Intro to Searching

Algorithm

- What is an algorithm, again?

Algorithm

- What is an algorithm, again?
  - Describes the process of how we solve a problem
  - Is code-neutral

Searching Algorithms

- Many objects already provide searching methods for us
  - Good example of encapsulation
  - myList.index(4)
- But how do those functions work?
- Let’s write some code in basic python to search a list without methods

```
def basicSearch(lyst,what):
    for item in lyst:
        if item == what:
            return True
    return False
```

```
def linearSearch(lyst,what):
    for index in range(len(lyst)):
        if lyst[index] == what:
            return index
    return -1
```

Big “O” Notation

- Can we improve our search?
- Big “O” notation is analysis on how long our algorithm takes as our data set grows
  - Typically asking for two values:
    - On average
    - Worst case scenario

Big “O” Notation

- Based on the functions we developed
  - Worst case scenario, how many things do we need to look at to find our number in the list?
  - On average, how many things do we need to look at to find our number in the list?

- What if our list were of size 20?

Big "O" Notation

- Based on the functions we developed
  - Worst case scenario, how many things do we need to look at to find our number in the list?
    - 20
  - On average, how many things do we need to look at to find our number in the list?
    - 10

Big “O” Notation

- Based on the functions we developed
  - Worst case scenario, how many things do we need to look at to find our number in the list?
    - N, where N is the number of items in the list
  - On average, how many things do we need to look at to find our number in the list?
    - N/2

Linear Search

- Our previous algorithms make one comparison for every item as worst case.
- Also called a ***linear search***, because the number of comparisons scales as the amount of items in the list increases
- Double the number of items in list, double the amount of time needed to complete the search in the worst case
- Can we do better than a linear search
  - Less comparisons, even on worst case?

Optimized Searching

- Have you ever played the higher or lower game?
  - Think of a number
  - As the player guesses the number, you say “higher” or “lower” until the player finally guesses the correct number

Optimized Searching

- One good strategy if you are the guesser is to
  - Guess the middle number of the range
  - If the person says "higher", then adjust your low range bound to be your guess+1
  - If the person says "lower", then adjust your high range bound to be your guess-1
  - Repeat

Optimized Searching

- Same idea if you are looking up a vocabulary term in a dictionary
- You will open the book, look at the current word, and figure out if you should search lower or higher
- We might as well use this kind of additional information to optimize our searching

Binary Search

- We can use this type of search on our list
- Does our list have to be sorted for this to work?
- Say that we have a list of 20 items
  - What is the worst case number of comparisons?
  - What about a list of 40 items?

Binary Search

- Every time I double the number of items in my list, my search complexity only goes up by 1
  - Is much better than linear time as number of items in the list goes up
- Let's write a binary search.

Binary Search

- Binary search algorithm
  - Try the guess at middle index of the range
  - If the value we are searching for is higher than number at the index, then adjust your low range bound to be your guess+1
  - If the value we are searching for is lower than number at the index, then adjust your high range bound to be your guess-1
  - Repeat

```
def binarySearch(lyst,what):
    lowIndex = 0
    highIndex = len(lyst) - 1


    while lowIndex <= highIndex  :
        middle = (lowIndex + highIndex) // 2

        if lyst[middle] == what:
            return middle
        if lyst[middle] > what:
            highIndex = middle - 1
        if lyst[middle] < what:
            lowIndex = middle + 1

    return -1
```

Binary Search

- What is the worst-case scenario of the binary search?
- Thinking of a number between 1 and 100
  - 7 guesses in total – why?
    - 1 guesses – cut down to 50 possibilities
    - 2 guesses – cut down to 25
    - 3 guesses – cut down to 12
    - 4 guesses – cut down to 6
    - 5 guesses – cut down to 3
    - 6 guesses – cut down to 1
    - 7 guesses – to figure out if last guess is right

Binary Search

- What is the complexity of a binary search?
  - Big O value of $\log_2 N$
  - This is "log base 2"
- $\log_2(100) = x$
  - What is this saying?

Binary Search

- What is the complexity of a binary search?
  - Big O value of $\log_2 N$
  - This is "log base 2"
- $\log_2(100) = x$
  - What is this saying?
  - $2^x = 100$
  - Go "to the next power" when not exact

Binary Search

- How does that relate to our binary search?
  - Let's say there are 16 items in our list. What is the worst case number of guesses?

Binary Search

- How does that relate to our binary search?
  - Let's say there are 16 items in our list. What is the worst case number of guesses?
  - It takes at most 20 guesses to find an item in a 1,000,000 item list:
  - 2^10 = 1024 (10 guesses to find an item in 1000 items)
  - One million is 1000 squared, so twice as much