Week 5 Workshop

Python Fundamentals, Data Structures, and Algorithms

Workshop Agenda

Activity	Estimated Duration
Welcome and check in	10 mins
Week 5 Review	50 mins
Portfolio Project Show & Tell	50 mins
Break	15 mins
Workshop Assignment	100 mins
Check-Out (Feedback & Wrap-Up)	15 mins

Week 5 Review

Overview

Algorithms	Bubble Sort
Linear Search	Quicksort
Binary Search	Big O Notation

Review: Algorithms

Algorithm Techniques

Brute Force:

Simple to understand

Trial and Error

Inefficient

Resource Intensive

Divide & Conquer:

More sophisticated

Divides the problem into sub-problems

Combines sub-problems to solve

Review: Algorithms

Classifying Algorithms

- Algorithms can be classified by what they “do” or how they work.
- The two primary classifications or types of algorithms are Search and Sort.

Review: Linear Search

- What is one advantage and one disadvantage of Linear Search?

- PRO: Can be used on any data set, sorted or unsorted
- CON: Not the most efficient

```python
mylist = [59, 76, 60, 13, 11, 83, 48, 52, 54, 41, 97, 76, 33, 9, 67]
findval = 41
cnt = 0
for item in mylist:
    cnt += 1
    if item == findval:
        print('Found', findval, 'in', cnt, 'steps')
        break
```

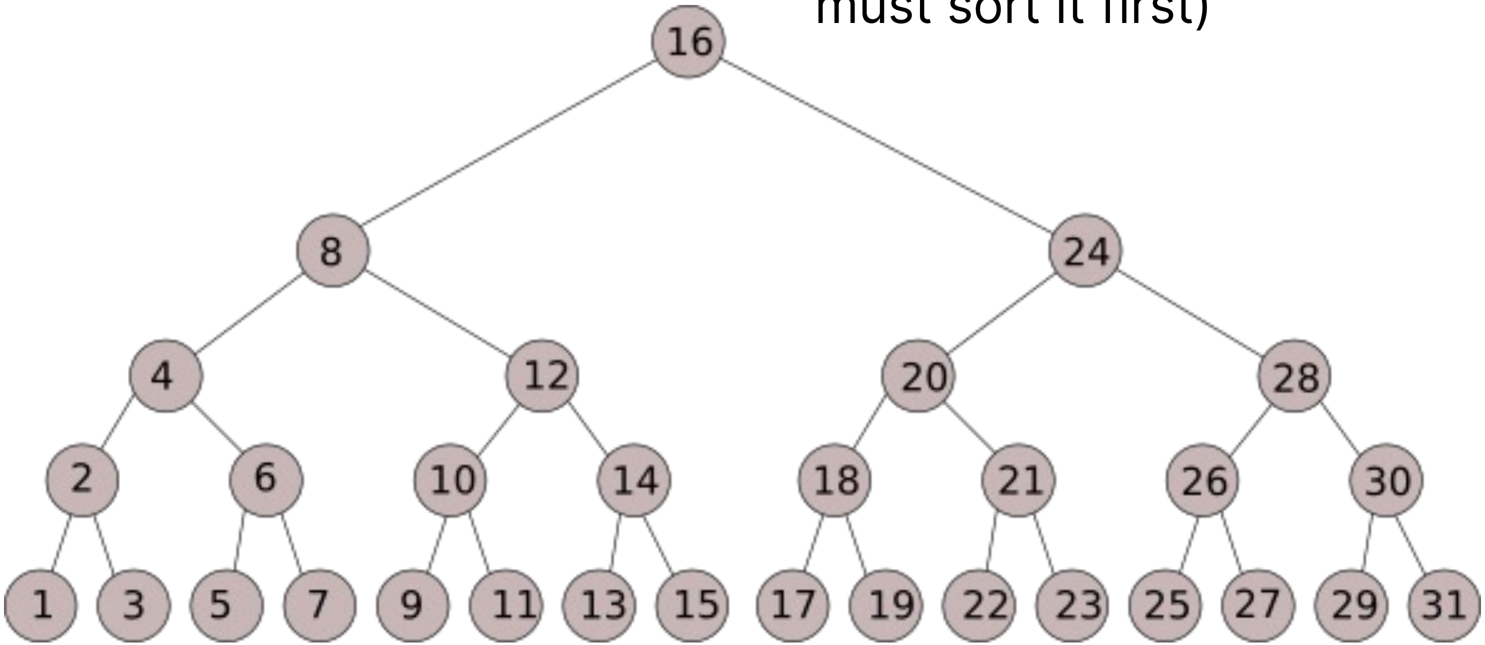
```
$ python linearsearch.py
```

```
Found 41 in 10 steps
```

Review: Binary Search

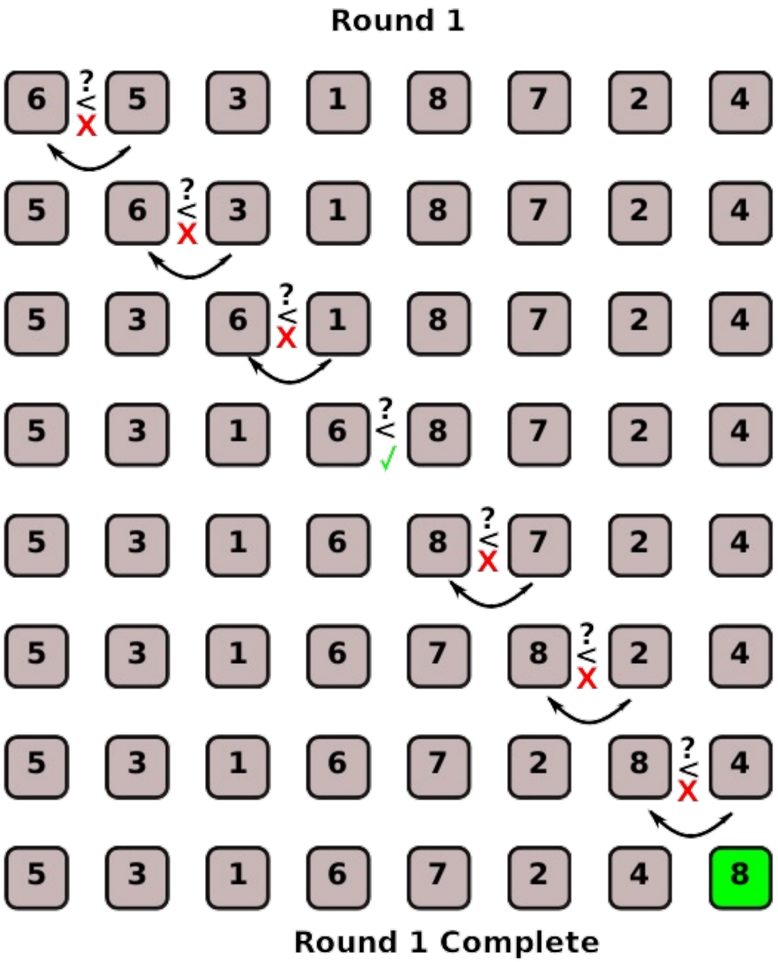
- What is one advantage and one disadvantage of Binary Search?

- **PRO**: More efficient algorithm
- **CON**: Only works on sorted data (if data is unsorted, you must sort it first)



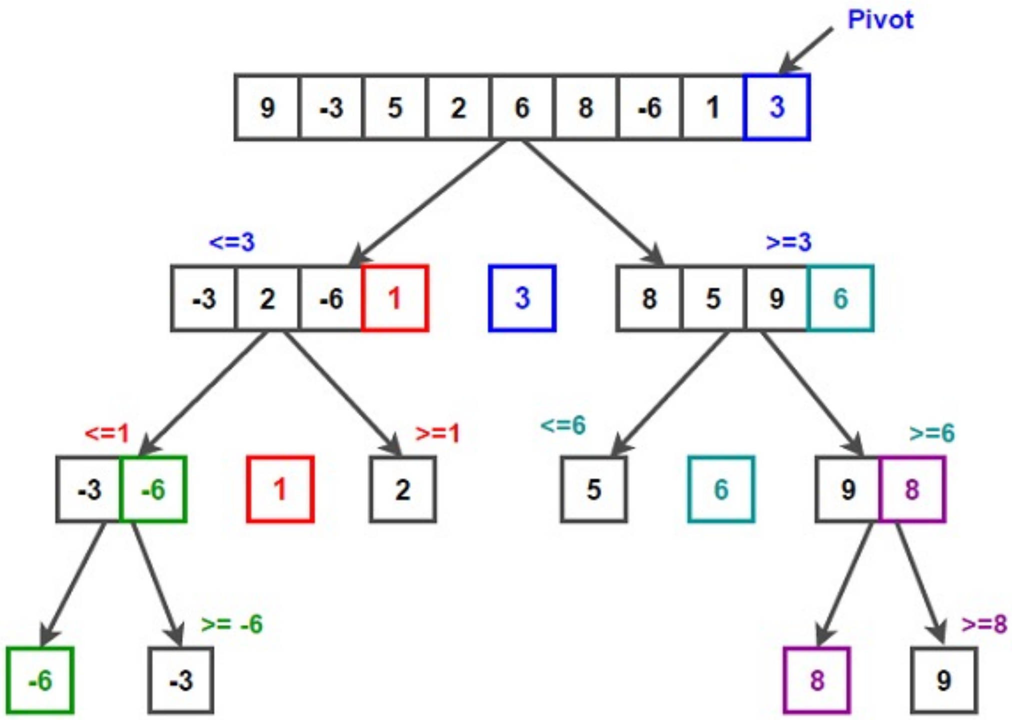
Review: Bubble Sort

- Implements brute force technique, great as a teaching tool but not an efficient algorithm generally
- **Round 1:**
  - Move from one end of the set to the other (8x)
  - Compare two numbers
  - If left is greater than right, swap
  - Move to the next pair
  - Repeat step 1 until the end
- After Round 1, the highest value will have "bubbled" to the top (or end) of the list.
- Now repeat the steps above 7 more times.



Review: Quicksort

- Quicksort uses divide & conquer; it is considered a highly efficient algorithm
- Both Bubble Sort and Quicksort are in-place/sort-in-place algorithms, meaning the values are moved within the list to a new position (instead of to a separate temporary list)



Review: Big O Notation

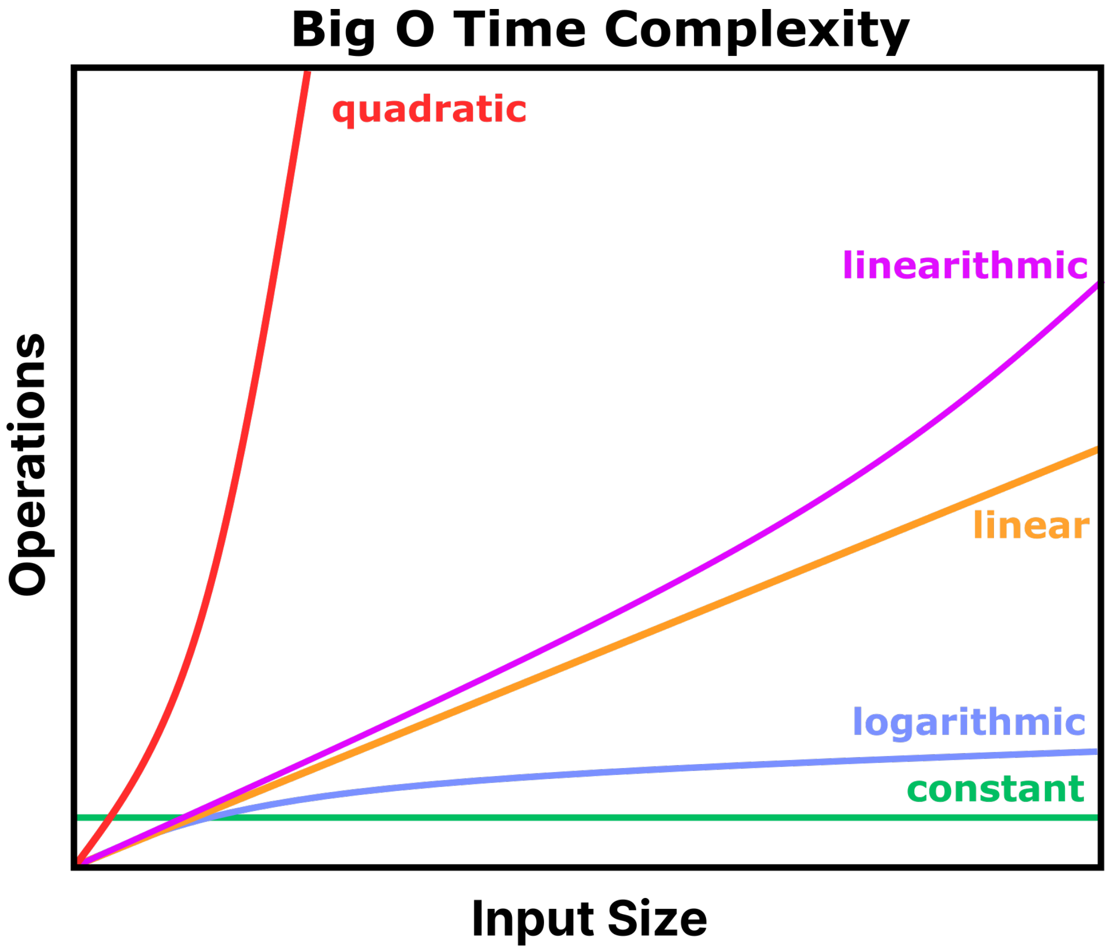
Big O Notation is an expression of the **Time Complexity** of an algorithm.

Name	Time Complexity
Constant Time	$O(1)$
Logarithmic Time	$O(\log n)$
Linear Time	$O(n)$
Linearithmic Time	$O(n \log n)$
Quadratic Time	$O(n^2)$
Exponential Time	$O(2^n)$
Factorial Time	$O(n!)$

Review: Time Complexities

The efficiency of the algorithm found in estimating the number of operations it will take to execute based on the increasing size of the input (Input Size)

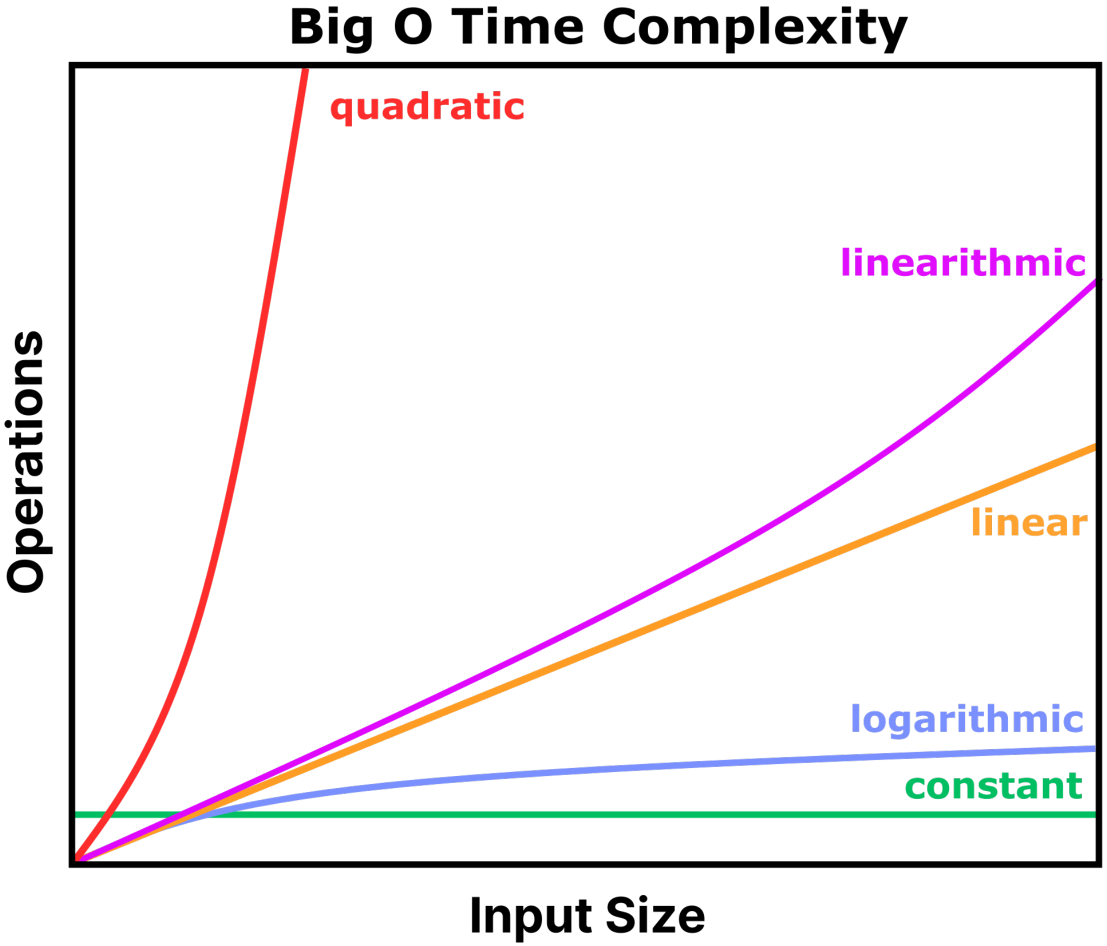
In general, the Time Complexity gives the worst-case scenario.



Review: Time Complexities

Discussion:

1. What other cases might be possible to determine other than worst-case?
   - average-case and best-case
2. And what would cause the complexity to NOT be the worst case?
   - if we can limit the practical values of the input in some way



Review: Constant Time

- A Constant Time algorithm is not dependent on the size of the input data.
- It will always run for the same amount of time regardless of the input size.

```python
def const(n):
    if n < 100:
        return True
    else:
        return False
```

Review: Logarithmic Time

- A **Logarithmic Time** algorithm will increase the time of execution directly proportional to the input size
- In a binary logarithm, each time the value of x doubles, the value of y only increases by 1 (or another constant number).

```python
from random import randint

def bin_search(l, val):
    start = 0
    end = len(l)-1
    match = False
    cnt = 0

    while start <= end and not match:
        cnt += 1
        middle = (start+end)//2
        if l[middle] == val:
            match = True
        else:
            if val < l[middle]:
                end = middle-1
            else:
                start = middle+1
    return match, cnt
```

```python
for i in range(10):
    # make a random list of integers using a sorted list comprehension
    listsize = 100000
    randlist = [randint(1, listsize) for n in range(listsize)]
    randlist.sort()
    print('Random List of', listsize)
    # print(randlist)

    # pick a random value and then try to find it in the list
    pickval = randlist[randint(0, listsize-1)]
    found, cnt = bin_search(randlist, pickval)
    print(
        f'Searching {cnt} times I {("found" if found else "did not find")} the value {pickval}')
```

Review: Logarithmic Time

Notice the average number of searches does increase, but at a much slower rate than the input size.

Input Size: 20

```
Random List of 20
Searching 2 times I found the value 11
Random List of 20
Searching 2 times I found the value 14
Random List of 20
Searching 2 times I found the value 7
Random List of 20
Searching 5 times I found the value 10
Random List of 20
Searching 3 times I found the value 8
Random List of 20
Searching 3 times I found the value 18
Random List of 20
Searching 5 times I found the value 17
Random List of 20
Searching 1 times I found the value 8
Random List of 20
Searching 5 times I found the value 20
Random List of 20
Searching 2 times I found the value 13
```

Input Size: 1,000

```
Random List of 1000
Searching 6 times I found the value 156
Random List of 1000
Searching 7 times I found the value 792
Random List of 1000
Searching 7 times I found the value 135
Random List of 1000
Searching 8 times I found the value 141
Random List of 1000
Searching 9 times I found the value 979
Random List of 1000
Searching 7 times I found the value 322
Random List of 1000
Searching 8 times I found the value 175
Random List of 1000
Searching 10 times I found the value 210
Random List of 1000
Searching 8 times I found the value 344
Random List of 1000
Searching 9 times I found the value 828
```

Input Size: 100,000

```
Random List of 100000
Searching 14 times I found the value 29593
Random List of 100000
Searching 16 times I found the value 71367
Random List of 100000
Searching 15 times I found the value 93967
Random List of 100000
Searching 14 times I found the value 89221
Random List of 100000
Searching 17 times I found the value 56427
Random List of 100000
Searching 15 times I found the value 10088
Random List of 100000
Searching 16 times I found the value 30155
Random List of 100000
Searching 13 times I found the value 7164
Random List of 100000
Searching 16 times I found the value 11498
Random List of 100000
Searching 15 times I found the value 64177
```

Review: Linear Time

- A **Linear Time** algorithm will increase the time of execution directly proportional to the input size.

```python
def sumall(n):
    tot = 0
    for i in range(1, n):
        tot += i
    return tot
```

Review: Linearithmic Time

- A **Linearithmic Time** algorithm will increase the number of operations by the input size n times log n.
- This algorithm works in a linear fashion, but as each iteration occurs, the set of remaining values is reduced by the work previously completed.
- Thus, Linearithmic is a bit more inefficient than Linear algorithms.
- The Quicksort algorithm is a great example of this – the pivot value must be linearly compared with each element in its list, which is then halved in a recursion, down to list size 1.

Review: Quadratic Time

- A **Quadratic Time** algorithm increases the number of operations by the square of the input size.

```python
def qtime(n):
    cnt = 0
    for i in range(n):
        for j in range(n):
            cnt += 1
    return cnt


c = qtime(2)
print('Total Operations:', c)
c = qtime(4)
print('Total Operations:', c)
c = qtime(6)
print('Total Operations:', c)
c = qtime(8)
print('Total Operations:', c)
c = qtime(10)
print('Total Operations:', c)
```

```
Total Operations: 4
Total Operations: 16
Total Operations: 36
Total Operations: 64
Total Operations: 100
```

Portfolio Projects: Show & Tell

- What did you make?
- Take turns briefly describing and showing off your Portfolio Project.

Workshop 5 Assignment

Goal: Code a number-guessing game, 3 ways

- **Task 1:** Write a number guessing game where the user gets to guess.
- **Task 2:** Write a number guessing game and have the program guess, using linear search
- **Task 3:** Write a number guessing game and have the program guess, using binary search

- You will be split up into groups to work on the assignment together.
- Talk through each step out loud with each other, code collaboratively.
- If your team spends more than 10 minutes trying to solve one problem, ask your instructor for help!