CSE 332 Summer 2026
Lecture 3: Algorithm Analysis pt.2

Michael Whitmeyer

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

Announcements

- CC0 due tonight
- Assignment 1 should be released today

Running Time Analysis

- Units of “time”
  - Operations
    - Whichever operations we pick
- How do we express running time?
  - Function
    - Domain (input): size of the input
    - Range: count of operations

Options for your running time function

- Worst-case complexity:
  - max number of steps algorithm takes on “most challenging” input
- Best-case complexity:
  - min number of steps algorithm takes on “easiest” input
- Average/expected complexity:
  - avg number of steps algorithm takes on random inputs (context-dependent)
- Amortized complexity:
  - max total number of steps algorithm takes on M “most challenging” consecutive inputs, divided by M (i.e., divide the max total sum by M).

Amortized Analysis Analogy

- Suppose I'd like to park in a lot where they charge \$10 per day to park
- If you are caught in the lot without paying you are given a warning
- If you get 3 warnings, you are charged a \$25 fine, and your warnings reset.
- Should you actually pay to park?
  - If you pay every day then you pay an average of \$10 per day
  - If you do not pay then for every three days parking costs \$0+\$0+\$25, for an average of \$8.33 per day
    - This is an amortized analysis

ArrayList - Worst Case Time of Add

```
public void add(int value){
    if(data.length == size)
        resize();
    data[size] = value;
    size++;
}
private void resize(){
    int[] oldData = data;
    data = new int[data.length*2];
    for(int i = 0; i < oldData.length; i++)
        data[i] = oldData[i];
}
```

- What is the worst case running time of add?
  - Input size: size of “this”
  - Operations counted: indexing
  - $O(n)$

ArrayList – Amortized Time of Add

```
public void add(int value){
    if(data.length == size)
        resize();
    data[size] = value;
    size++;
}
private void resize(){
    int[] oldData = data;
    data = new int[data.length*2];
    for(int i = 0; i < oldData.length; i++)
        data[i] = oldData[i];
}
```

Every time we resize, we earn `data.length` more adds before the next resize!

- Amortized Analysis Idea:
  - Suppose we have a program that in total does n adds.
  - How much time was spent "on average" across all n?
- Let c be the initial size of `data`
  - The first c adds take: $c + c = 2c$
  - The next $2c$ adds: $2c + 2c = 4c$
  - The next $4c$ adds: $4c + 4c = 8c$
  - Overall: $\frac{14c}{7c} = 2c$

Searching in a Sorted List

5	8	13	42	75	79	88	90	95	99
0	1	2	3	4	5	6	7	8	9

```
contains(int k, List L):
    for i=0 to L.size:
        if L[i] == k:
            return true
    return false
```

Faster way?

5	8	13	42	75	79	88	90	95	99
0	1	2	3	4	5	6	7	8	9

Can you think of a faster algorithm to solve this problem?

Binary Search

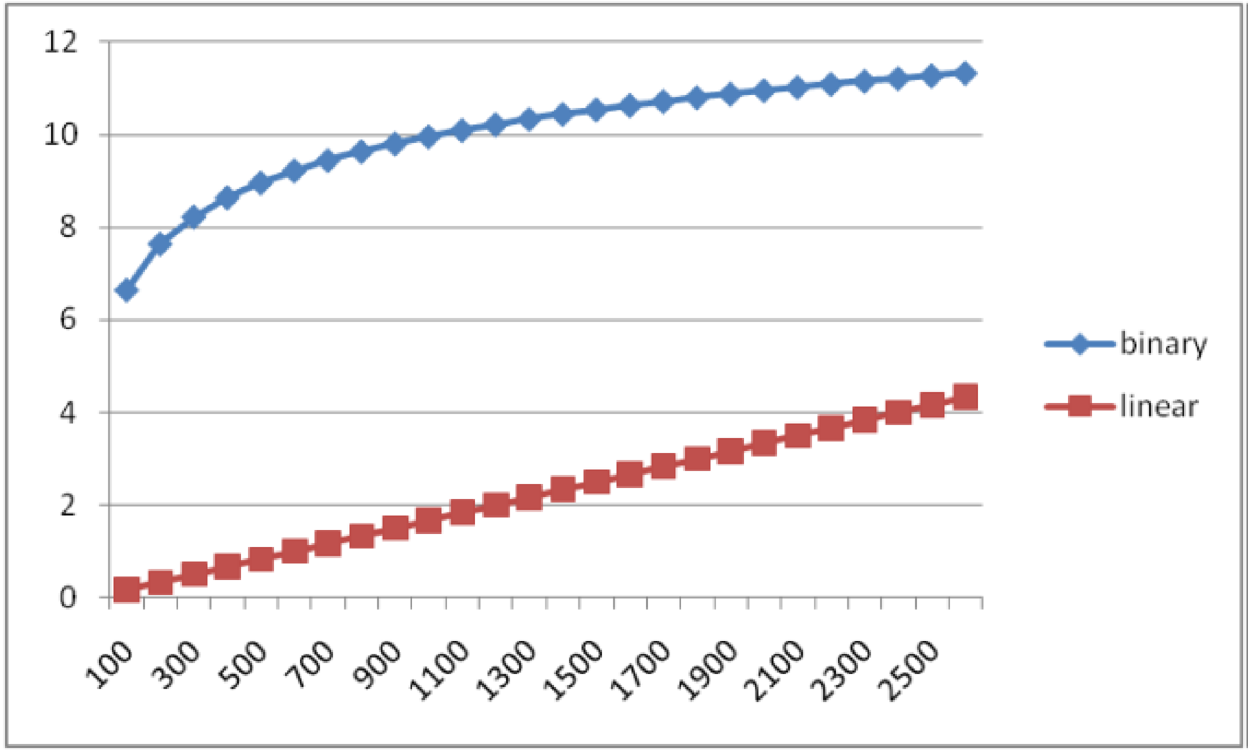
5	8	13	42	75	79	88	90	95	99
0	1	2	3	4	5	6	7	8	9

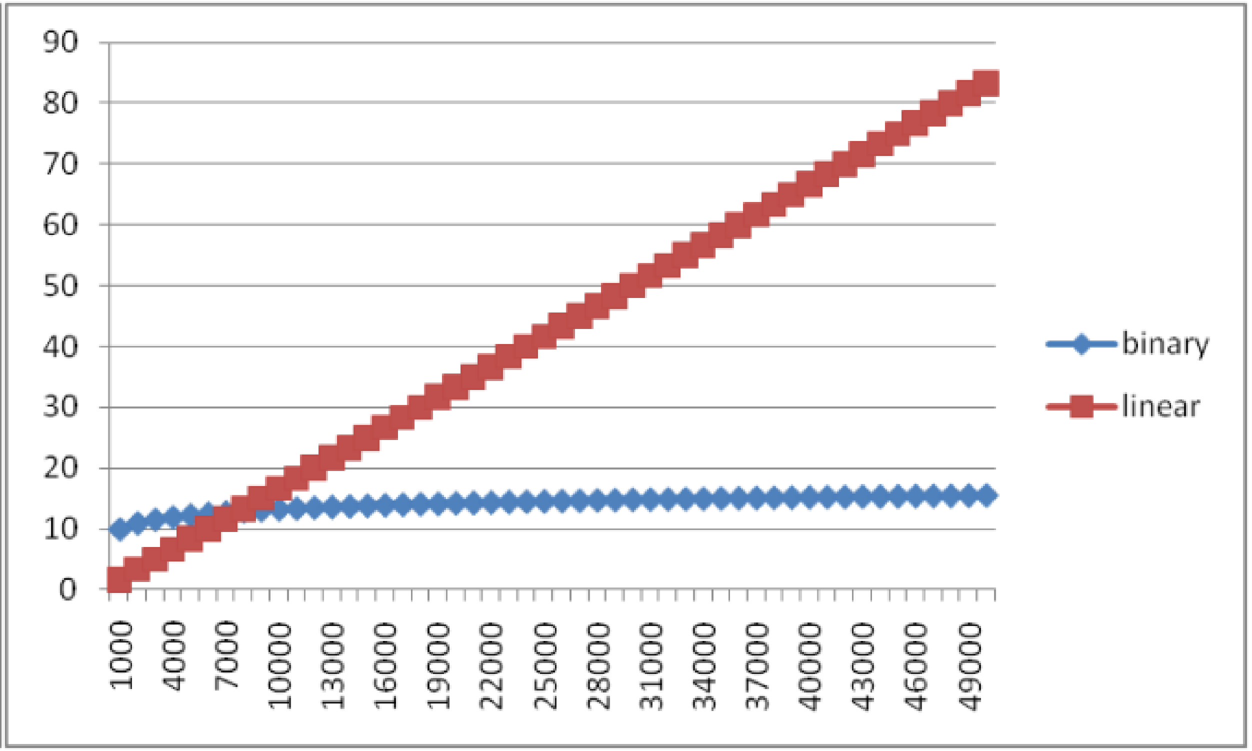
```
contains(List L, int k):
    start = 0
    end = L.size
    while(start < end):
        mid = start + (end-start)/2;
        if L[mid] == k:
            return true
        else if L[mid] < k:
            start = mid+1
        else:
            end = mid
    return false
```

Why is this $\log_2 n$?

- In the beginning: `end-start`=n
- After 1 iteration: `end-start`=$\frac{n}{2}$
  - `mid-start = (start+(end-start)/2)-start`
  - `end-mid = end-(start+(end-start)/2)`
- Each iteration cuts the “gap” in half!
  - x iterations the size is $\frac{n}{2^x}$
- We stop when the gap is 1
  - Solve for x: $\frac{n}{2^x} = 1$

Running Time Graphed

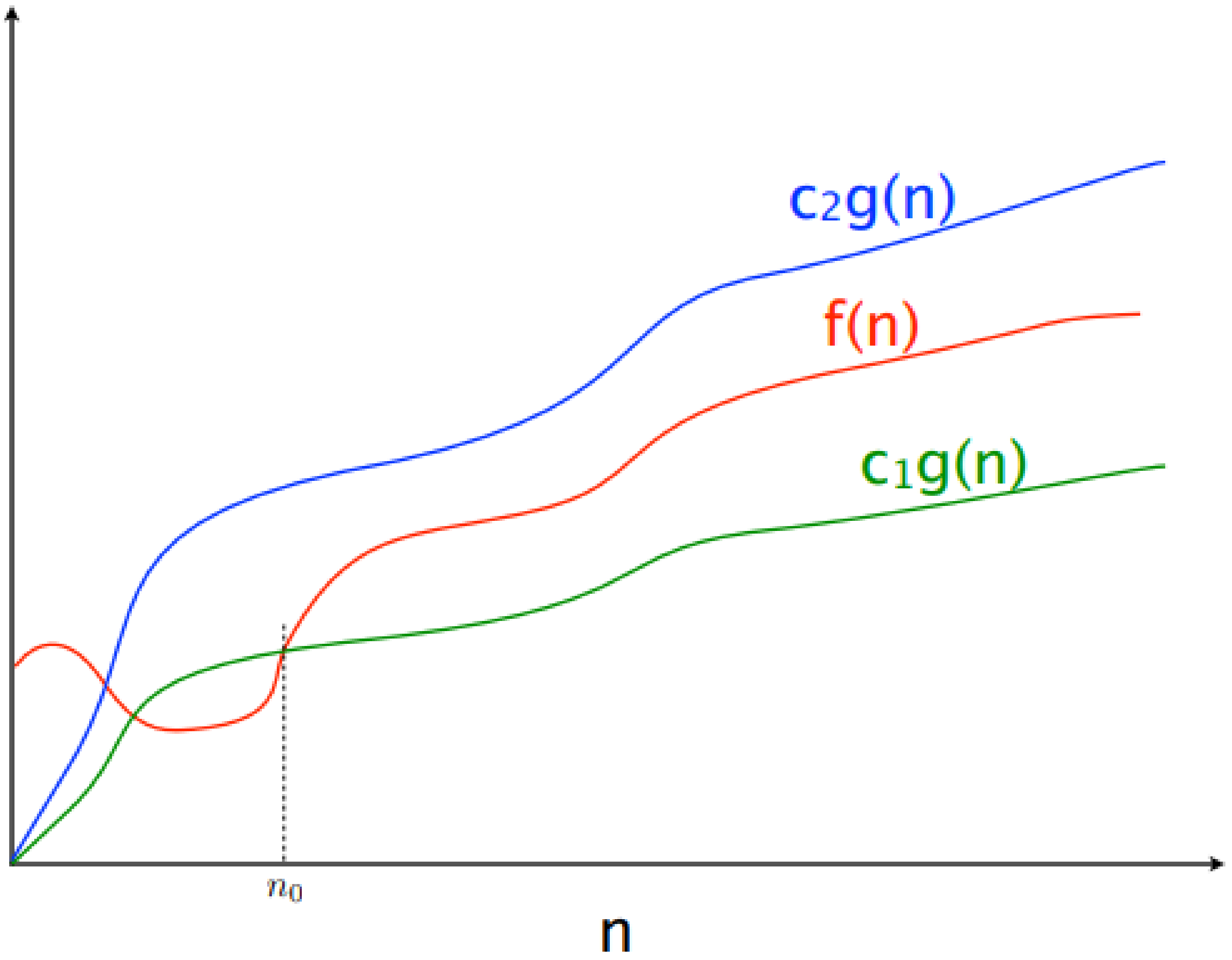




Comparing Running Times

- 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?

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



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

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

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

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
  - $\Omega(g(n)) \cap O(g(n))$

Idea of Θ

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

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 $n \geq n_0$

Asymptotic Notation Example 3

- Show: $n^2 \notin O(n)$
- Want to show that there does not exist a pair of c and n_0 such that $\forall n_0 > n. n^2 \leq c \cdot n$

Asymptotic Notation Example 3 (Proof)

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

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"