Algorithms, Design & Analysis

Lecture 08: **Secretary Problem And Heap Sort**

Abuhurairah Faheem And Muhammad Talha

Information Technology University

June 19, 2025

About Your Fellows

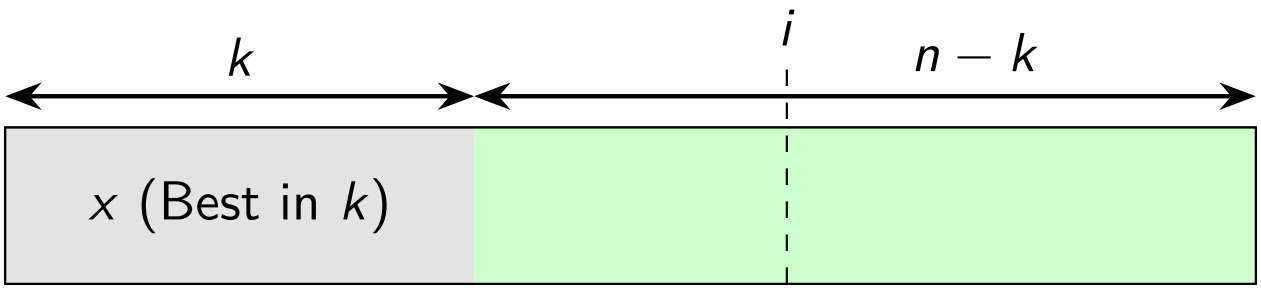
- Hi there! We are **Abuhurairah** and **Talha**.
- We are Associate Students at ITU.

Secretary Problem

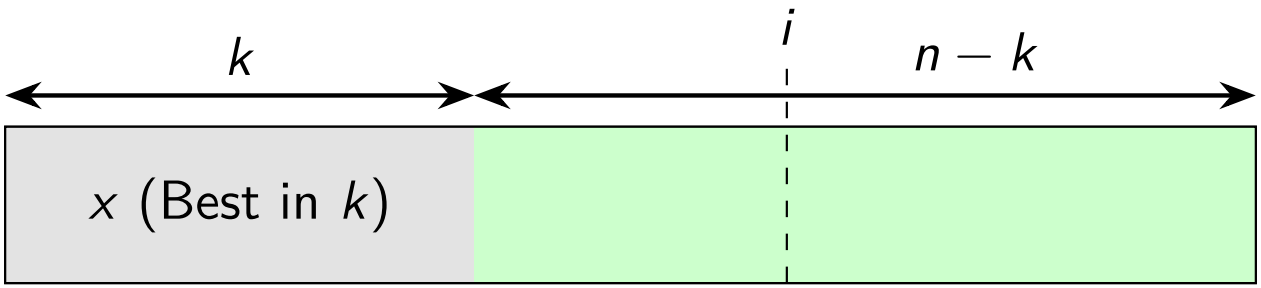
Goal:

- Select the best secretary (or candidate) out of n applicants.

Secretary Problem Diagram



Secretary Problem Diagram



Dashed line marks candidate i:
the first candidate in the decision phase
whose score exceeds x.

Recap: **Secretary Problem**

- **Secretary Problem**: Select the best candidate out of n applicants.
- Strategy:
  - Interview the first k candidates (skip them).
  - Let x be the best candidate among the first k.
  - For any candidate i (with $i > k$), if $score(i) > score(x)$ then consider hiring.

Probability of Hiring the Best Candidate

- We wish to compute:

$$P(\text{Hiring best candidate}) = \sum_{i=1}^{n} \Pr(i \text{ is best}) \times \Pr(\text{hire } i \mid i \text{ is best})$$

- Since every candidate is equally likely to be the best:

$$\Pr(i \text{ is best}) = \frac{1}{n}.$$

Conditional Probability

- The probability of hiring candidate i given that i is best is:

$$\Pr(\text{hire } i \mid i \text{ is best}) = \Pr\left(\text{no candidate before } i \text{ is hired}\right)$$

- This probability is modeled as:

$$\frac{k-1}{i-1}.$$

- Thus, we have:

$$P(\text{Hiring best candidate}) = \frac{k-1}{n} \sum_{i=k+1}^{n} \frac{1}{i-1}.$$

Harmonic Series Approximation

- Recognize that:

$$\sum_{i=k+1}^{n} \frac{1}{i-1} = \sum_{j=k}^{n-1} \frac{1}{j} \quad (\text{with } j = i-1)$$

- The harmonic series $H_m = \sum_{i=1}^{m} \frac{1}{i}$ is approximately:

$$H_m \approx \ln(m) \quad (\text{or } \ln(m+1)).$$

- Therefore:

$$P(\text{Hiring best candidate}) \approx \frac{k-1}{n} \left[\ln(n-1) - \ln(k-1)\right].$$

Optimizing the Success Probability

- Let $j = k - 1$. Then:

$$P(\text{success}) = \frac{j}{n} \left[\ln(n-1) - \ln(j) \right].$$

- To maximize $P(\text{success})$, we differentiate with respect to j and set the derivative to zero.

Derivative Calculation

- Define:
$$u = \frac{j}{n} \quad \text{and} \quad v = \ln(n-1) - \ln(j).$$
- Then:
$$\frac{du}{dj} = \frac{1}{n} \quad \text{and} \quad \frac{dv}{dj} = -\frac{1}{j}.$$
- Applying the product rule:
$$\frac{d}{dj}(uv) = u\frac{dv}{dj} + v\frac{du}{dj} = \frac{j}{n}\left(-\frac{1}{j}\right) + \frac{1}{n}\left[\ln(n-1) - \ln(j)\right].$$
- Simplifying:
$$\frac{d}{dj}(uv) = -\frac{1}{n} + \frac{1}{n}\left[\ln(n-1) - \ln(j)\right].$$

Setting the Derivative to Zero

- Set:
$$\frac{1}{n}\Big[\ln(n-1) - \ln(j) - 1\Big] = 0.$$
- This implies:
$$\ln(n-1) - \ln(j) - 1 = 0 \quad \rightarrow \quad \ln(j) = \ln(n-1) - 1.$$
- Exponentiating both sides:
$$j = \frac{n-1}{e}.$$

Final Result

- Recall that $j = k - 1$. Thus:

$$k = \frac{n-1}{e} + 1.$$

- For large n, this simplifies to:

$$k \approx \frac{n}{e}.$$

- **Optimal Strategy:** Skip the first $\frac{n}{e}$ candidates.

Algorithms, Design & Analysis

Introduction to Heap

What is a Heap?

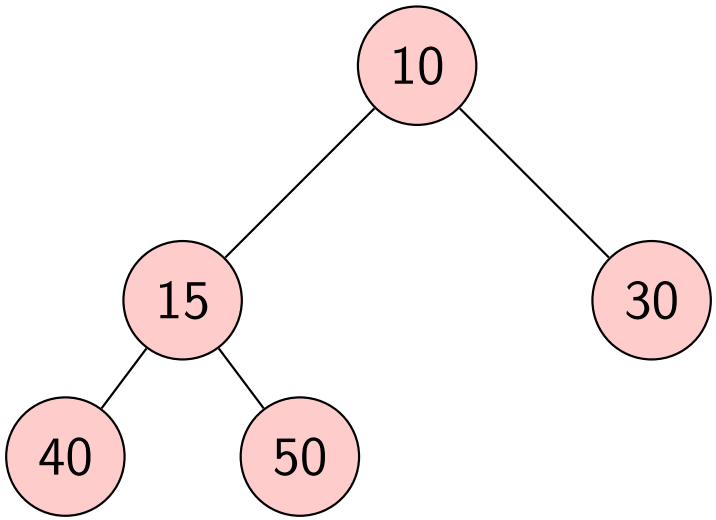
- A complete binary tree.

What is a Complete Binary Tree?

Definition:

- A binary tree where all levels are completely filled, except possibly the last one.
- The last level must be filled from left to right.

Example of a Complete Binary Tree:

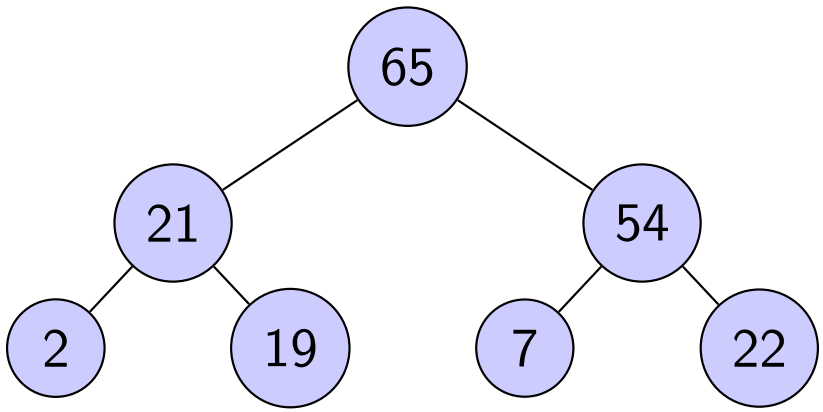


Height of a Complete Binary Tree

- In a complete binary tree with n nodes:
  - The height h is approximately $\log_2 n$.
  - More precisely, $h = \lfloor \log_2 n \rfloor$.

Height of a Complete Binary Tree

- In a complete binary tree with n nodes:
  - The height h is approximately $\log_2 n$.
  - More precisely, $h = \lfloor \log_2 n \rfloor$.



Mathematical Derivation of Height

- A complete binary tree follows the pattern:

$$n = 1 + 2 + 4 + \cdots + 2^h \tag{1}$$

- Using the summation formula for a geometric series:

$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \tag{2}$$

- Solving for height by taking logarithms:

$$h = \lfloor \log_2(n+1) - 1 \rfloor \tag{3}$$

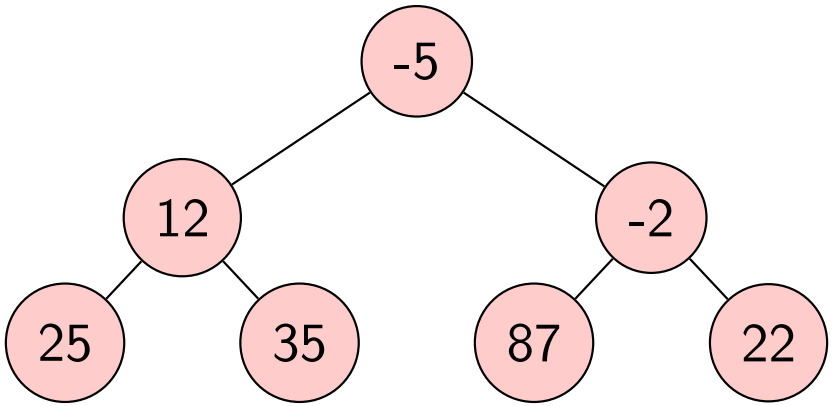
Summation and Node Count

- The number of nodes at level i in a complete binary tree is 2^i.
- The total number of nodes in a complete binary tree:

$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \tag{4}$$

- Rearranging for height:

$$h = \lfloor \log_2 n \rfloor \tag{5}$$



Final Height Formula and Conclusion

- We derived that the height of a complete binary tree is:

$$h = \lfloor \log_2 n \rfloor \tag{6}$$

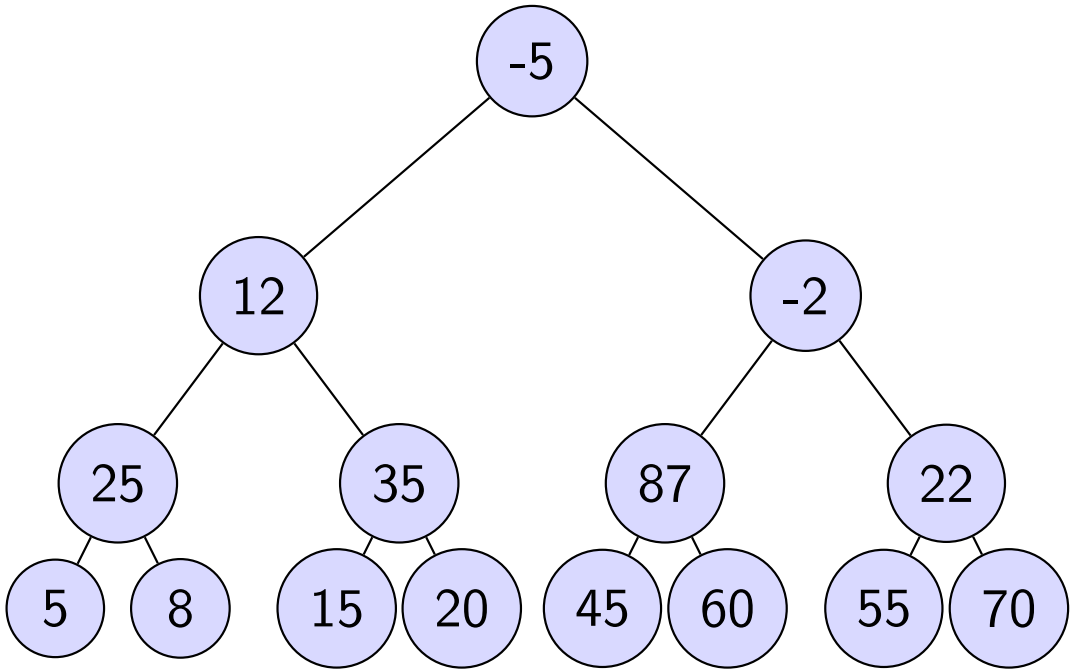
- Since heaps are complete binary trees, their height follows this formula.

Example: Verifying Height Formula

- Consider a complete binary tree with **n = 14** nodes.
- Using our formula:

$$h = \lfloor \log_2 14 \rfloor = 3 \tag{7}$$

- Visualizing the tree:

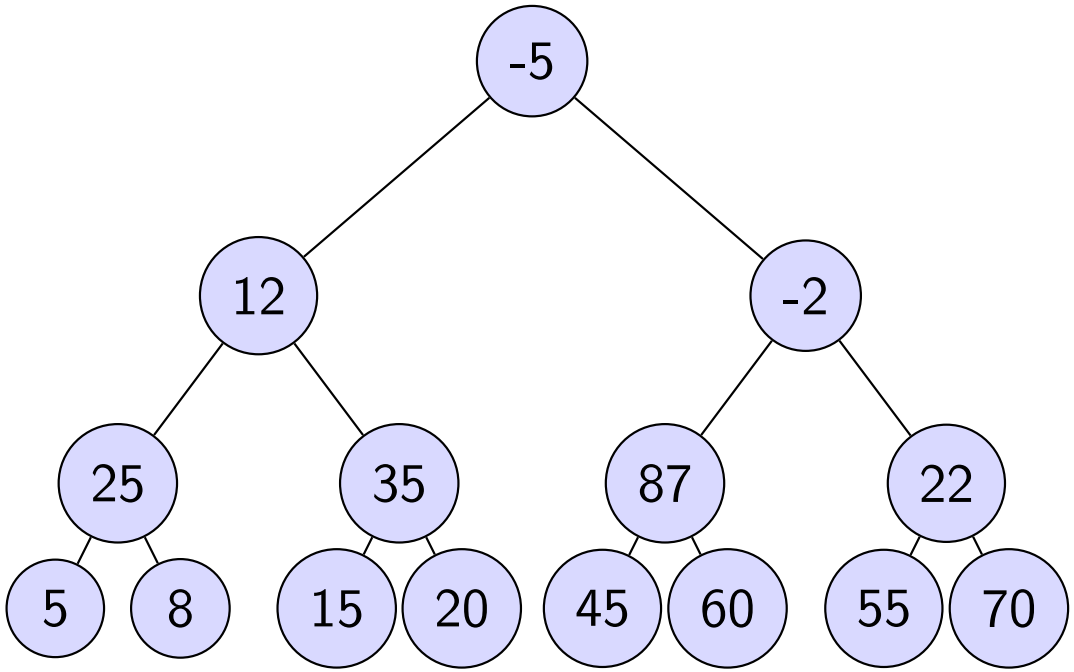


Example: Verifying Height Formula

- Consider a complete binary tree with **n = 14** nodes.
- Using our formula:

$$h = \lfloor \log_2 14 \rfloor = 3 \tag{7}$$

- Visualizing the tree:

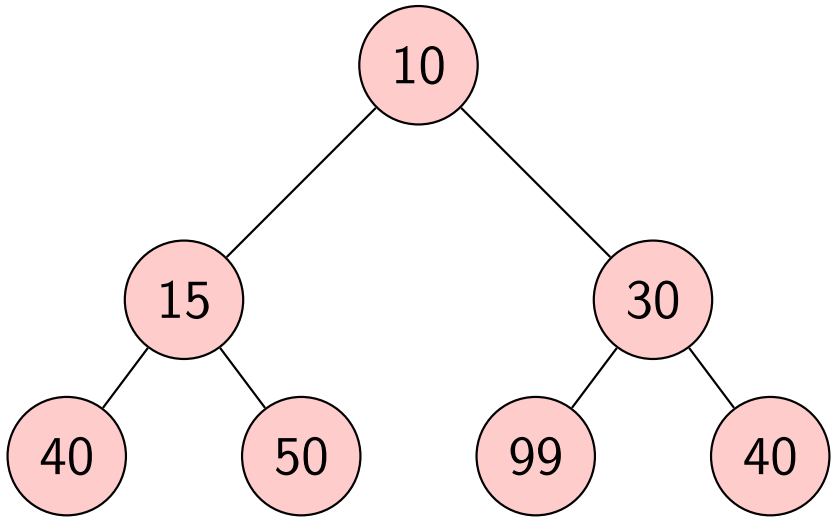


- As observed, the tree has exactly **h = 3** levels, confirming our formula.

What is a Heap?

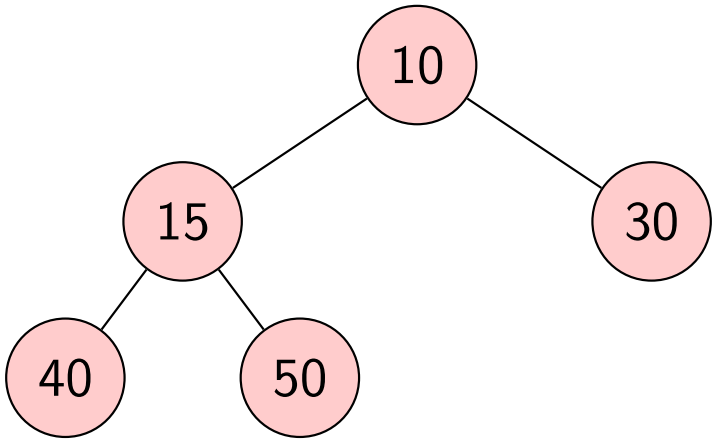
- A complete binary tree.
- Heap property:
  - Max Heap: Parent node $\geq$ children.
  - Min Heap: Parent node $\leq$ children.

Example of a Min Heap:

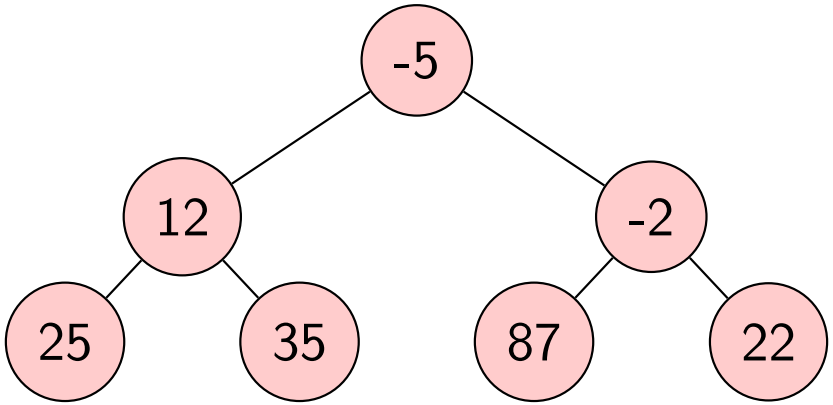


Examples of Min Heap

Example 1:

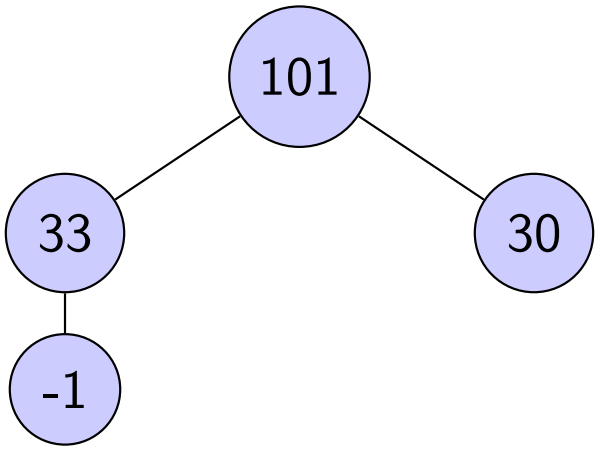


Example 2:

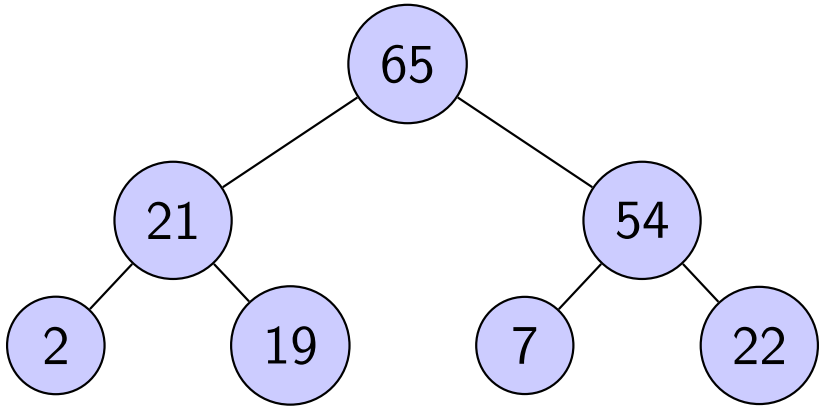


Examples of Max Heap

Example 1:



Example 2:



Accessing a Parent Node in a Heap

Formula:

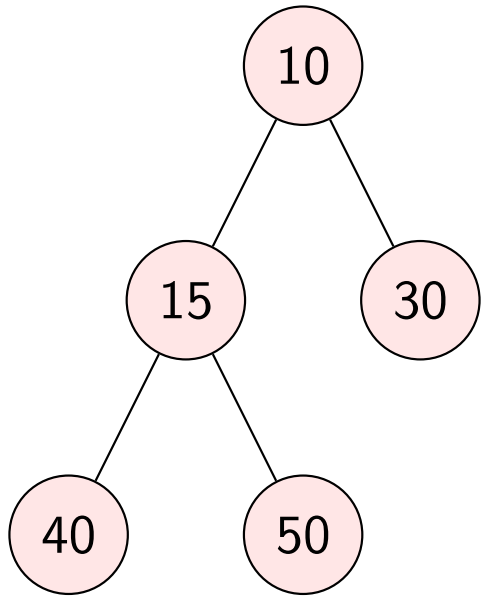
- In a zero-based index heap stored as an array:

$$\text{Parent}(i) = \frac{i-1}{2} \quad \text{(using integer division)}$$

- This means that for any node at index i, its parent is found at index $\lfloor (i-1)/2 \rfloor$.

Accessing a Parent Node in a Heap

Visualization:



Example:

- The node 40 is at the index 3.
- Using the formula: $\lfloor (3-1)/2 \rfloor = 1$, so its parent is at index **1** (value = **15**).

Accessing Child Nodes in a Heap

Formulas:

- In a zero-based index heap stored as an array:

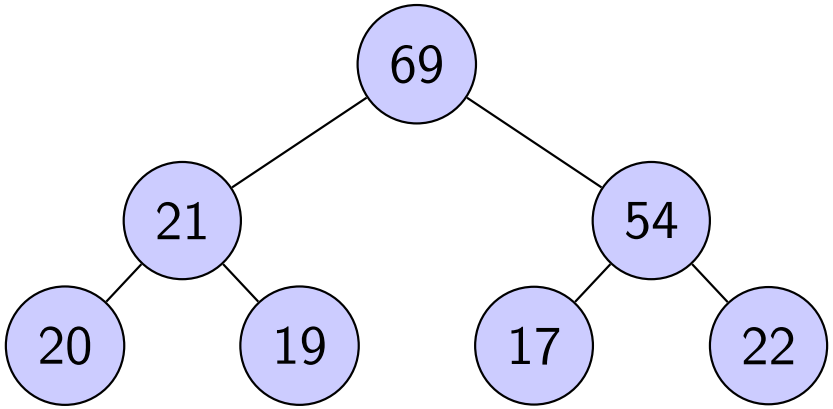
$$\text{Left Child}(i) = 2i + 1$$

$$\text{Right Child}(i) = 2i + 2$$

- This means that for any node at index i:
  - The left child is found at index $2i + 1$.
  - The right child is found at index $2i + 2$.

Accessing Child Nodes in a Heap

Visualization:



Example:

- Node 54 is at index 2.
- Using the formulas:
  - **Left Child**: $2(2) + 1 = 5$ (value = **17**).
  - **Right Child**: $2(2) + 2 = 6$ (value = **22**).

How Do We Build a Heap? (Min Heap)

Steps to Build a Heap

1. Start with an unsorted list of numbers.
2. Turn it into a tree shape.
3. Fix the tree by making sure each parent is smaller than its children.
4. Keep fixing until the smallest number is on top!

Understanding Heapify

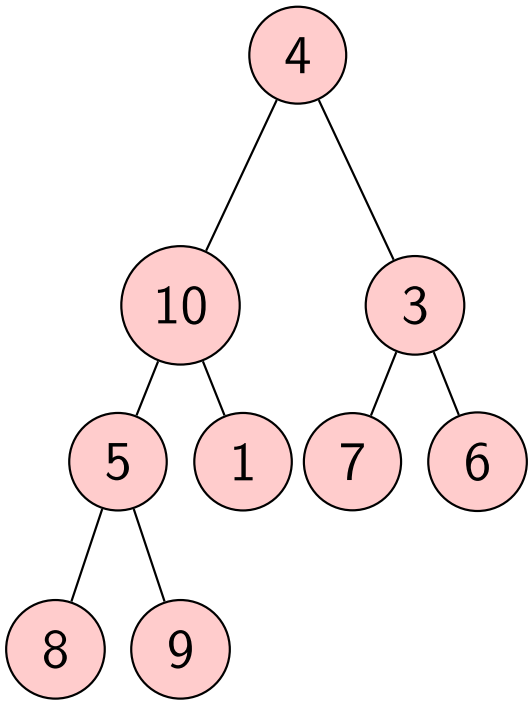
What is Heapify?

- Heapify is like cleaning up a messy room!
- It makes sure each parent follows the heap rule.
- If a child is bigger (in case of min-heap), we swap places.
- We keep fixing from bottom to top.

Let's Build a Min-Heap!

Given Numbers: 4, 10, 3, 5, 1, 7, 6, 8, 9

Step 1: Turn into a Tree



Apply Heapify Step-by-Step

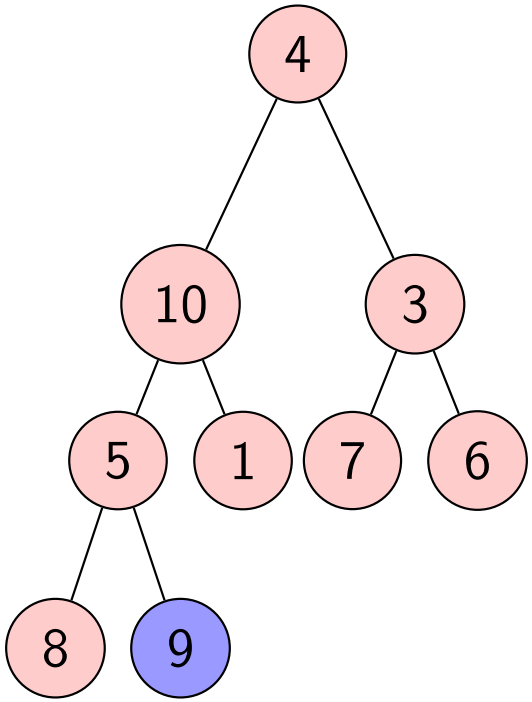
Steps:

1. Start from the last node.
2. Check if it's valid.
3. Swap if needed.
4. Repeat until the top is the smallest.

Step-by-Step Heapify Process

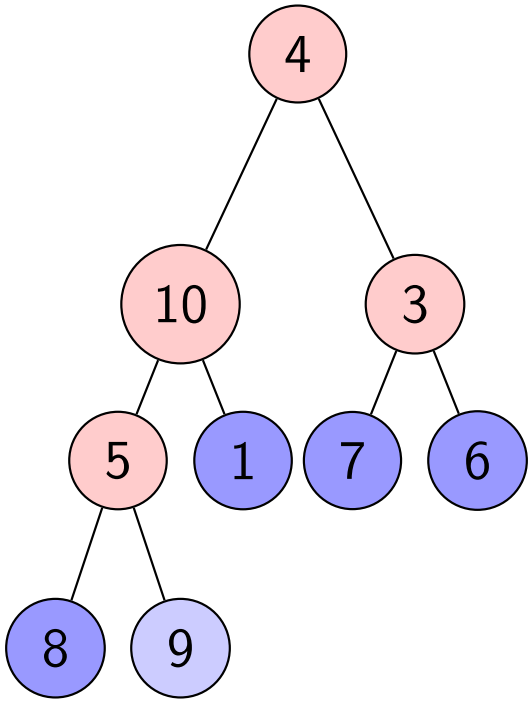
Starting from the last index, checking each node:

Step 1: Check Node 8 (Valid)



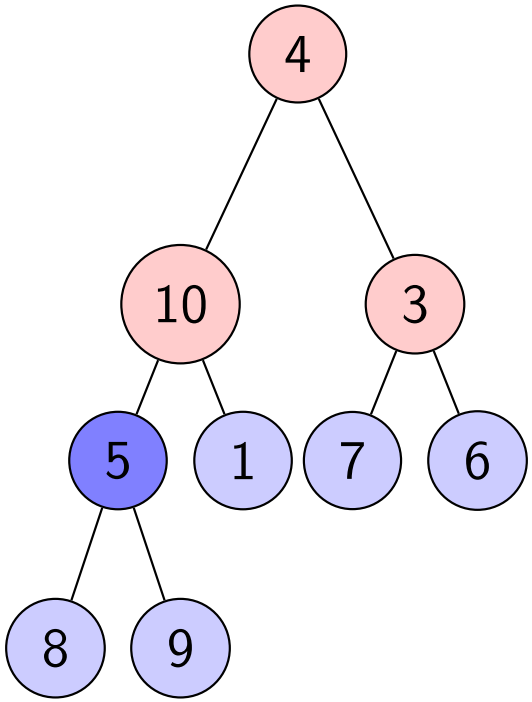
No swaps needed.

Step 2: Check Node 7,6,5,4 (Valid)



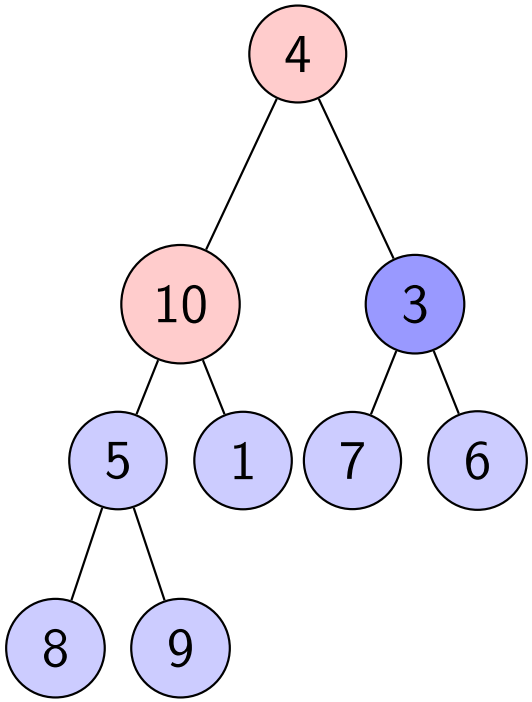
No swaps needed.

Step 3: Check Node 3 (Valid)



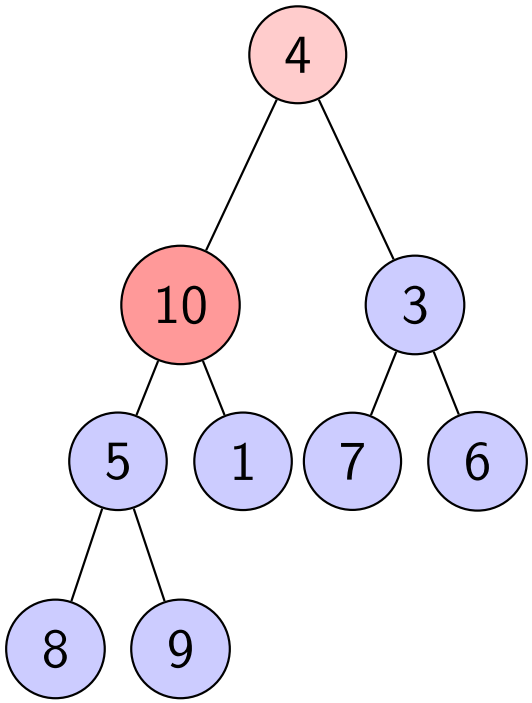
No swaps needed.

Step 4: Check Node 2 (Valid)



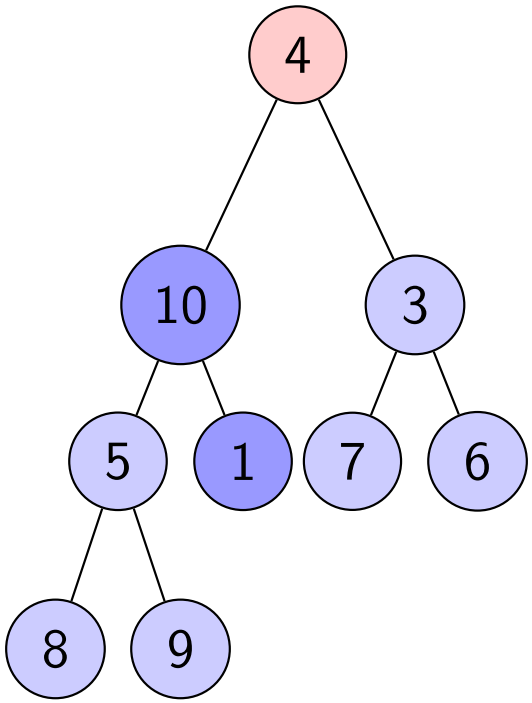
No swaps needed.

Step 5: Check Node 1 (InValid)



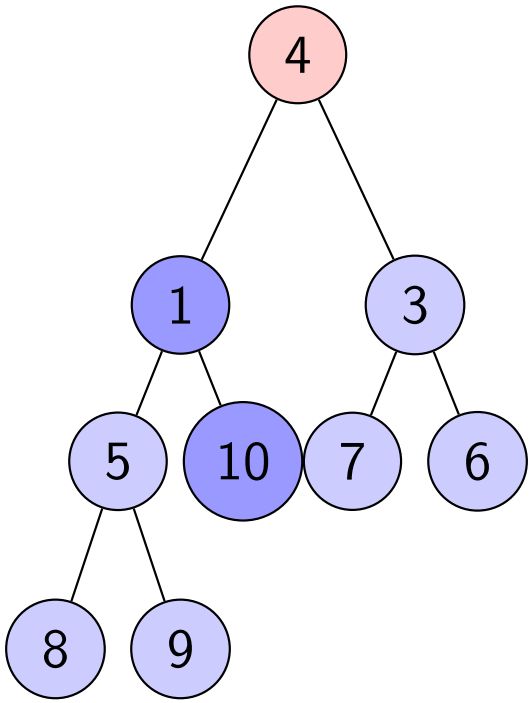
swap(currentNode, MIN([leftChild],[rightChild])).

Step 6: swap(node[1],node[4])



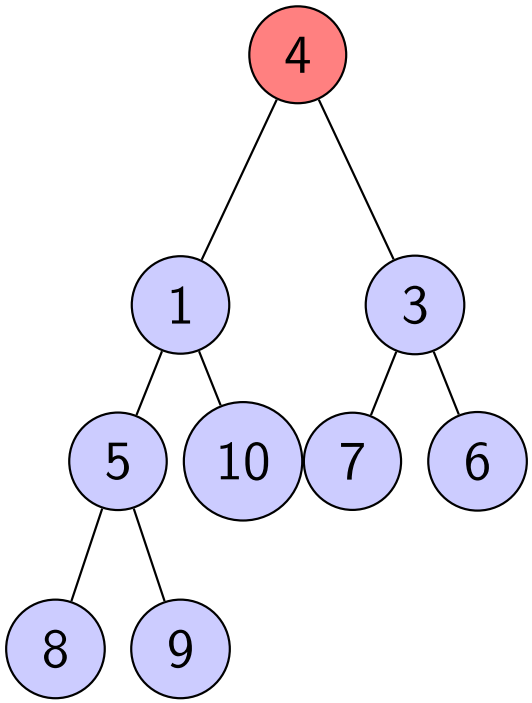
swap(currentNode, MIN([leftChild],[rightChild])).

Step 6: swap(node[1],node[4])



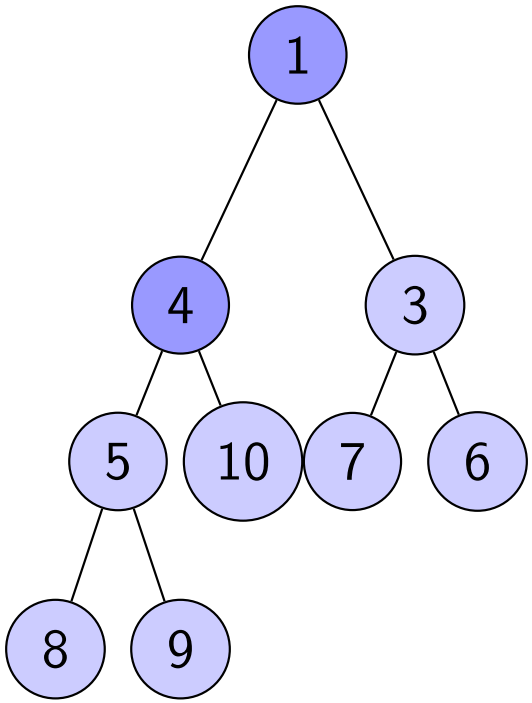
swap(currentNode, MIN([leftChild],[rightChild])).

Step 7: Check Node 0 (InValid)



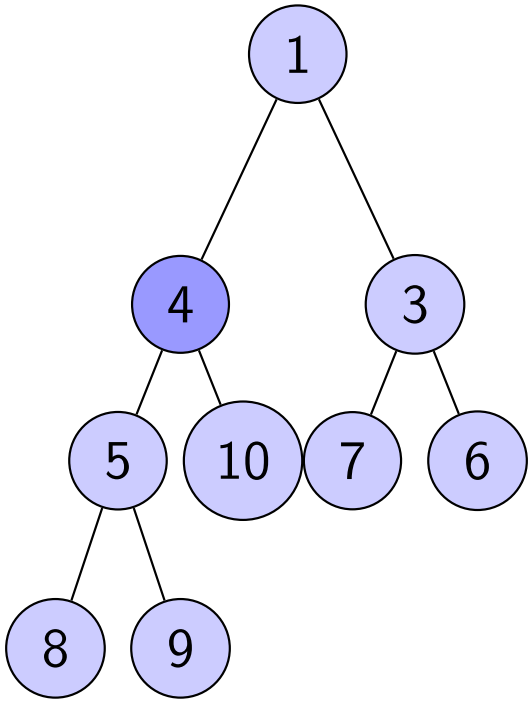
swap(currentNode, MIN([leftChild],[rightChild])).

Step 8: swap(node[0],[1])



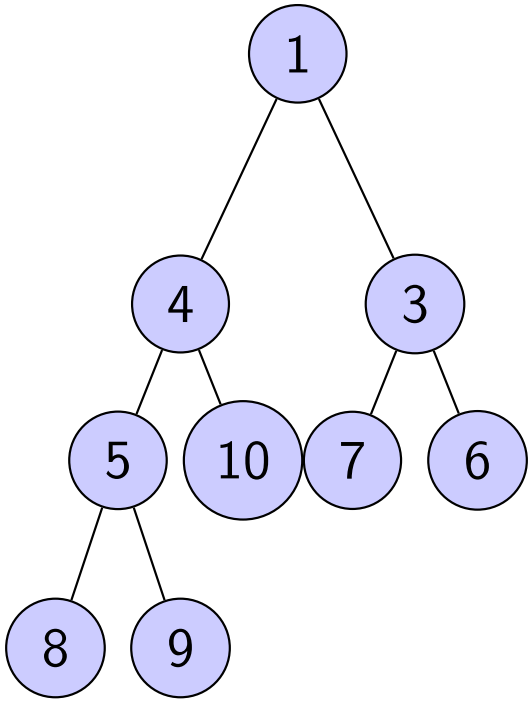
Recursively apply heapify on node[1]]

Step 9: Heapify



No swap needed.

Final Heap



What Happens at Each Step?

Heapify Explanation:

- Start from node i.
- Compare with left-hand($[2i+1]$) and right-hand($[2i+2]$) children.
- If a child is smaller, swap it with the parent.
- Recursively apply heapify.

HomeWork:

Figure out: How long this would take?

Thank You!

Thank you!