CSE 332 Spring 2026
Lecture 4: Algorithm Analysis and Priority Queues

Nathan Brunelle

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

Motivation for Algorithm Analysis

- We want to define algorithm running times in a way that’s predictive:
  - We’re not required to implement+run the algorithms to determine the time
  - Our conclusions do not depend on how the algorithm is implemented/run
    - E.g. variation in choice of programming language, machine, or other tasks the computer is doing in parallel
  - We can use it to compare running times of different algorithms
  - We do not need to select an input size in advance

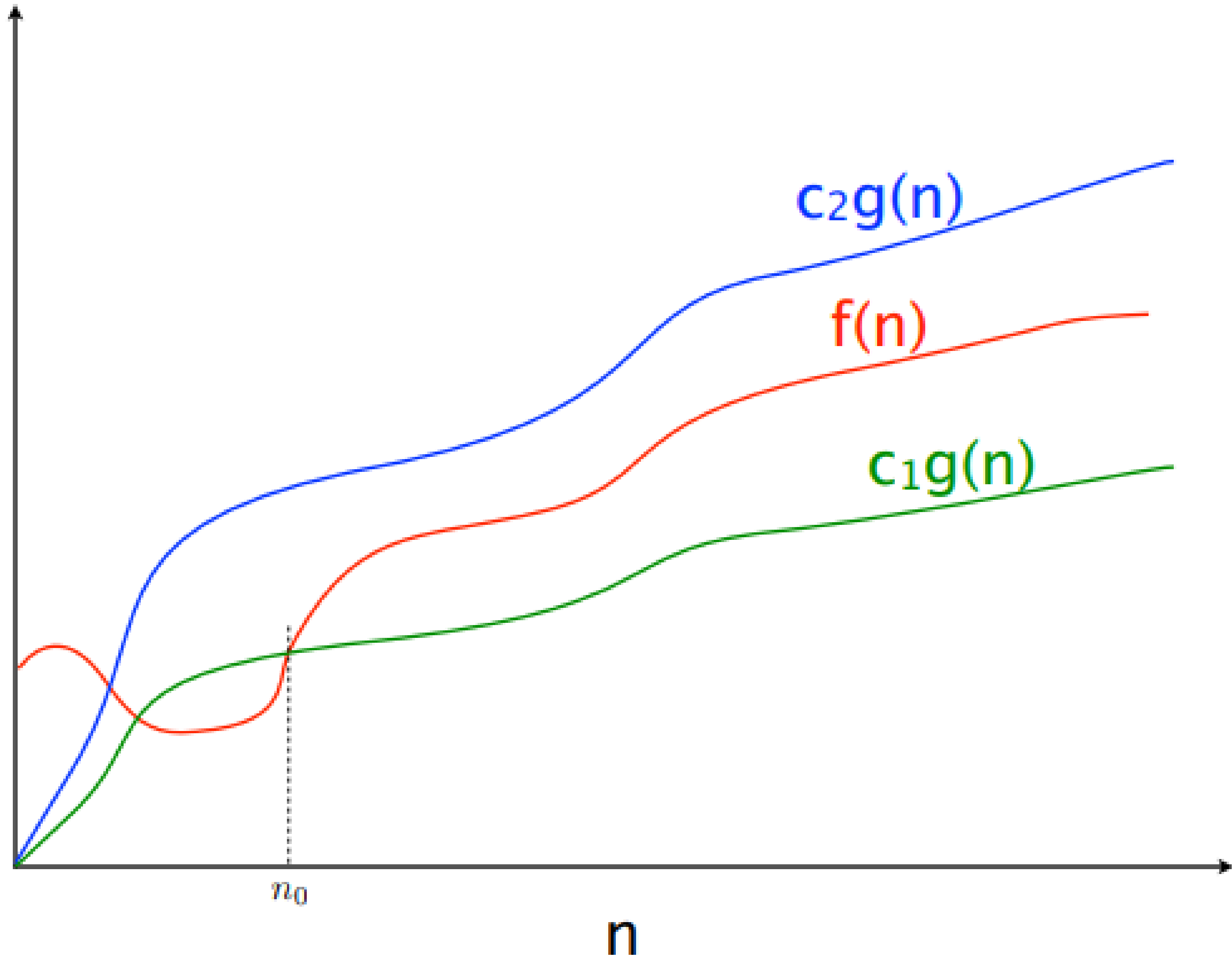
Process for Algorithm Analysis

- 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 Running Times

- To compare two algorithms' running times, we need a way to compare functions
- We want this comparison to have the following properties:
  - We answer based on long-term behavior (i.e. large inputs)
    - In most cases, all algorithms are fast on small inputs, so it's more important to compare for large inputs
  - We ignore constant coefficients and non-dominant terms
    - Different people might count different subsets of operations, which would give different constant coefficients in the running time
    - Even if we count the same operations, different operations take different amounts of time on different computers
    - Constant coefficients matter more than the non-dominant terms (e.g. $2n^2 > n^2 + n$)

Asymptotically comparing $f(n)$ with $g(n)$



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

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

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

Idea of Θ

- $x = y$
  - $x \leq y \land x \geq y$

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 1

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

Asymptotic Notation Example 1 (Scratchwork)

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

    We need $10n + 100 \leq c \cdot n^2$

    The algebra looks like it might be easier if we select $c = 10$

    Now we need $10n + 100 \leq 10n^2$

    Equivalently, $n + 10 \leq n^2$

    To select n_0 we need to consider what values of n will cause "add 10" to have less impact than "multiply by n".

    If $n \geq 10$ then $n + 10 \leq 2n$, and also $2n \leq 10n$, and also $10n \leq n^2$, so $n + 10 \leq n^2$

    Thus it should work to use $c = 10, n_0 = 10$

Asymptotic Notation Example 1 (Proof)

- 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 $n_0 = 10$. Show $\forall n \geq 10 \quad 10n + 100 \leq 10n^2$

$$10n + 100 \leq 10n^2$$
$$\equiv n + 10 \leq n^2$$
$$\equiv 10 \leq n^2 - n$$
$$\equiv 10 \leq n(n-1)$$

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

Asymptotic Notation Example 2

- Show: $13\mathrm{n}^2 - 50\mathrm{n} \in \Omega(n^2)$
  - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0.\, 13n^2 - 50n \geq c \cdot n^2$

Asymptotic Notation Example 2 (Scratchwork)

- Show: $13\mathrm{n}^2 - 50\mathrm{n} \in \Omega(n^2)$
  - **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0 .\, 13n^2 - 50n \geq c \cdot n^2$
  - **Scratchwork:**

    we need $13n^2 - 50n \geq c \cdot n^2$

    equivalently, $13n - 50 \geq c \cdot n$

    It looks like things might simplify if we pick $c = 13$

    Now we need $13n - 50 \geq 13n$

    equivalently we need $-50 \geq 0$

    hmmm….. That didn't work! But, it tells us that to make the left hand side bigger, it needs to be 50 larger than the right! This means we definitely need $c < 13$

    let's instead use $c = 12$, now we need $13n - 50 \geq 12n$

    This simplifies to $n \geq 50$, so we can use 50 as n_0

Asymptotic Notation Example 2 (Proof)

- Show: $13\mathrm{n}^2 - 50\mathrm{n} \in \Omega(n^2)$
  - **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 $\forall n \geq 50$ we have $13n^2 - 50n \geq 12 \cdot n^2$
  Applying algebra:
  $13n^2 - 50n \geq 12n^2$
  $\equiv n^2 - 50n \geq 0$
  $\equiv n - 50 \geq 0$ (allowed because $n \neq 0$ since $n \geq 50$)
  $\equiv n \geq 50$
  Because we chose $n_0 = 50$, the inequality holds for all values of $n \geq n_0$

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 $f(n) + g(n) \in \Theta\big(f(n)\big)$
  - Ignore all multiplicative constants
    - $f(n) + c \in \Theta\big(f(n)\big)$ 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?
  - $4 + 3n \in O(n)$
  - $n + 2\log n \in O(\log n)$
  - $\log n + 2 \in O(1)$
  - $n^{50} \in O(1.1^n)$
  - $3^n \in \Theta(2^n)$

Solutions

- Is each of the following True or False? Justifications are “semi-intuitive” rather than rigorous:
  - $4 + 3n \in \ O(n)$
    - True because for large inputs, $4 + 3n < n + 3n = 4n$
  - $n + 2\log n \in \ O(\log n)$
    - False because for any constant c eventually $n > c \log n$ (specifically, when $\frac{2^n}{n} > c$)
  - $\log n + 2 \in \ O(1)$
    - False because for any constant c eventually $\log n > c$ (specifically when $n > 2^c$)
  - $n^{50} \in \ O(1.1^n)$
    - True because for large n it will be that $50 \log_{1.1} n \leq n$
  - $3^n \in \Theta(2^n)$
    - False because $3^n \leq c2^n$ is true iff $(\log_2 3)\, n \leq \log_2 c + n$, and eventually multiplying by $\log_2 3$ has more impact than adding $\log_2 c$ (for example, when $n > \frac{\log_2 c}{\log_2 3 - 1}$)

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
  - extract
    - 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 1

```
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.extract)
Print(PQ.extract)
Print(PQ.extract)
Print(PQ.extract)
Print(PQ.extract)
```

Prints:
1
3
5
6
8

Priority Queue Example 2

```
PriorityQueue PQ = new PriorityQueue();
PQ.insert(5,5)
PQ.insert(6,6)
PQ.insert(3,3)
Print(PQ.extract)
PQ.insert(1,1)
Print(PQ.extract)
Print(PQ.extract)
PQ.insert(8,8)
Print(PQ.extract)
Print(PQ.extract)
```

Prints:
3
1
5
6
8

Applications?

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…)

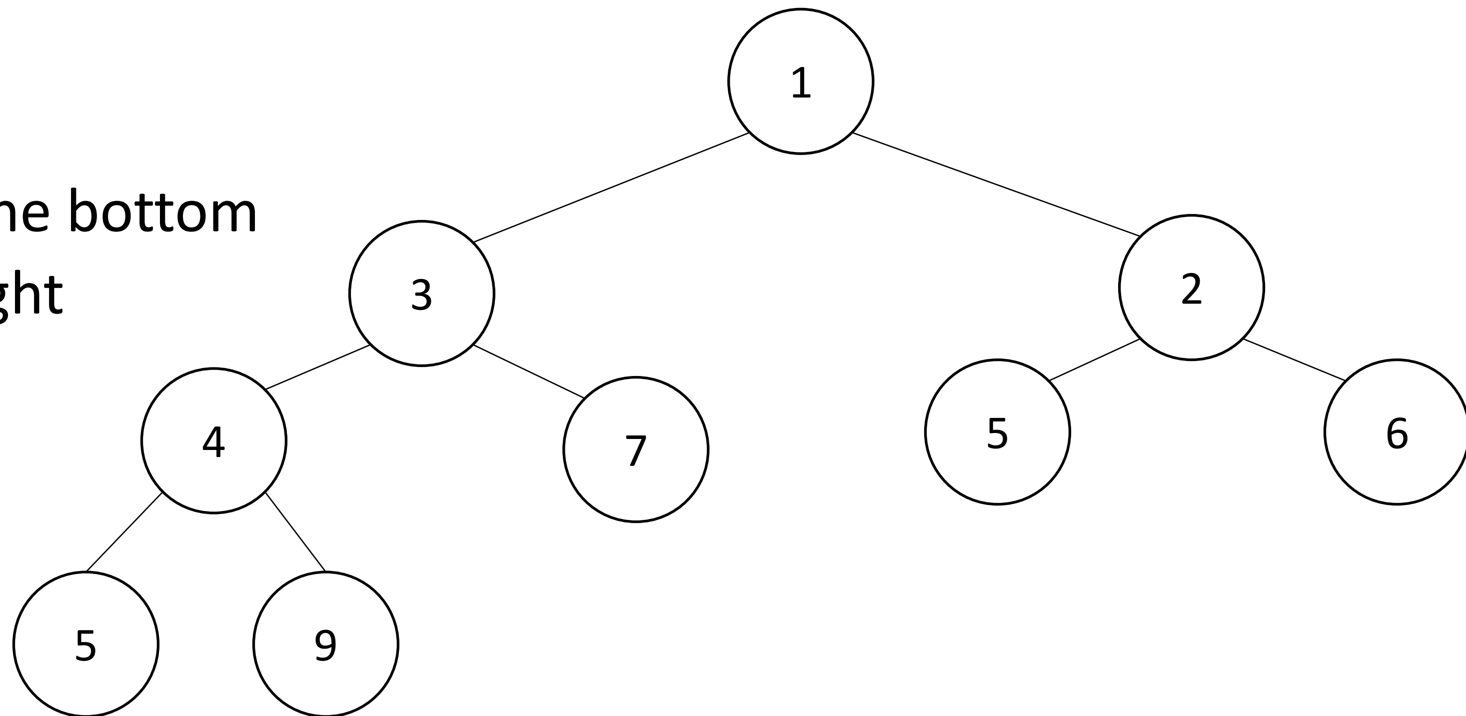
Thinking through implementations (solutions)

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…)

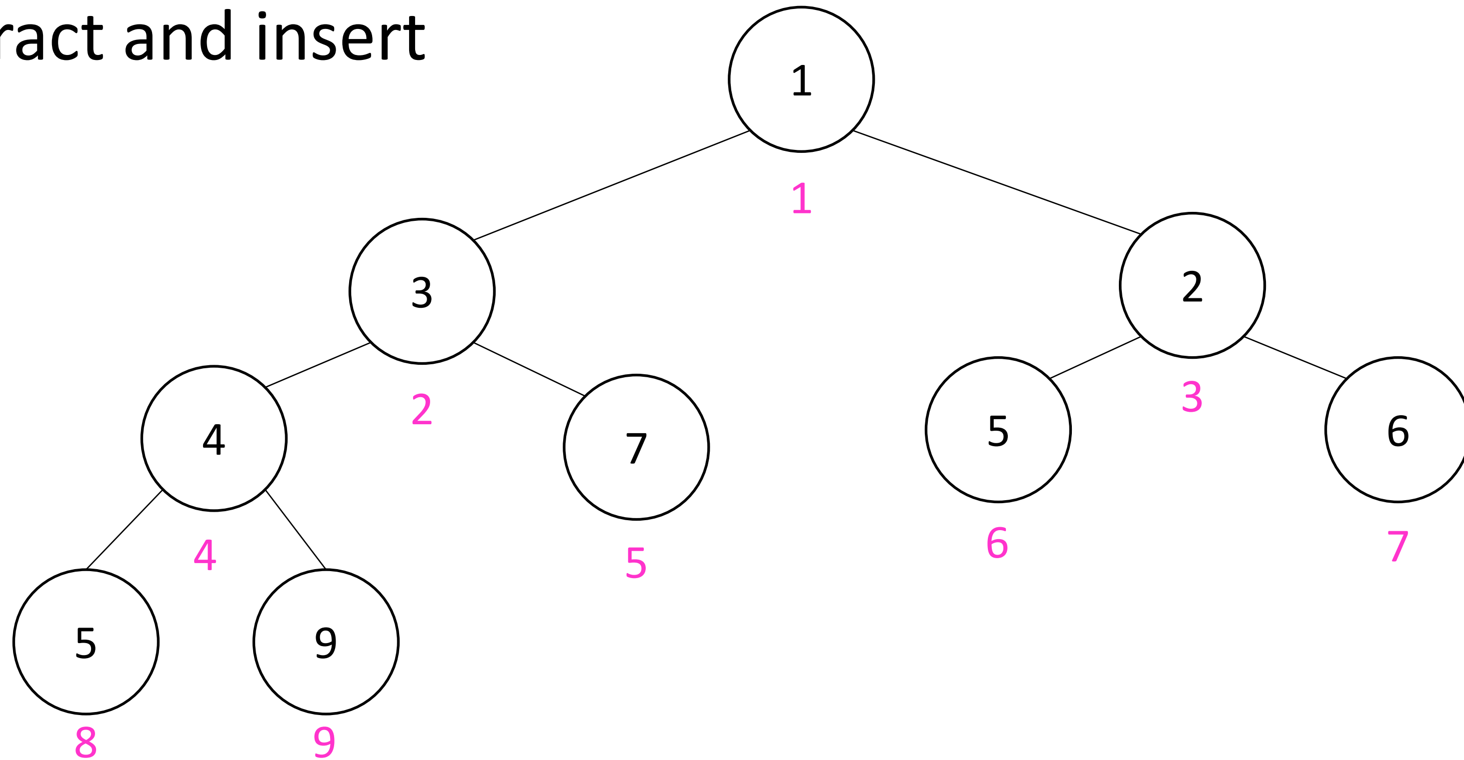
Trees for Heaps

- 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



Tree Measurements 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 ≤ its children's priority
  - Max Heap Property: every node's priority is ≥ its children
- Where is the min?
- How do I insert?
- How do I extract?
- How to do it in Java?