sit with your clan
if you can

CMPS 12B/M
Introduction to Data Structures

- Instructor: Nathan Whitehead

How fast is HighArray?

- How many steps does it take on average to do these operations for HighArray?
  - (Assume no duplicates allowed)
  - Let *n* be number of elements in array.
  - Insert
  - Delete
  - Find

What are we discovering?

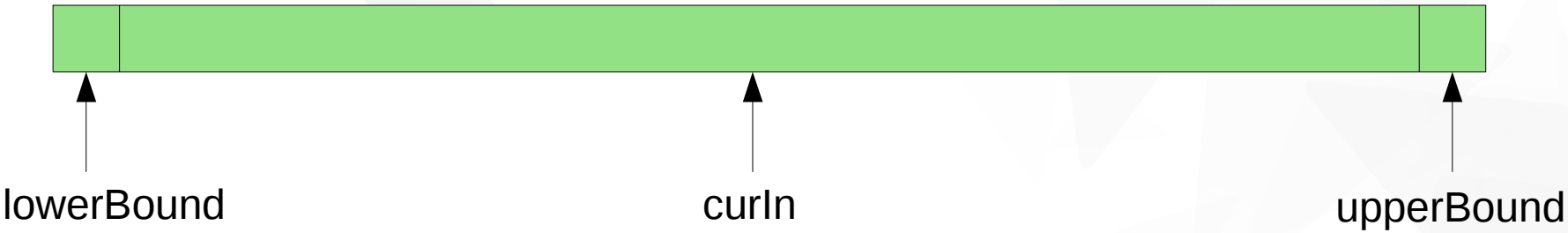
- Need to carefully define the thing we're counting
  - "step" is ambiguous
  - "comparison" or "array access" are better
- Insert is faster than delete
- Find is linear in size of array

How fast is OrderedArray?

- How many steps does it take on average to do these operations for OrderedArray?
  - (Assume no duplicates allowed)
  - Let *n* be number of elements in array.
  - Insert
  - Delete
  - Find

Binary Search

- You're looking for a needle in a haystack
- Compare needle to middle hay
  - If it matches, you're done
  - If it is smaller, then find the needle in the left haystack
  - If it is larger, then find the needle in the right haystack
- To keep track of where we're looking:
  - lowerBound, curIn, upperBound
  - Keep going until lowerBound > upperBound



Questions

- How many comparisons will it take?
- Does it matter if we pick the "upper" middle or the "lower" middle?
- Experiment time
  - Run the algorithm for n=10
  - Each group pick randomly: upper/lower middle
  - Use seat row last digit for which number to search for, in:
    - 0 1 2 3 4 5 6 7 8 9

Formula for Comparisons

- What function is the number of comparisons as n gets bigger?
- Worse case:
  - n=10 4
  - n=100 7
  - n=1000 10
  - n=10000 14
  - n=100000 17

What function looks like this?

$$\log_2(n)$$

Recurrence Relation

- Another way of thinking about the question...
  - Let T(n) be the function we're trying to find

- What happens when you double the number of businesses?

  - For OrderedArray:
    - T(2n) = ___ T(n) ___

You Can Solve Recurrences

- Techniques
  - Guess and check (the most powerful technique)
  - Look for patterns (linear, quadratic, logarithmic, exponential)
  - Ask Wolfram Alpha

Another Recurrence

- What about HighArray?
- How many comparisons does HighArray take?
  - T(2n) = ___ T(n) ___
  - Solve it using guess & check (since you already know the answer)

Comparing Algorithms

- If we know the exact problem size n
  - Can directly compare $T_1(n)$ and $T_2(n)$ for different algorithms
- How do we compare algorithms more abstractly?
  - Different "classes" of algorithms
  - Grow bigger in n in different ways
  - Don't care so much about specific ratio with fixed problem size

Some Classes of Algorithms

- Examples:
  - *Linear* – doubling the n makes it twice as big
    - Searching unordered array (average or worst case)
  - *Constant* – changing n doesn't make it change
    - Inserting into an array without holes
- Our latest class:
  - *Logarithmic* – doubling n makes it one bigger

Ignoring the Constant

- All linear algorithms have a formula like:
  - T(n) = K n
  - Average number of comparisons to find an element in an unordered array: 1/2 n
  - Worst case number of comparisons to find an element in an unordered array: n
- Key idea of Big O notation
  - ***Ignore the constant***
  - All linear algorithms are in O(n)
  - O(*something*) is a set of functions that grow as n gets bigger like *something*

Big O

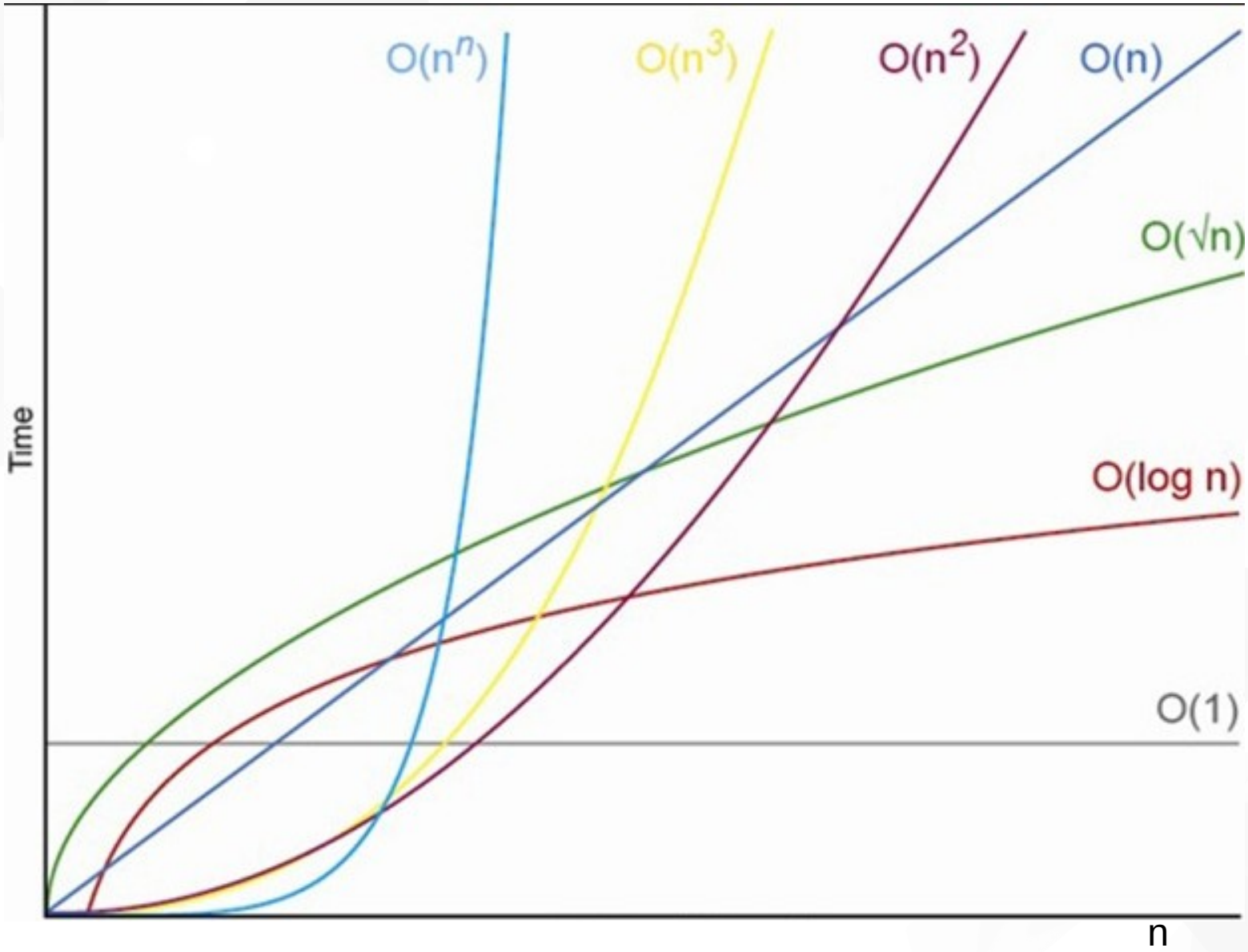
- Some algorithms and running times
  - Linear search — O(n)
  - Binary search — O(log n)
  - Insert into unordered array — O(1)
  - Insert into ordered array — O(N)
  - Delete in unordered array — O(N)
  - Delete in ordered array — O(N)

Simplifying into Big O

- Suppose an algorithm always takes 3 comparisons no matter the size of the input.
  - T(n) = 3 O(3) O(1)
- Suppose an algorithm takes n comparisons to scan through an entire array, then does another n comparisons to scan it *again*
  - T(n) = 2n O(2n) O(n)

ignore any multiplicative constant

Complexities Graphed



Cheatsheet

Data Structure	Time Complexity					
	Average				Worst	
	Indexing	Search	Insertion	Deletion	Indexing	Search
Basic Array	O(1)	O(n)	-	-	O(1)	O(n)
Dynamic Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)

http://bigocheatsheet.com/

A Business Problem

- Suppose we have a text file
  - One line per business
  - Business name, colon, phone number
- Need a program that:
  - Starts up, reads all businesses
  - Waits for queries that are business names
  - Prints out phone number for matching business
  - OR shows "NOT FOUND" if no business found

- *Which data structure should we use?*
  - How do we decide?
  - Does it even matter?

Simplest Thing

- Start with simplest thing
  - Put everything in an array

```
class Record {
    public String name;
    public String number;
}

class BusinessDatabase {

    // Database of all businesses
    private Record[] businesses;

    public Record parse(String txt) {
        // Parsing code here
        ...
    }

    // Parse the input lines
    // Assuming one business per line
    public void parseData(String[] lines) {
        int numBusinesses = lines.length;
        businesses = new Record[numBusinesses];

        for(int i = 0; i < lines.length; i++) {
            Record rec = parse(lines[i]);
            businesses[i] = rec;
        }
    }
}
```

What about search?

- To find a business:

```
// Search for a business
public String getPhoneNumber(String name) {
    for (int i = 0; i < businesses.length; i++) {
        if (name.equals(businesses[i].name)) {
            return businesses[i].number;
        }
    }
    return “NOT FOUND”;
}
```

How many comparisons does it take to find a business if there are 1,000,000?

Which is better for problem?

- Decide between unordered and ordered array
  - Once data is all loaded and processed, which is better for searching?
  - For small numbers of businesses, probably doesn't matter
  - Logarithmic growth is slower as n increases
  - For 1,000,000 businesses, go with ordered array
- But how do we get the data into order?
  - Inserting each business would take O(n), for each business...

how do we order the data?

Simple Sorting

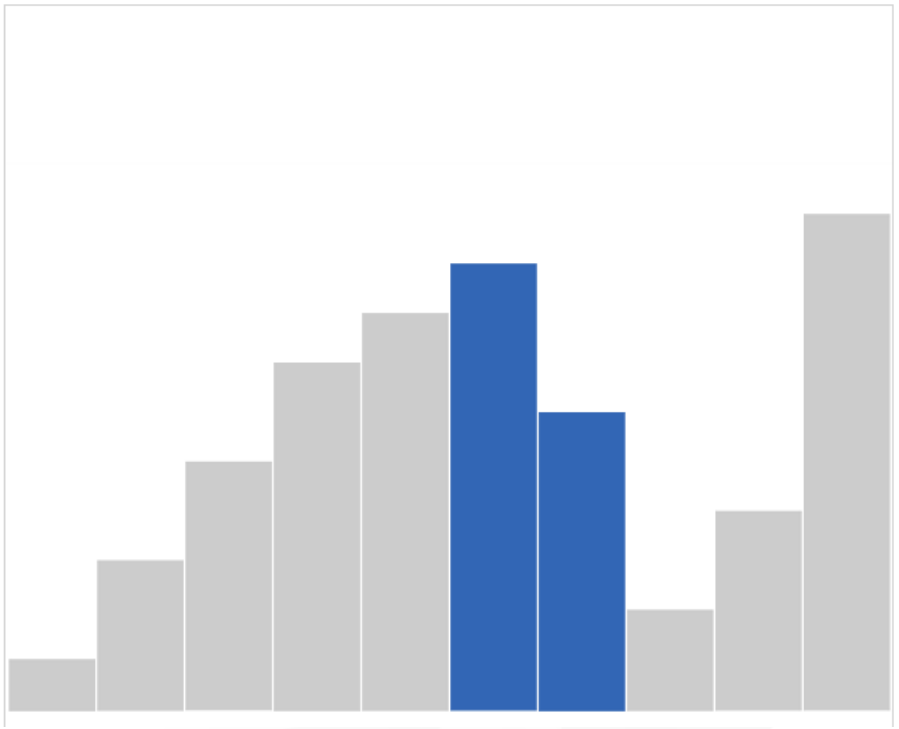
- Well if you're sure, better be... GRYFFINDOR!

Sort Problem

- Suppose you had an unordered array, and wanted an ordered array
- Operations allowed:
  - Compare two items
  - Swap two items, or copy one item to a temp location
- All sorting algorithms do the same basic things
  - Produce ordered output using compares/swaps
  - Difference in the details

Bubble Sort

- Scan through left to right
  - Compare adjacent entries
  - If they are wrong way around, swap them
- If any swaps were made, loop back again
- No swaps made means you're done



http://www.algomation.com/player?algorithm=54162b10f3166302000a91f2

Bubble Sort in Java

- Examples/Chap03/BubbleSort/bubbleSort.java

- Observation:
  - Each run through, we know we have the biggest element to the right
  - So stop scanning once we hit the finished section on the right

Running Time of BubbleSort

- How many comparisons does optimized bubble sort make in the worst case?
  - For n=10:
    - Our variation: 9+9+9+9+9+9+9+9+9
    - With observation: 9+8+7+6+5+4+3+2+1

The End