CSE 332 Autumn 2023 Lecture 2: Algorithm Analysis and Priority Queues

Nathan Brunelle

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

Warm Up

Questions to ask:

- What are the units of the input size?
- What are the operations we're counting?
- For each line:
  - How many times will it run?
  - How long does it take to run?
  - Does this change with the input size?

Give the worst case running time for the following code

```
doSomething(List myList){
    n = myList.size();
    x = 0;
    for (i=0; i < n; i++){
        for (j=0; j < i; j++){
            x++;
        }
    }
    return x;
}
```

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

Worst Case Running Time Analysis

- If an algorithm has a worst case running time of $f(n)$
  - Among all possible size-n inputs, the “worst” one will do $f(n)$ “operations”
  - I.e. $f(n)$ gives the maximum operation count from among all inputs of size n

Worst Case Running Time Analysis

- If an algorithm has a worst case running time of $f(n)$
  - Among all possible size-n inputs, the “worst” one will do $f(n)$ “operations”
  - I.e. $f(n)$ gives the maximum operation count from among all inputs of size n

Worst Case Running Time – General Guide

- Add together the time of consecutive statements
- Loops: Sum up the time required through each iteration of the loop
  - If each takes the same time, then [time per loop $\times$ number of iterations]
- Conditionals: Sum together the time to check the condition and time of the slowest branch
- Function Calls: Time of the function's body
- Recursion: Solve a recurrence relation

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

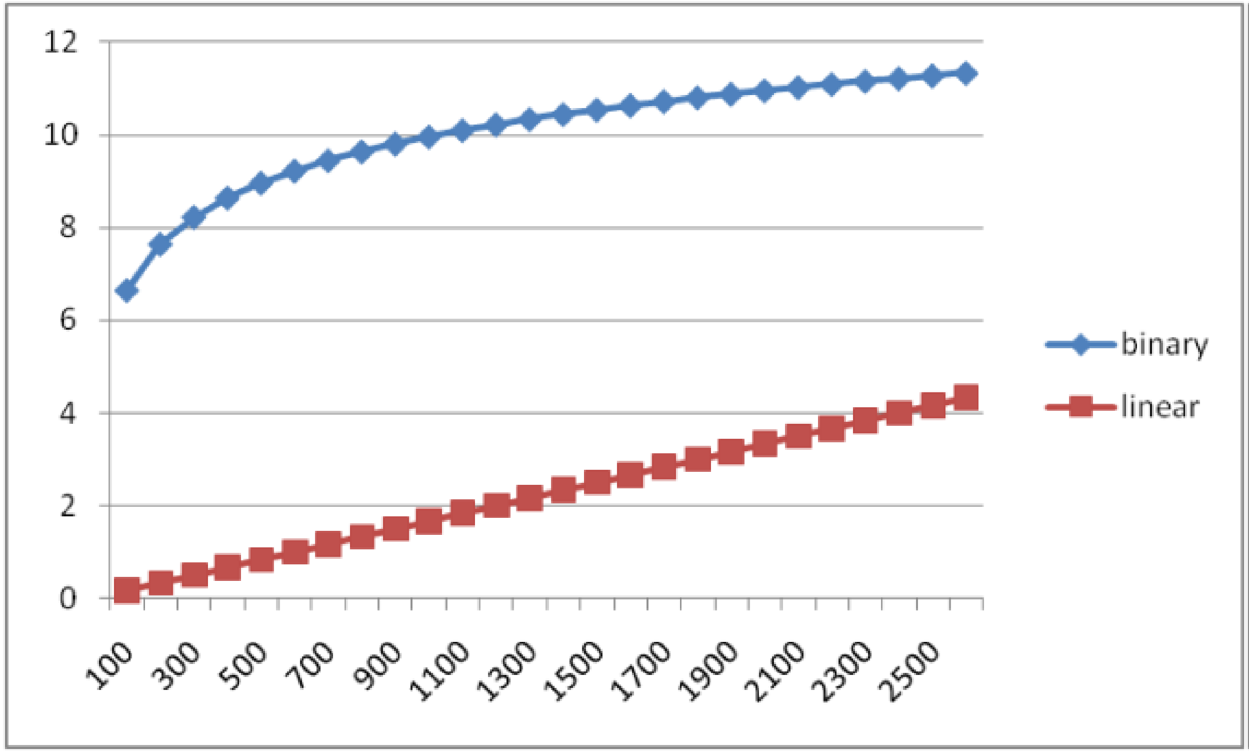
```
boolean linearSearch(array a, integer k){
        for(i=0; i< a.length; i++){
                if (a[i] == k){
                        return true;
                }
        }
        return false;
}
```

Faster way?

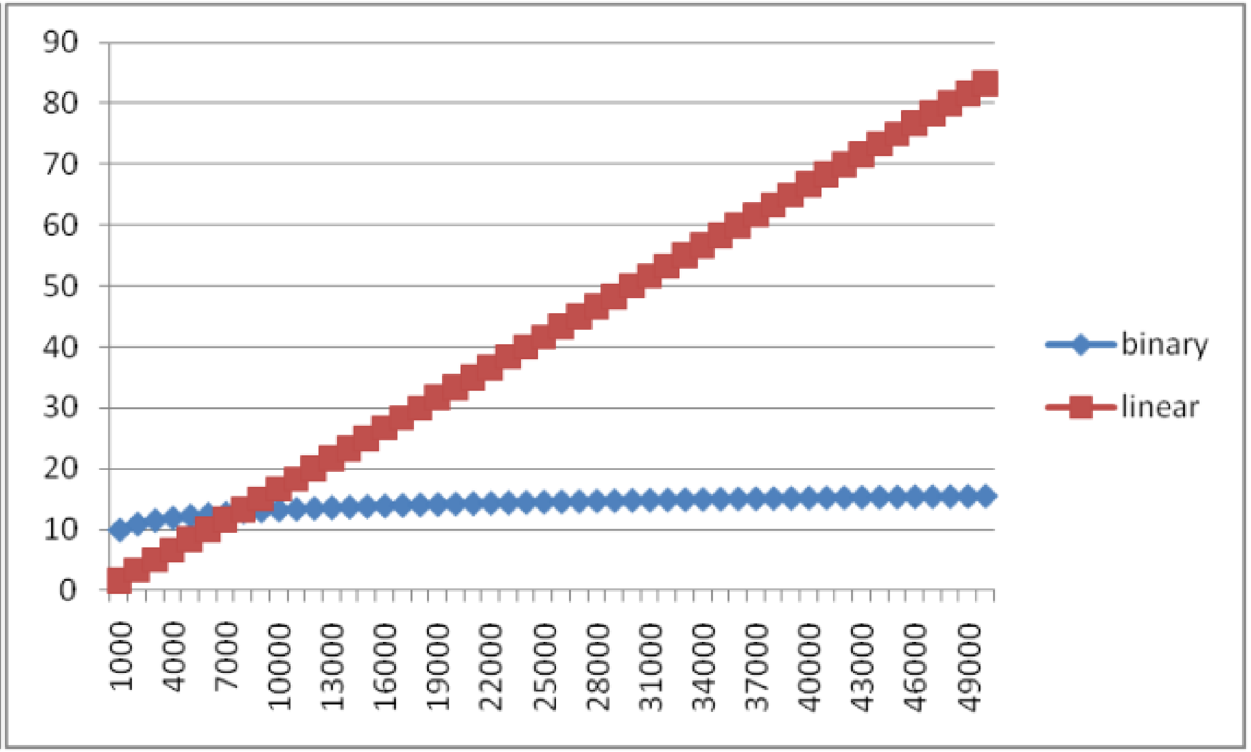
5	8	13	42	75	79	88	90	95	99
0	1	2	3	4	5	6	7	8	9

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

Comparing



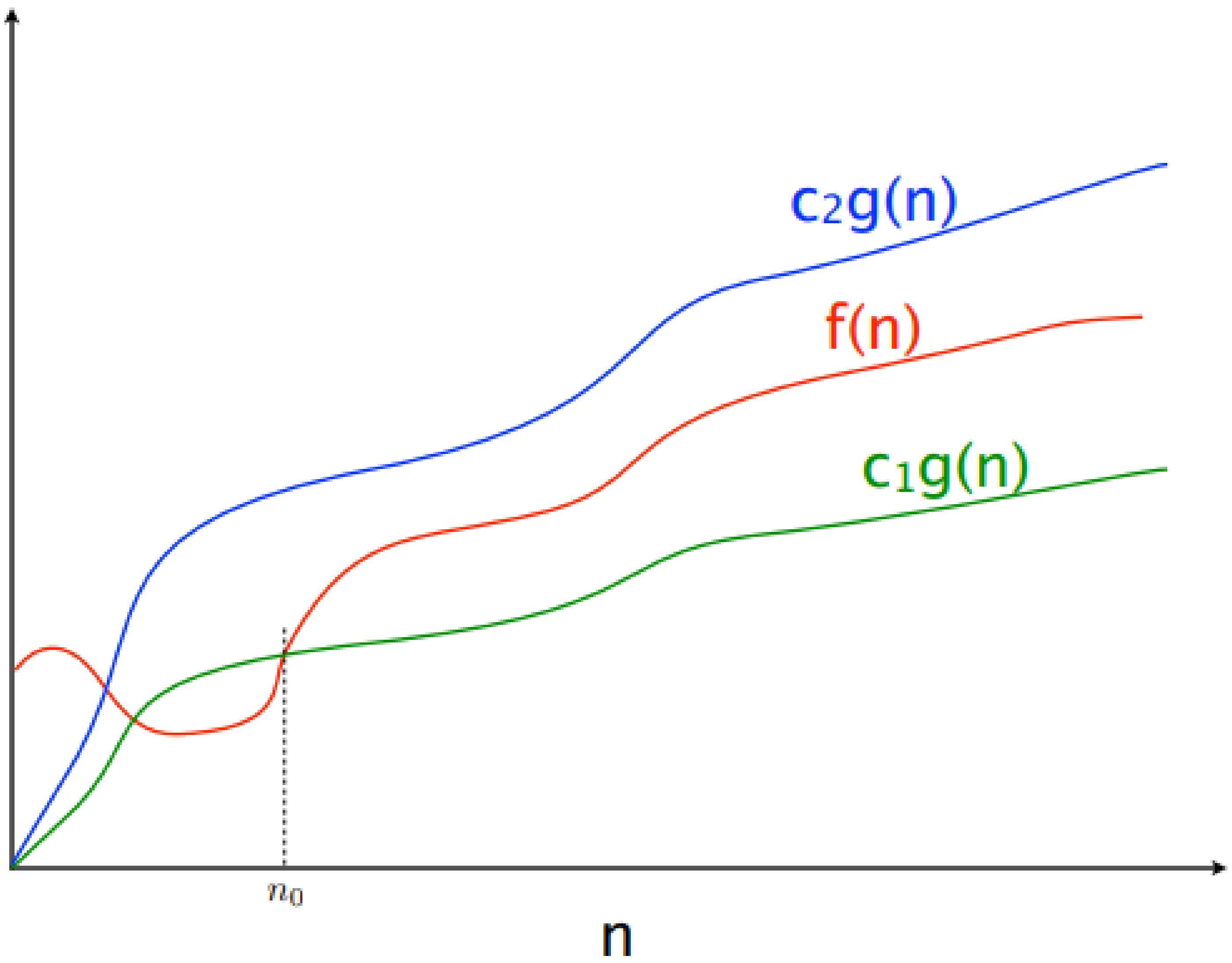
12
10
8
6
4
2
0
100
300
500
700
900
1100
1300
1500
1700
1900
2100
2300
2500
binary
linear



90
80
70
60
50
40
30
20
10
0
1000
4000
7000
10000
13000
16000
19000
22000
25000
28000
31000
34000
37000
40000
43000
46000
49000
binary
linear

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 $\forall n > n_0.\, 10n + 100 \leq c \cdot n^2$
  - **Proof:**

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 $\forall n \geq 6 . 10n + 100 \leq 10n^2$

    $10n + 100 \leq 10n^2$

    $\equiv n + 10 \leq n^2$

    $\equiv 10 \leq n^2 - n$

    $\equiv 10 \leq n(n-1)$

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

Asymptotic Notation Example

- 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:**

Asymptotic Notation Example

- 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 $\forall n \geq 50.\, 13n^2 - 50n \geq 12n^2$

$$13n^2 - 50n \geq 12n^2$$
$$\equiv n^2 - 50n \geq 0$$
$$\equiv n^2 \geq 50n$$
$$\equiv n \geq 50$$

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 $f(n) + g(n) \in \Theta\big(f(n)\big)$
  - 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”

Defining 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).

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