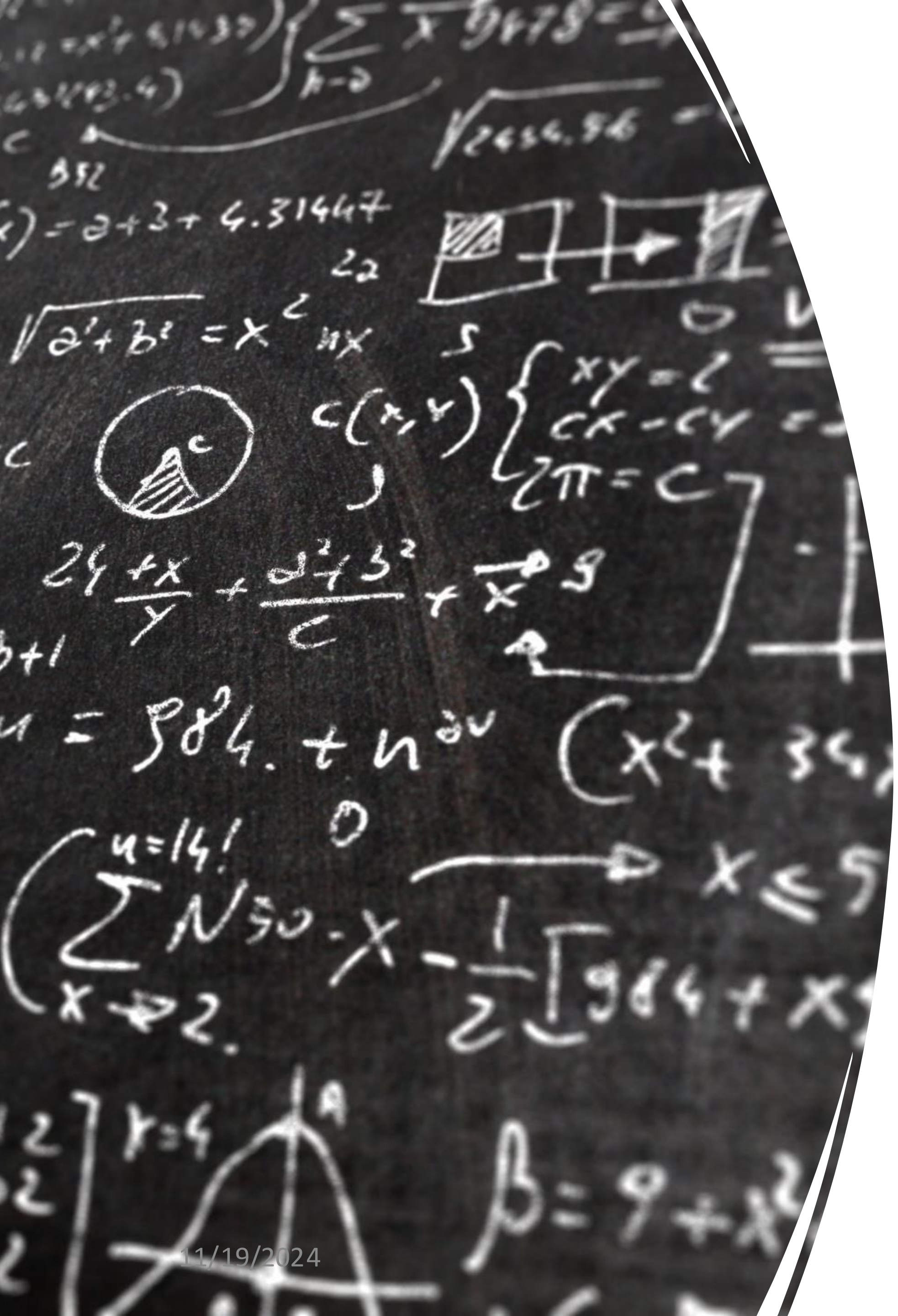

CS 113 – Computer Science I

Lecture 19 – Searching

Tuesday 11/19/2024

Announcements

HW 09 – Due Monday 11/18

Autograder is up

HW 10 – Due Monday 11/25

Object Oriented Programming:

Inheritance & Interface

Releasing it tomorrow

Mid-semester feedback form: https://forms.gle/Ed7G9oe74QQBT5sy5

Midterm 2: Thursday December 5th

Inheritance vs Interfaces

Each of these lines is related to either interfaces or inheritance...

- `extends` keyword
- guarantees a class has implemented certain methods
- `implements` keyword
- reuses implementations
- *is-a* relationship
- specifies what a class *does*

Interfaces

- For any new shape that is created, we want to **enforce** that these methods are also implemented.

```
interface Shape {
    public double getArea();
    public String getName();
}
```

```
class Circle implements Shape {
```

Interfaces

A contract - A set of shared methods that users **must** implement

A collection of method signatures with no bodies

A class can implement more than one interface

Interfaces

An interface is not a class!

A class is what an object **<u>is</u>**

An interface is what an object **<u>does</u>**

- can not be instantiated
- no constructors
- incomplete methods

Interface

No modifier - implicitly `public`

No instance variables except for constants (`static final`)

Searching

Finding whether an item is in a collection

Applications:

- Specific email in an inbox
- Word in a document
- Course in a list of course offerings
- Professor is on RateMyProfessor
- …

Common search problems

Is an item in an array?

- Returns: True or False

Where in an array is the item?

- Returns: the index (an integer)
    - Standard: -1 if the item is not found

How many times does the item appear in an array?

- Returns: a count (an integer)

What is the min, max, or average value in an array?

- Returns: the value (double)

Searching in Bank

Do we have an account for a specific person/name

Idea:

Iterate through each BankAccount in the array

Check if the current bank account's name is the same as the name we are searching for.

If yes:

return True;

return False;

Searching in an array of Animals

Does our collection contain a specific Animal?

Idea:

Iterate through each Animal in the array

Check if the current animal is the same as the animal we are searching for.

If yes:

return True;

return False;

Comparing objects

Recall: variables for objects are references (pointers) to objects

==

compares whether the two references are the same

Object.equals(Object obj)

compares two objects

Every (base) class should implement this

Searching in our array of Animals

Where in our collection is the specific Animal?

Idea:

Iterate through each Animal in the array

Check if the current point is the same as the animal we are searching for.

If yes:

return the index;

return False;

Searching in our array of Animals

Where in our collection is the specific Animal?

Idea:

Iterate through each Animal in the array

Check if the current point is the same as the animal we are searching for.

If yes:

return the index;

return -1;

Searching in our array of Animals

How many animals in our collection are less than 5lbs?

Linear Search

These previous approaches are examples of linear search

Check each item in a collection one by one

Why is this call linear search?

Time it takes to search increases *linearly* with the size of the list

Linear Search

What happens (in terms of speed) when the list is very large?

The search becomes slower

In what cases do we do the most work (i.e. perform the most comparisons)?

When the item is not in the list

In what cases do we do the least amount of work?

When the item is the first element in the list

Binary Search

If we could change the list, is there a way to search more efficiently?

Yes, if the list is sorted

Guessing game – in class exercise

Pair up:

- Person A chooses a number between 1 and 100
- Person B guesses the number
- Until the guess is correct:
    - Person A tells whether the guess is too high or too low
    - Person B guesses again

Binary Search

Assuming list is sorted in ascending order

High-level Algorithm:

- Step 1: Find the midpoint of the list:
  - if the search value is at the midpoint – we are done!
  - if the value we are searching for is above the midpoint,
    - Search right: cut our list in half and repeat step 1 with the right half of the list
  - If the value we are searching for is below the midpoint
    - Search left: cut out list in half and repeat step 1 with the left half of the list

Binary Search – Initial Values

lowIndex, highIndex, midIndex

lowIndex = 0

highIndex = length of the array – 1

midIndex = $\frac{lowIndex+highIndex}{2}$

Binary Search – Initial Values

lowIndex, highIndex, midIndex

If value at midIndex== searchValue:
    Success!

If value at midIndex < searchValue:
    lowIndex = midIndex + 1
    update midIndex

If value at midIndex > searchValue:
    highIndex = midIndex – 1
    update midIndex

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 99

low	mid	high	ls[mid]

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 99

low	mid	high	ls[mid]
0	2	5	44

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 99

low	mid	high	ls[mid]
0	2	5	44
3	4	5	99 (found!)

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 30

low	mid	high	ls[mid]

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 30

low	mid	high	ls[mid]
0	2	5	44

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 30

low	mid	high	ls[mid]
0	2	5	44
0	0	1	-20

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 30

low	mid	high	ls[mid]
0	2	5	44
0	0	1	-20
1	1	1	-4

Binary search

0 1 2 3 4 5

String[] ls = {-20, -4, 44, 58, 99, 145}

Search for 30

low	mid	high	ls[mid]
0	2	5	44
0	0	1	-20
1	1	1	-4
2		1	Not found!

Binary search w/ Strings

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “cow”

low	mid	high	ls[mid]

Binary search w/ Strings

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “cow”

low	mid	high	ls[mid]
0	3	7	“cat”

Binary search w/ Strings

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “cow”

low	mid	high	ls[mid]
0	3	7	“cat”
4	5	7	“dog”

Binary search w/ Strings

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “cow”

low	mid	high	ls[mid]
0	3	7	“cat”
4	5	7	“dog”
4	4	4	“cow”!

Binary search

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “elephant”

low	mid	high	ls[mid]

Binary search

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “elephant”

low	mid	high	ls[mid]
0	3	7	“cat”

Binary search

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “elephant”

low	mid	high	ls[mid]
0	3	7	“cat”
4	5	7	“dog”

Binary search

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “elephant”

low	mid	high	ls[mid]
0	3	7	“cat”
4	5	7	“dog”
6	6	7	“fish”

Binary search

0 1 2 3 4 5 6 7

String[] ls = {"bear", "bird", "bug", "cat", "cow", "dog", "fish, "lion“};

Search for “elephant”

low	mid	high	ls[mid]
0	3	7	“cat”
4	5	7	“dog”
6	6	7	“fish”
6		6	

Binary Search

If the list is sorted in ascending order, we don’t need to consider every element.

Which element should we check?

The middle

If the middle element isnt what we are looking for, what should we do?

Chop the search space in half (this is why its called ***<u>binary</u>*** search)

Binary Search run time

As the size of our collection increases, the number of guesses/comparisons increases, but not *linearly*

The time increases by $\log n$ $(\text{we use base } 2)$. Why?

Because we cut our search space in half each time

If our collection contains 8 data points, how many comparisons in worst case do we make:

$$\log_2 8 = 3$$

If our collection contains 512 data points, how many comparisons in worst case do we make:

$$\log_2 512 = 9$$