**Practice Quiz: Binary Searching a Problem**

**TOTAL POINTS 5**

1.Question 1

You have a list of computers that a script connects to in order to gather SNMP traffic and calculate an average for a set of metrics. The script is now failing, and you do not know which remote computer is the problem. How would you troubleshoot this issue using the bisecting methodology?

1 point



Run the script with the first half of the computers.



Run the script with last computer on the list.



Run the script with first computer on the list



Run the script with two-thirds of the computers.

2.Question 2

The find\_item function uses binary search to recursively locate an item in the list, returning True if found, False otherwise. Something is missing from this function. Can you spot what it is and fix it? Add debug lines where appropriate, to help narrow down the problem.

def find\_item(list, item):

  #Returns True if the item is in the list, False if not.

  if len(list) == 0:

    return False

  #Is the item in the center of the list?

  middle = len(list)//2

  if list[middle] == item:

    return True

  #Is the item in the first half of the list?

  if item < list[middle]:

    #Call the function with the first half of the list

    return find\_item(list[:middle], item)

  else:

    #Call the function with the second half of the list

    return find\_item(list[middle+1:], item)

  return False

#Do not edit below this line - This code helps check your work!

list\_of\_names = ["Parker", "Drew", "Cameron", "Logan", "Alex", "Chris", "Terry", "Jamie", "Jordan", "Taylor"]

print(find\_item(list\_of\_names, "Alex")) # True

print(find\_item(list\_of\_names, "Andrew")) # False

print(find\_item(list\_of\_names, "Drew")) # True

print(find\_item(list\_of\_names, "Jared")) # False

Solution:

3.Question 3

The binary\_search function returns the position of key in the list if found, or -1 if not found. We want to make sure that it's working correctly, so we need to place debugging lines to let us know each time that the list is cut in half, whether we're on the left or the right. Nothing needs to be printed when the key has been located.

For example, binary\_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 3) first determines that the key, 3, is in the left half of the list, and prints "Checking the left side", then determines that it's in the right half of the new list and prints "Checking the right side", before returning the value of 2, which is the position of the key in the list.

Add commands to the code, to print out "Checking the left side" or "Checking the right side", in the appropriate places.

def binary\_search(list, key):

    #Returns the position of key in the list if found, -1 otherwise.

    #List must be sorted:

    list.sort()

    left = 0

    right = len(list) - 1

    while left <= right:

        middle = (left + right) // 2

        if list[middle] == key:

            return middle

        if list[middle] > key:

            right = middle - 1

        if list[middle] < key:

            left = middle + 1

    return -1

print(binary\_search([10, 2, 9, 6, 7, 1, 5, 3, 4, 8], 1))

"""Should print 2 debug lines and the return value:

Checking the left side

Checking the left side

0

"""

print(binary\_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 5))

"""Should print no debug lines, as it's located immediately:

4

"""

print(binary\_search([10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 7))

"""Should print 3 debug lines and the return value:

Checking the right side

Checking the left side

Checking the right side

6

"""

print(binary\_search([1, 3, 5, 7, 9, 10, 2, 4, 6, 8], 10))

"""Should print 3 debug lines and the return value:

Checking the right side

Checking the right side

Checking the right side

9

"""

print(binary\_search([5, 1, 8, 2, 4, 10, 7, 6, 3, 9], 11))

"""Should print 4 debug lines and the "not found" value of -1:

Checking the right side

Checking the right side

Checking the right side

Checking the right side

-1

"""

Solution:

4.Question 4

When trying to find an error in a log file or output to the screen, what command can we use to review, say, the first 10 lines?

1 point



wc



tail



head



bisect

5.Question 5

The best\_search function compares linear\_search and binary\_search functions, to locate a key in the list, and returns how many steps each method took, and which one is the best for that situation. The list does not need to be sorted, as the binary\_search function sorts it before proceeding (and uses one step to do so). Here, linear\_search and binary\_search functions both return the number of steps that it took to either locate the key, or determine that it's not in the list. If the number of steps is the same for both methods (including the extra step for sorting in binary\_search), then the result is a tie. Fill in the blanks to make this work.

def linear\_search(list, key):

    #Returns the number of steps to determine if key is in the list

    #Initialize the counter of steps

    steps=0

    for i, item in enumerate(list):

        steps += 1

        if item == key:

            break

    return \_\_\_

def binary\_search(list, key):

    #Returns the number of steps to determine if key is in the list

    #List must be sorted:

    list.sort()

    #The Sort was 1 step, so initialize the counter of steps to 1

    steps=1

    left = 0

    right = len(list) - 1

    while left <= right:

        steps += 1

        middle = (left + right) // 2

        if list[middle] == key:

            break

        if list[middle] > key:

            right = middle - 1

        if list[middle] < key:

            left = middle + 1

    return \_\_\_

def best\_search(list, key):

    steps\_linear = \_\_\_

    steps\_binary = \_\_\_

    results = "Linear: " + str(steps\_linear) + " steps, "

    results += "Binary: " + str(steps\_binary) + " steps. "

    if (\_\_\_):

        results += "Best Search is Linear."

    elif (\_\_\_):

        results += "Best Search is Binary."

    else:

        results += "Result is a Tie."

    return results

print(best\_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 1))

#Should be: Linear: 1 steps, Binary: 4 steps. Best Search is Linear.

print(best\_search([10, 2, 9, 1, 7, 5, 3, 4, 6, 8], 1))

#Should be: Linear: 4 steps, Binary: 4 steps. Result is a Tie.

print(best\_search([10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 7))

#Should be: Linear: 4 steps, Binary: 5 steps. Best Search is Linear.

print(best\_search([1, 3, 5, 7, 9, 10, 2, 4, 6, 8], 10))

#Should be: Linear: 6 steps, Binary: 5 steps. Best Search is Binary.

print(best\_search([5, 1, 8, 2, 4, 10, 7, 6, 3, 9], 11))

#Should be: Linear: 10 steps, Binary: 5 steps. Best Search is Binary.

Solution: