Name:_______________________________ **Partner:** _________________________

Python Activity 50: Iterative Search

Search is very central to how we use computers.

Learning Objectives
Students will be able to:
Content:
- Identify ***best case*** and ***worst case*** scenarios for searching algorithms
- Predict how changes in a ***searching*** algorithm impacts efficiency
- Describe the ***linear*** and ***binary searching*** algorithms for sorted vs. unsorted data

Process:
- Write code that implements ***linear search*** and ***binary search*** ***iteratively***

Prior Knowledge
- Python concepts: computational thinking, lists, functions, while loops, conditionals

Concept Model:

CM1. List examples of when you *search*: ____

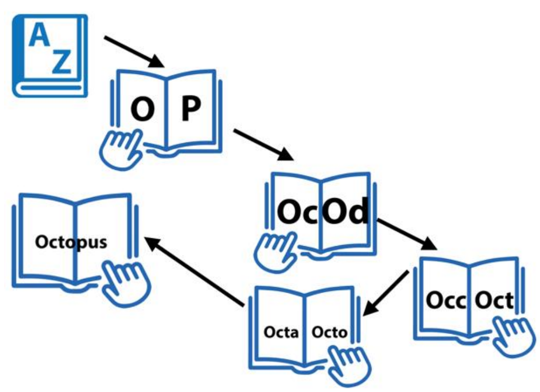
____

What would happen if any of these search activities took twice as long as you expected?

____

CM2. The text and diagram below represent **two** approaches to finding the word "octopus" in a physical, paper dictionary.

Finding a Word in a Dictionary – Two Ways	

For each page in our dictionary book:
    Check to see if our word is on that page
    If it is, then we've found the word!
    If it isn't, then turn the page.
	

a. What might be the *best case* for the approach on the left? ______________________________
What might be the *worst case* for the approach on the left? _________________________

b. Is the approach on the left how you typically find a word in a physical dictionary? ___________
What is your typical approach?

____

____

Is your approach more efficient than the one described on the left? ____________
What might be the *best case* for your approach? _______________________________
What might be the *worst case* for your approach? _________________________

c. Which of these approaches would work better for finding a word in an *unsorted* order? Why?

____

____

FYI: A ***best case*** scenario is when the minimum number of operations is required (i.e., when an approach will take the fewest number of steps). A ***worst case*** scenario is when the maximum number of operations is required (i.e., most number of operations over all possible inputs). An ***average case*** scenario is when the average/typical number of operations is required.

Critical Thinking Questions:

1. Examine the following partially complete code for *searching* for an item in a list:

linear.py

```
def linear_search(mylist, item):
    # (i) for each item in our list


        # (ii) check to see if it's our item and…?


    # (iii) otherwise…return False
```

a. Complete the code above where the comments scaffold a linear search of a list.

b. Which searching algorithm is this most similar to from CM2? ____________________

c. What is the *best* case scenario for this algorithm? ________________________________

d. What is the *worst* case scenario for this algorithm? _______________________________

2. Examine the following partially complete code for *searching* for an item in a *sorted* list:

binary.py

```
def binary_search(target_list, item):
    # initialize vars determining what portion of the list we look at
    left_index = 0
    right_index = len(target_list) - 1

    # search until we've exhausted all relevant halves of the list
    while left_index <= right_index:

        mid_index = (left_index + right_index) // 2
        if item == target_list[mid_index]:
            return True
        # case where the item may be in the left half of the list
        if item < target_list[mid_index]:
            right_index = mid_index - 1
        # case where the may be in the right half of the list
        else:
            # (iv) what should be here?

    # if we're here, we haven't found the element!
    return False
```

a. Step through the code, and explain what the following sections do:

`def binary_search(target_list, item):`	

`left_index = 0`
 	
`right_index = len(target_list) - 1`	

`while left_index <= right_index:`	
`mid_index = (left_index + right_index) // 2`	
`if item == target_list[mid_index]:` `return True`	
`if item < target_list[mid_index]:` `right_index = mid_index - 1`	
`(iv) what should be here?`	
`return False`	

b. Which searching algorithm is this most similar to from CM2? ____________________

c. Write one lines of code to complete the `(iv)` comment section:

____

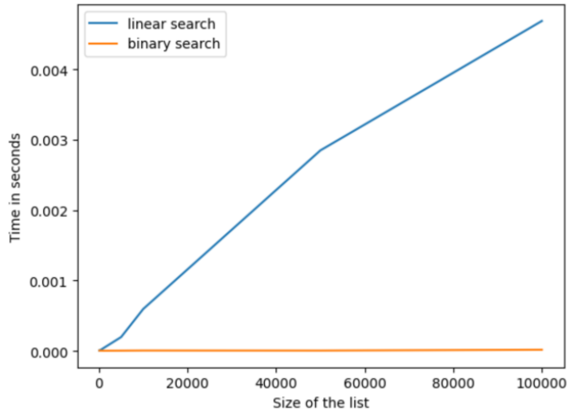
d. What is the *best* case scenario for this algorithm? ________________________________

e. What is the *worst* case scenario for this algorithm? ________________________________

f. Will this code work on an *unsorted* list? Why or why not?

____

3. When we compare the run-times of these two algorithms, and plot them with the number of elements on the X-axis and time on the Y-axis, we see the following chart:

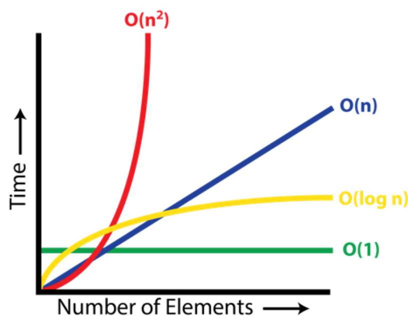


a. According to the graph above, which Search Algorithm is faster? ______________________

b. Which search algorithm would be faster for *unsorted* data? ______________________

c. Which search algorithm might be better for small datasets? ______________________

d. If you had to fit the empirical runtimes above to a more generalized runtime plot from the ones shown below, what would you you choose?



Linear Search: $O(n^2)$ or $O(n)$ or $O(\log n)$ or $O(1)$?

Binary Search: $O(n^2)$ or $O(n)$ or $O(\log n)$ or $O(1)$?