

CSE 332 Winter 2026

Lecture 4: Algorithm Analysis and Priority Queues

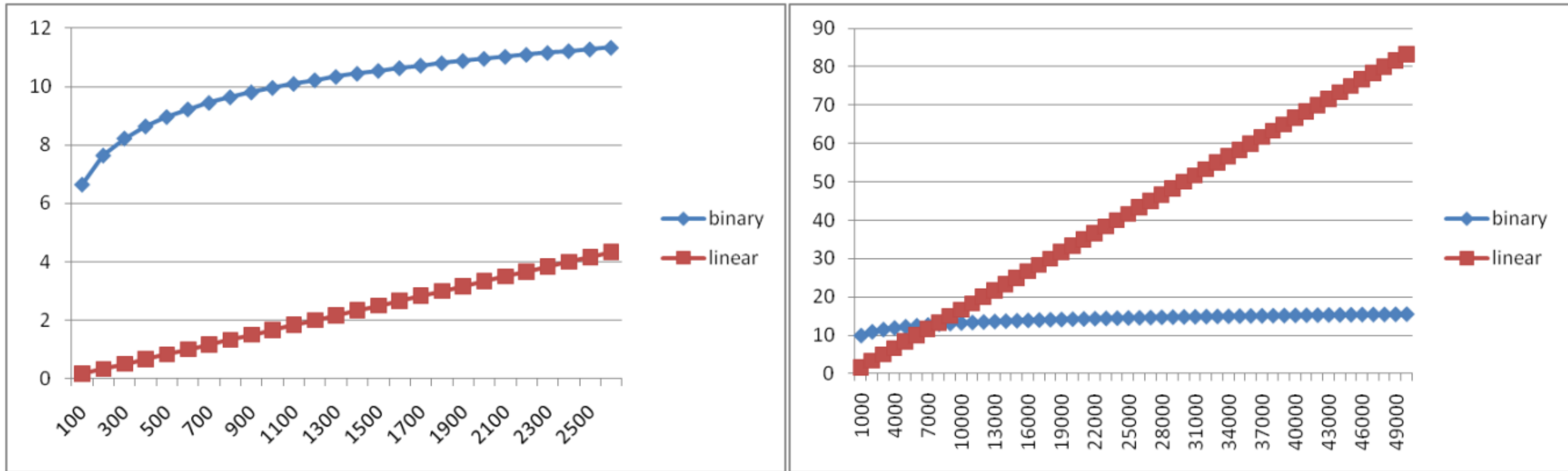
Nathan Brunelle

<http://www.cs.uw.edu/332>

Goals for Algorithm Analysis

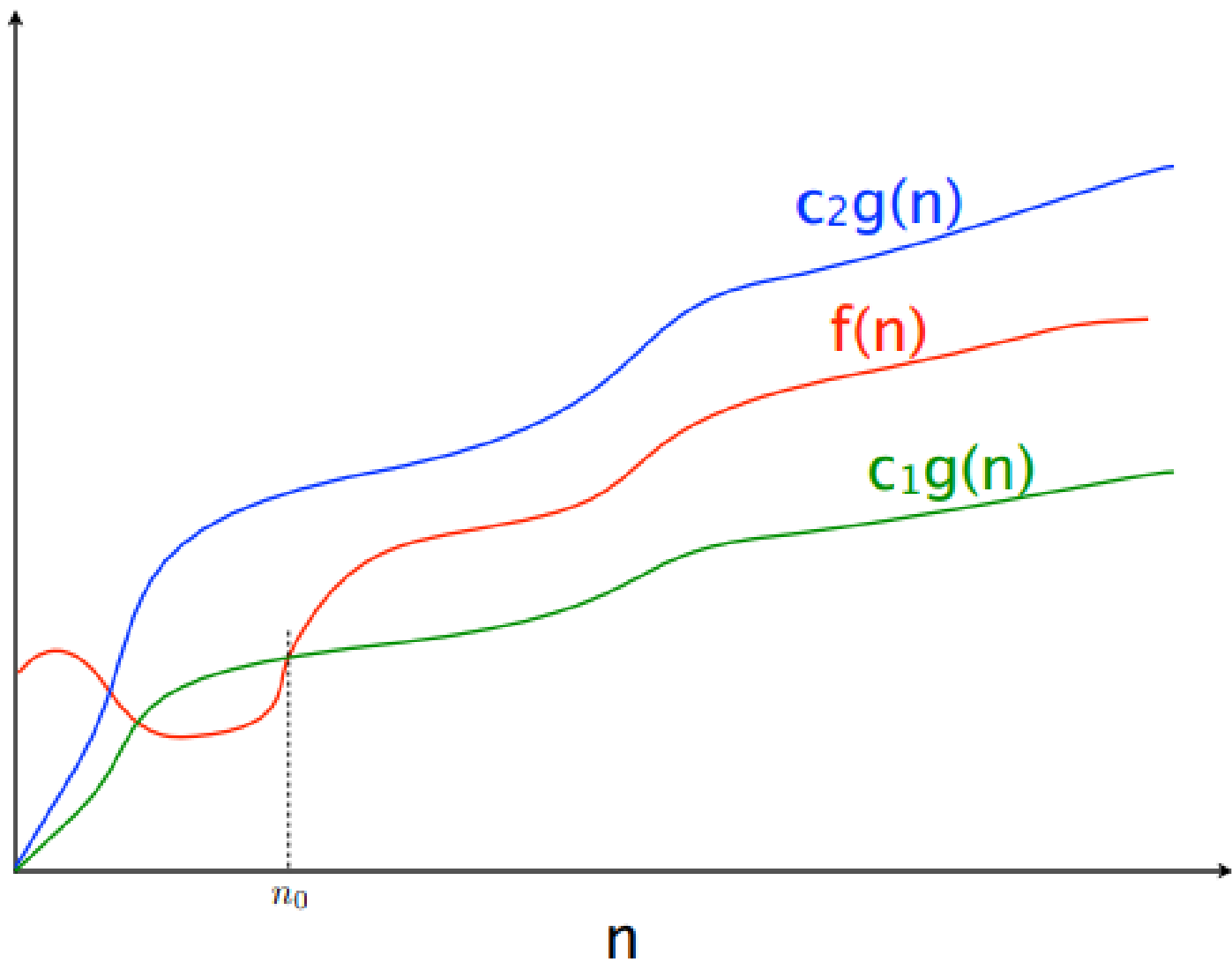
- Identify a *function* which maps the algorithm's input size to a measure of resources used
 - Domain of the function: **sizes** of the input
 - Number of characters in a string, number of items in a list, number of pixels in an image
 - Codomain of the function: **counts** of resources used
 - Number of times the algorithm adds two numbers together, number times the algorithm does a $>$ or $<$ comparison, maximum number of bytes of memory the algorithm uses at any time
- Important note: Make sure you know the “units” of your domain and codomain!
 - Domain = inputs to the function
 - Codomain = outputs to the function

Comparing



Comparing Running Times

- Suppose I have these algorithms, all of which have the same input/output behavior:
 - Algorithm A's worst case running time is $10n + 900$
 - Algorithm B's worst case running time is $100n - 50$
 - Algorithm C's worst case running time is $\frac{n^2}{100}$
- Which algorithm is best?



$$f(n) = O(g(n))$$

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

Asymptotic Notation

- $O(g(n))$
 - The **set of functions** with asymptotic behavior less than or equal to $g(n)$
 - **Upper-bounded** by a constant times g for large enough values n
 - $f \in O(g(n)) \equiv \exists c > 0. \exists n_0 > 0. \forall n \geq n_0. f(n) \leq c \cdot g(n)$
- $\Omega(g(n))$
 - the **set of functions** with asymptotic behavior greater than or equal to $g(n)$
 - **Lower-bounded** by a constant times g for large enough values n
 - $f \in \Omega(g(n)) \equiv \exists c > 0. \exists n_0 > 0. \forall n \geq n_0. f(n) \geq c \cdot g(n)$
- $\Theta(g(n))$
 - **“Tightly”** within constant of g for large n
 - $\Omega(g(n)) \cap O(g(n))$

Asymptotic Notation Example

- Show: $10n + 100 \in O(n^2)$
 - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n > n_0. 10n + 100 \leq c \cdot n^2$
 - **Scratch work:**

Asymptotic Notation Example

- Show: $10n + 100 \in O(n^2)$
 - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. 10n + 100 \leq c \cdot n^2$
 - **Proof:** Let $c = 10$ and $n_0 = 6$. Show $\forall n \geq 6. 10n + 100 \leq 10n^2$
$$\begin{aligned}10n + 100 &\leq 10n^2 \\ \equiv n + 10 &\leq n^2 \\ \equiv 10 &\leq n^2 - n \\ \equiv 10 &\leq n(n - 1)\end{aligned}$$

This is True because $n(n - 1)$ is strictly increasing and $6(6 - 1) > 10$

Asymptotic Notation Example

- Show: $13n^2 - 50n \in \Omega(n^2)$
 - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. 13n^2 - 50n \geq c \cdot n^2$
 - **Scratch work:**
 - We need $13n^2 - 50n \geq c \cdot n^2$, so we need $13n^2 \geq cn^2 + 50n$, dividing by n we have $13n \geq cn + 50$. When $n > 50$ we know that adding n is “more impactful” than adding 50, so if $n_0 = 50$ we can pick any value of $c \leq 12$.

Asymptotic Notation Example

- Show: $13n^2 - 50n \in \Omega(n^2)$
 - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. 13n^2 - 50n \geq c \cdot n^2$
 - **Proof:** let $c = 12$ and $n_0 = 50$. Show $\forall n \geq 50. 13n^2 - 50n \geq 12n^2$
$$\begin{aligned}13n^2 - 50n &\geq 12n^2 \\ \equiv n^2 - 50n &\geq 0 \\ \equiv n^2 &\geq 50n \\ \equiv n &\geq 50\end{aligned}$$
This is certainly true $\forall n \geq 50$.

Asymptotic Notation Example

- Show: $n^2 \notin O(n)$

Asymptotic Notation Example

Proof by
Contradiction!

- To Show: $n^2 \notin O(n)$
 - **Technique: Contradiction**
 - **Proof:** Assume $n^2 \in O(n)$. Then $\exists c, n_0 > 0$ s.t. $\forall n > n_0, n^2 \leq cn$
Let us derive constant c . For all $n > n_0 > 0$, we know:
 $cn \geq n^2,$
 $c \geq n.$

Since c is lower bounded by n , c cannot be a constant and make this True.

Contradiction. Therefore $n^2 \notin O(n)$.

Gaining Intuition

- When doing asymptotic analysis of functions:
 - If multiple expressions are added together, ignore all but the “biggest”
 - If $f(n)$ grows asymptotically faster than $g(n)$, then $f(n) + g(n) \in \Theta(f(n))$
 - Ignore all multiplicative constants
 - $f(n) + c \in \Theta(f(n))$ for any constant $c \in \mathbb{R}$
 - Ignore bases of logarithms
 - Do NOT ignore:
 - Non-multiplicative and non-additive constants (e.g. in exponents, bases of exponents)
 - Logarithms themselves
- Examples:
 - $4n + 5$
 - $0.5n \log n + 2n + 7$
 - $n^3 + 2^n + 3n$
 - $n \log(10n^2)$

More Examples

- Is each of the following True or False?
 - $4 + 3n \in O(n)$
 - $n + 2 \log n \in O(\log n)$
 - $\log n + 2 \in O(1)$
 - $n^{50} \in O(1.1^n)$
 - $3^n \in \Theta(2^n)$

Common Categories

- $O(1)$ “constant”
- $O(\log n)$ “logarithmic”
- $O(n)$ “linear”
- $O(n \log n)$ “log-linear”
- $O(n^2)$ “quadratic”
- $O(n^3)$ “cubic”
- $O(n^k)$ “polynomial”
- $O(k^n)$ “exponential”

ADT: Queue

- What is it?
 - A “First In First Out” (FIFO) collection of items
- What Operations do we need?
 - Enqueue
 - Add a new item to the queue
 - Dequeue
 - Remove the “oldest” item from the queue
 - Is_empty
 - Indicate whether or not there are items still on the queue

ADT: Priority Queue

- What is it?
 - A collection of items and their “priorities”
 - Allows quick access/removal to the “top priority” thing
- What Operations do we need?
 - insert(item, priority)
 - Add a new item to the PQ with indicated priority
 - Usually, smaller priority value means more important
 - deleteMin
 - Remove and return the “top priority” item from the queue
 - Is_empty
 - Indicate whether or not there are items still on the queue
- Note: the “priority” value can be any type/class so long as it’s comparable (i.e. you can use “<” or “compareTo” with it)

Priority Queue, example

```
PriorityQueue PQ = new PriorityQueue();
```

```
PQ.insert(5,5)
```

```
PQ.insert(6,6)
```

```
PQ.insert(1,1)
```

```
PQ.insert(3,3)
```

```
PQ.insert(8,8)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

Priority Queue, example

```
PriorityQueue PQ = new PriorityQueue();
```

```
PQ.insert(5,5)
```

```
PQ.insert(6,6)
```

```
PQ.insert(1,1)
```

```
Print(PQ.deleteMin)
```

```
PQ.insert(3,3)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

```
PQ.insert(8,8)
```

```
Print(PQ.deleteMin)
```

```
Print(PQ.deleteMin)
```

Applications?

Thinking through implementations

| Data Structure | Worst case time to insert | Worst case time to deleteMin |
|-----------------------|---------------------------|------------------------------|
| Unsorted Array | | |
| Unsorted Linked List | | |
| Sorted Circular Array | | |
| Sorted Linked List | | |
| Binary Search Tree | | |

Note: Assume we know the maximum size of the PQ in advance

Thinking through implementations

| Data Structure | Worst case time to insert | Worst case time to extract |
|----------------------|---------------------------|----------------------------|
| Unsorted Array | | |
| Unsorted Linked List | | |
| Sorted Array | | |
| Sorted Linked List | | |
| Binary Search Tree | | |

For simplicity, Assume we know the maximum size of the PQ in advance (otherwise we'd do an amortized analysis, but get the same answers...)

Thinking through implementations

| Data Structure | Worst case time to insert | Worst case time to extract |
|----------------------|---------------------------|----------------------------|
| Unsorted Array | $\Theta(1)$ | $\Theta(n)$ |
| Unsorted Linked List | $\Theta(1)$ | $\Theta(n)$ |
| Sorted Array | $\Theta(n)$ | $\Theta(1)$ |
| Sorted Linked List | $\Theta(n)$ | $\Theta(1)$ |
| Binary Search Tree | $\Theta(n)$ | $\Theta(n)$ |

For simplicity, Assume we know the maximum size of the PQ in advance (otherwise we'd do an amortized analysis, but get the same answers...)

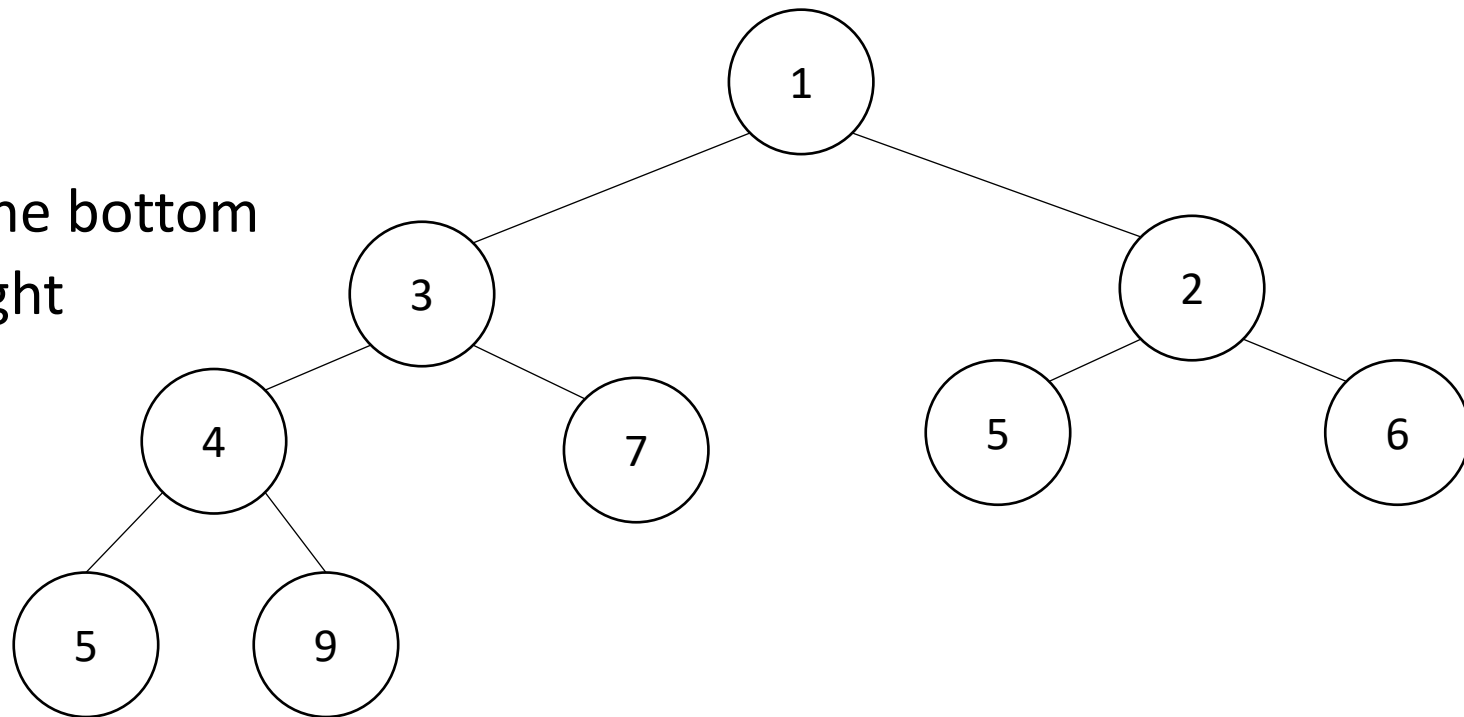
Thinking through implementations

| Data Structure | Worst case time to insert | Worst case time to extract |
|----------------------|---------------------------|----------------------------|
| Unsorted Array | $\Theta(1)$ | $\Theta(n)$ |
| Unsorted Linked List | $\Theta(1)$ | $\Theta(n)$ |
| Sorted Array | $\Theta(n)$ | $\Theta(1)$ |
| Sorted Linked List | $\Theta(n)$ | $\Theta(1)$ |
| Binary Search Tree | $\Theta(n)$ | $\Theta(n)$ |
| Binary Heap | $\Theta(\log n)$ | $\Theta(\log n)$ |

For simplicity, Assume we know the maximum size of the PQ in advance (otherwise we'd do an amortized analysis, but get the same answers...)

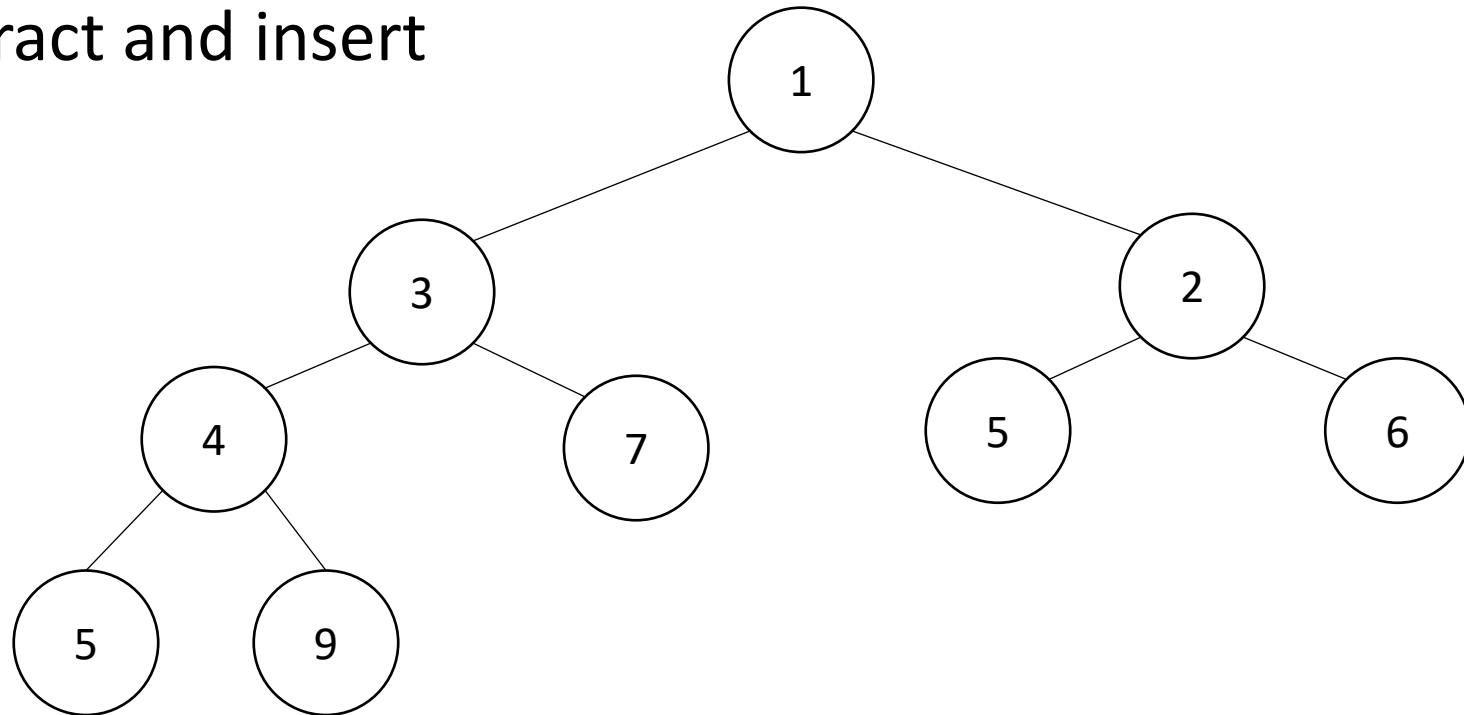
Trees for Heaps

- Binary Trees:
 - The branching factor is 2
 - Every node has ≤ 2 children
- Complete Tree:
 - All “layers” are full, except the bottom
 - Bottom layer filled left-to-right



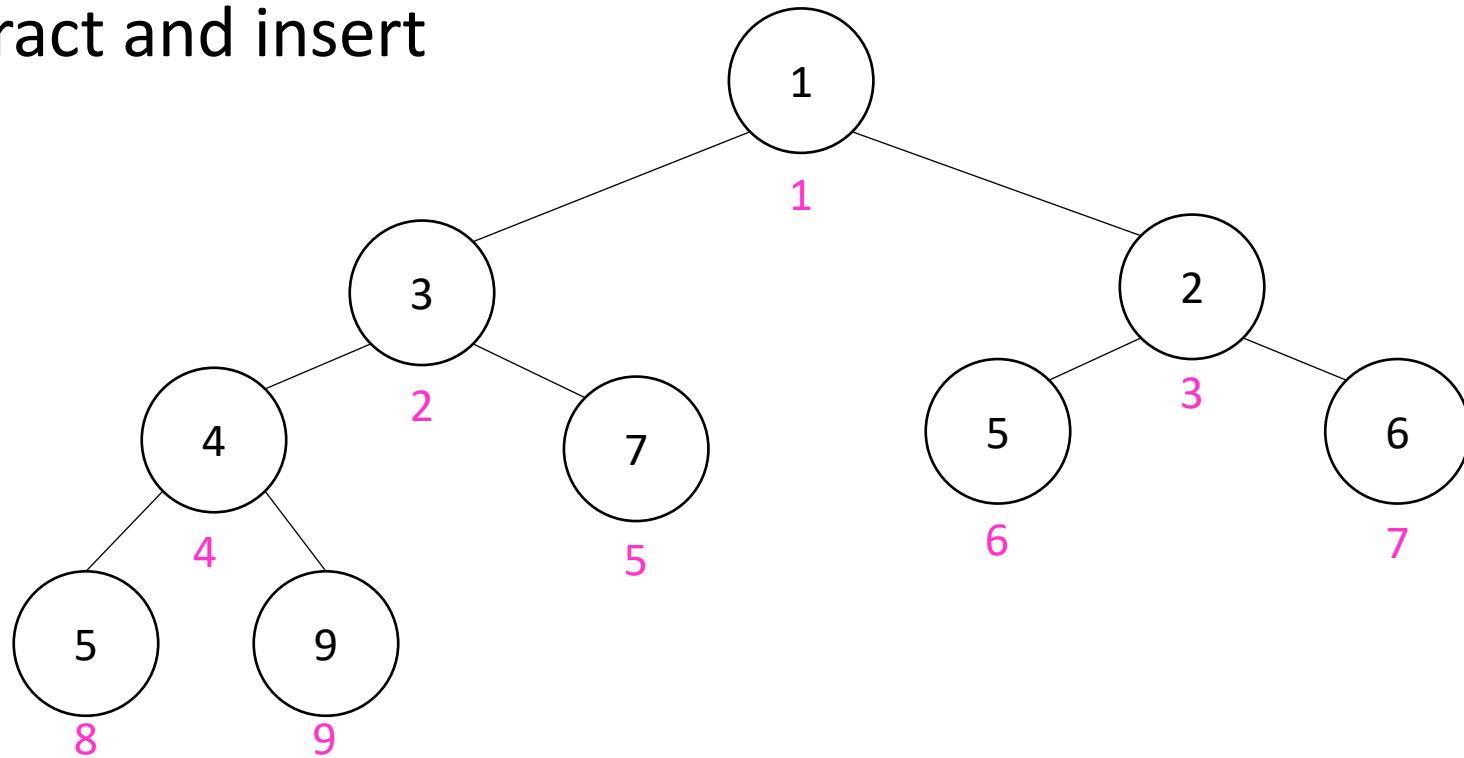
Heap – Priority Queue Data Structure

- Idea: We need to keep some ordering, but it doesn't need to be entirely sorted
- $\Theta(\log n)$ worst case for extract and insert



Heap – Priority Queue Data Structure

- Idea: We need to keep some ordering, but it doesn't need to be entirely sorted
- $\Theta(\log n)$ worst case for extract and insert



Challenge!

- What is the maximum number of total nodes in a binary tree of height h ?
 - $2^{h+1} - 1$
 - $\Theta(2^h)$
- If I have n nodes in a binary tree, what is its minimum height?
 - $\Theta(\log n)$
- **Heap Idea:**
 - If n values are inserted into a complete tree, the height will be roughly $\log n$
 - Ensure each insert and extract requires just one “trip” from root to leaf

(Min) Heap Data Structure

- Keep items in a complete binary tree
- Maintain the “(Min) Heap Property” of the tree
 - Every node's priority is \leq its children's priority
 - Max Heap Property: every node's priority is \geq its children
- Where is the min?
- How do I insert?
- How do I extract?
- How to do it in Java?

