}) + n \)
- \( S(m) = 2S(m/2) + 2^m \)
- Now the Master Theorem can be applied to \( S(m) \)
- \( a = 2, \ b = 2, \ f(m) = 2^m \)
- Compare \( 2^m \) to \( m^{\log_a b} \)
- \( f(m) = 2^m = \Omega(m^{\log_a b}) = \Omega(m^{\log_2 2}) = \Omega(m^1) = \Omega(m) \)
- Case 3) of the Master Theorem.
- \( S(m) = \Theta(2^m) \)
Homework problem 1d.

- \( T(n) = 2T(n^{1/2}) + n \)
- \( S(m) = 2S(m/2) + 2^m \)
- Now the Master Theorem can be applied to \( S(m) \)
- \( a = 2, \ b = 2, \ f(m) = 2^m \)
- Compare \( 2^m \) to \( m^{\log_a b} \)
- \( f(m) = 2^m = \Omega(m^{\log_a b}) = \Omega(m^{\log_2 2}) = \Omega(m^1) = \Omega(m) \)
- Case 3) of the Master Theorem.
- \( S(m) = \Theta(2^m) \)
- \( T(n) = T(2^m) = S(m) = \Theta(2^m) = \Theta(2^{\log n}) = \Theta(n) \)
Heaps

- Heap Review
  - Equivalence between complete trees and arrays.
  - More detail on Heap Insert and Heap Increase Key
Representing a Heap as an Array

- The **Max heap property** allows compact representation of a heap as an array.
- \( \text{Parent}(i) = \lfloor i/2 \rfloor \)
- \( \text{Left}(i) = 2i \)
- \( \text{Right}(i) = 2i + 1 \)
Heap Example

A[i] 16 14 10 8 7 9 3 2 4 1

i 1 2 3 4 5 6 7 8 9 10
The **Max heap property** allows compact representation of a heap as an array.

- \( \text{PARENT}(i) = \lfloor i/2 \rfloor \)
- \( \text{LEFT}(i) = 2i \)
- \( \text{RIGHT}(i) = 2i + 1 \)
$2^{d-1}$ is the maximum number of nodes in a binary tree at a given depth $d$. 
\begin{itemize}
  \item $2^{d-1}$ is the maximum number of nodes in a binary tree at a given depth $d$.
  \item **Base case** $d = 1$ A tree of depth 1 has only a root. $2^0 = 1$.
\end{itemize}
2^{d-1} is the maximum number of nodes in a binary tree at a given depth $d$.

Base case $d = 1$ A tree of depth 1 has only a root. $2^0 = 1$.

Inductive Step The maximum number of nodes at depth $d + 1$ arises when each node at $d$ has the maximum number of children.
2^{d-1} is the maximum number of nodes in a binary tree at a given depth \( d \).

**Base case** \( d = 1 \) A tree of depth 1 has only a root. \( 2^0 = 1 \).

**Inductive Step** The maximum number of nodes at depth \( d + 1 \) arises when each node at \( d \) has the maximum number of children.

The maximum number of children is 2. Thus, the maximum number of nodes at depth \( d + 1 \) is double the maximum number at depth \( d \).

The maximum at depth \( d \) is \( 2^{d-1} \).

The maximum at depth \( d + 1 \) is \( 2 \times 2^{d-1} = 2^{(d+1)-1} (= 2^d) \)
The maximum number of nodes in a tree of depth $d$ is $2^d - 1$.
The maximum number of nodes in a tree of depth $d$ is $2^d - 1$

**Base Case** A tree with depth $d = 1$ only has one node, a root. $2^d - 1 = 2^1 - 1 = 2 - 1 = 1$. 
The maximum number of nodes in a tree of depth $d$ is $2^d - 1$.

**Base Case** A tree with depth $d = 1$ only has one node, a root. $2^d - 1 = 2^1 - 1 = 2 - 1 = 1$.

**Inductive Step** The maximum number of nodes in a tree of depth $d + 1$ is the number of nodes in a tree of depth $d$ plus the maximum number of nodes at depth $d + 1$.

Max nodes in a tree of depth $d$ is $2^d - 1$

Max nodes at depth $d + 1$ is $2^{(d+1)-1} = 2^d$

$2^d + 2^d - 1 = 2 \times 2^d - 1 = 2^{d+1} - 1$
The nodes at any complete depth \( d \) can be uniquely indexed with the values between 1 and \( 2^d - 1 \).

**Base Case** When \( d = 1 \) the tree has only a root. Thus the root is uniquely indexed by 1. \( 2^{1-1} = 2^0 = 1 \) and \( 2^1 - 1 = 2 - 1 = 1 \)
The nodes at any complete depth $d$ can be uniquely indexed with the values between 1 and $2^d - 1$.

**Base Case** When $d = 1$ the tree has only a root. Thus the root is uniquely indexed by 1. $2^{1-1} = 2^0 = 1$ and $2^1 - 1 = 2 - 1 = 1$.

**Inductive Step** Assume all nodes at depths $d$ or below are uniquely indexed between 1 and $2^d - 1$.

The $d + 1$ depth of the tree contains $2^{(d+1)-1} = 2^d$ nodes.

The range between $2^{(d+1)-1}$ and $2^{(d+1)} - 1$ contains $2^d$ indices.

$$2^{d+1} - 1 - 2^{(d+1)-1} + 1 = 2^{d+1} - 2^d = 2 \times 2^d - 2^d = 2^d$$

Therefore the range between $2^d$ and $2^{d+1} - 1$ can contain enough elements for depth $d + 1$ and does not overlap with the elements at lower depths.
Representing a Heap as an array

- Assuming we have a complete graph with \( N \) nodes, can we arrange the elements compactly in an \( N \) element array using the following relationships?
  - \( \text{Parent}(i) = \lfloor i/2 \rfloor \)
  - \( \text{Left}(i) = 2i \)
  - \( \text{Right}(i) = 2i + 1 \)

- Does \( j \) correspond to a node in the tree?
- Does \( j \) correspond to a unique node in the tree?
Does an index $j$ in the range $1$ and $2^d - 1$ correspond to a node in a complete tree? Assume not. If there exists an index $j$ that does not correspond to a node. Therefore $j$ is not $\text{LEFT}(i)$ nor $\text{RIGHT}(i)$ for all $1 \leq i \leq 2^d - 1$.

If $j = 1$ then $j$ corresponds to the root.

Otherwise, assume without loss of generality that there exists $i$ such that $2i \leq j$ and $i + 1$ such that $2(i + 1) \geq j$.

Therefore $2i \leq j \leq 2(i + 1)$.

Thus, $j$ can be $2i$ in which case it is $\text{LEFT}(i)$

Thus, $j$ can be $2i + 1$ in which case it is $\text{RIGHT}(i)$

Thus, $j$ can be $2i + 2 = 2(i + 1)$ in which case it is $\text{LEFT}(i+1)$
Does an index \( j \) in the range 1 and \( 2^d - 1 \) correspond to a unique node in a complete tree?

Assume not. If there exists an index \( j \) that corresponds to two nodes.

\( j \) must be the child of two different nodes, \( i \) and \( i' \) where \( i \neq i' \).

Both Left children. \( 2i \neq 2i' \) if \( i \neq i' \).

Both Right children. \( 2i + 1 \neq 2i' + 1 \) if \( i \neq i' \).

Note \( 2i \) is even and \( 2i + 1 \) is odd.

Therefore \( 2i \neq 2i' + 1 \) for any integers \( i \) and \( i' \).
Heap Increase Key

**HeapIncreaseKey**(*A*, *i*, *key*)

\[A[i] \leftarrow key\]

while \(i > 1\) and \(A[\text{Parent}(i)] < A[i]\) do

\[
\text{swap } A[i] \leftrightarrow A[\text{Parent}(i)]
\]

\[
i \leftarrow \text{Parent}(i)
\]

end while

- \(O(\log n)\) - Heap traversal
Max Heap Insert

\[
\text{MaxHeapInsert}(A, \text{key})
\]

\[
\begin{align*}
  \text{size}(A) & \leftarrow \text{size}(A) + 1 \\
  A[\text{size}(A)] & \leftarrow -\infty \\
  \text{HeapIncreaseKey}(A, \text{size}(A), \text{key})
\end{align*}
\]

- \( O(\log n) \) - \( \text{HeapIncreaseKey}(A, i, \text{key}) \)
Bye

- Next time (10/8)
  - Balanced Binary Search Trees
- For Next Class
  - Homework 5 Due
  - Read 13.1, 13.2, 13.3, 13.4