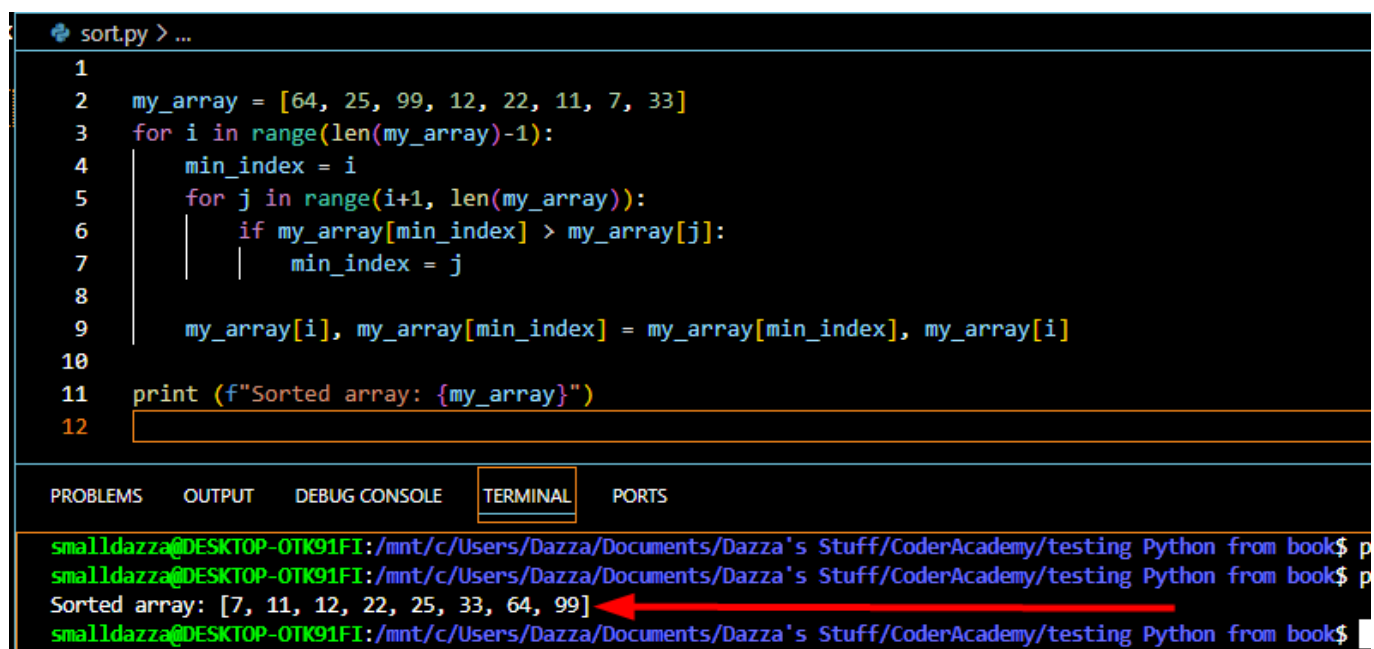


# Workbook DarrenSmall\_T2A1-B

## Question 1 Answer:

Two different types of sorting algorithms are:

1. Selection Sort Algorithm: the process for this algorithm is it assumes the first element in an array is the lowest of all the elements. It will then check all the other elements for any element lower than this assumed lowest element. If it finds another actual element lower than the assumed element, it swaps the position of the assumed for the actual lower element. If in the checking process it finds no other element lower than the assumed, it moves to compare and check the next element position. This process is repeated until all elements have been checked and sorted in the array. Example of selection sort in python code:



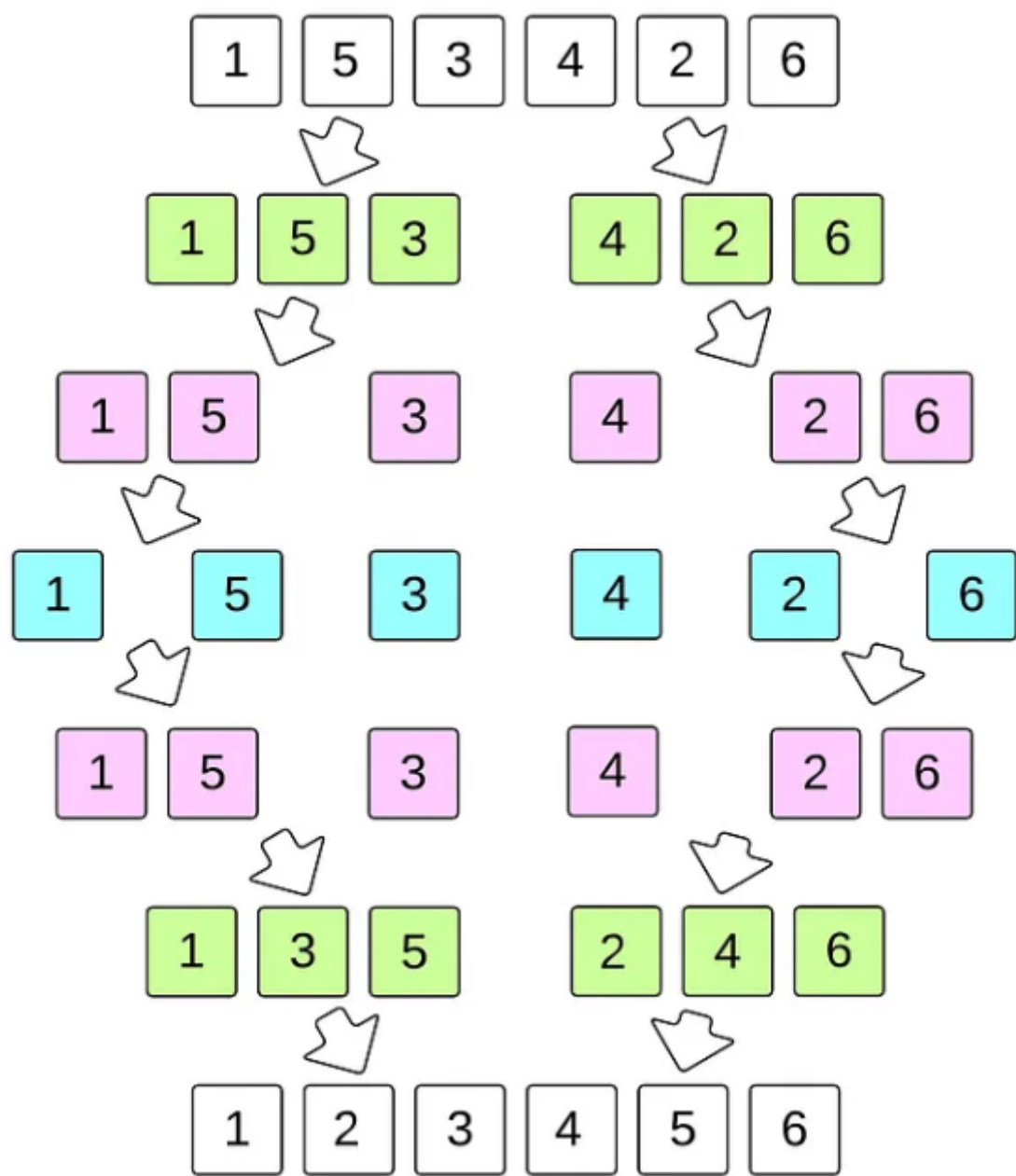
```
sort.py > ...
1
2 my_array = [64, 25, 99, 12, 22, 11, 7, 33]
3 for i in range(len(my_array)-1):
4     min_index = i
5     for j in range(i+1, len(my_array)):
6         if my_array[min_index] > my_array[j]:
7             min_index = j
8
9     my_array[i], my_array[min_index] = my_array[min_index], my_array[i]
10
11 print (f"Sorted array: {my_array}")
12
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$ p
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$ p
Sorted array: [7, 11, 12, 22, 25, 33, 64, 99]
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$
```

Efficiency / Performance: the time complexity / worst case Big O notation of this algorithm is  $O(n^2)$  because there are 2 nested loops. Each individual loop to select / compare an element in an array is  $O(n)$  time complexity. Therefore these both calculate as  $O(n) * O(n)$  which =  $O(n^2)$  time complexity for this algorithm. This algorithm will work well with small data sets but not good with large data sets.

2. Merge Sort Algorithm: the process for this algorithm is like a divide and conquer sorting approach. The technique used is to divide an array in half, then again divide these sub arrays into half again until each subsequent sub array has only one element. This has then conquered the unsorted issue by having each individual subarray sorted into individual elements. These individual element subarrays are then recursively merged back together in a sorted order, repeating this process until all sorted arrays are merged back to the input array which is now sorted correctly. An example image is like a tree root being created with branches going down from the unsorted input array as the main root, then these branches recursively merging back up the root tree in a sorted order back until only the main root exists in a sorted order:



Example of merge sort in python code:

```
sort.py > ...
14
15 my_array = [64, 25, 99, 12, 22, 11, 7, 33]
16 def mergeSort(arr):
17     if len(arr) <= 1:
18         return arr
19
20     mid = len(arr) // 2
21     leftHalf = arr[:mid]
22     rightHalf = arr[mid:]
23
24     sortedLeft = mergeSort(leftHalf)
25     sortedRight = mergeSort(rightHalf)
26
27     return merge(sortedLeft, sortedRight)
28
29 def merge(left, right):
30     result = []
31     i = j = 0
32
33     while i < len(left) and j < len(right):
34         if left[i] < right[j]:
35             result.append(left[i])
36             i += 1
37         else:
38             result.append(right[j])
39             j += 1
40
41     result.extend(left[i:])
42     result.extend(right[j:])
43
44     return result
45
46 print("Sorted array: ", (mergeSort(my_array)))
47
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from
Sorted array: [7, 11, 12, 22, 25, 33, 64, 99]
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from
```

Efficiency / Performance: the time complexity / worst case Big O notation of this algorithm is  $O(n \log n)$ . Because we are recursively dividing into subarrays, this will be a time complexity of  $O(\log n)$ . Then for each function call will be a time complexity of  $O(n)$ . Therefore this algorithm results in a  $O(n \log n)$  time complexity. This algorithm will work well on large data sets and is the preferred process for sorting linked lists.

Edge Cases For Sorting Algorithms:

- in any sorting algorithm issues will happen at the extreme boundaries of the operating parameters. Finding ways to reduce these will depend on each use case of the structure trying to be sorted. An edge case example would be if sorting a structure with both string & integer data types. This could be resolved by first converting all values to a string format, then sorting by the algorithm.

References:

GeekforGeeks. 2024. Selection Sort Algorithm [Online] Available at: <https://www.geeksforgeeks.org/selection-sort-algorithm-2/>

GeekforGeeks. 2024. Merge Sort – Data Structure and Algorithms Tutorials [Online] Available at: <https://www.geeksforgeeks.org/merge-sort/>

Parekh D. 2022. Sorting Algorithms: Slowest to Fastest [Online] Available at: <https://builtin.com/machine-learning/fastest-sorting-algorithm>

Bothwell S. 2017. Sorting Algorithms and Big-O Analysis [Online] Available at: <https://medium.com/@ssbothwell/sorting-algorithms-and-big-o-analysis-332ce7b8e3a1>

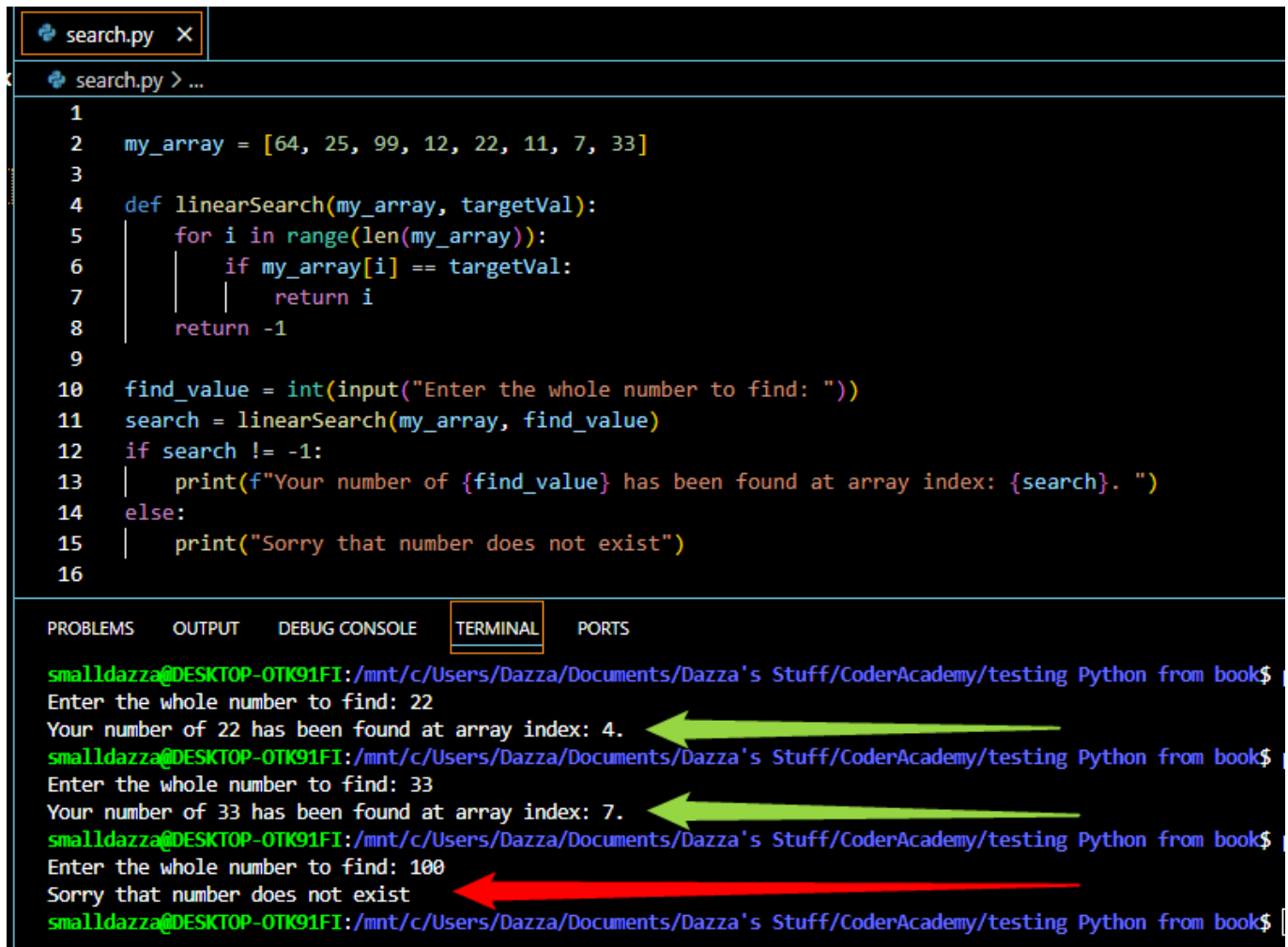
W3 Schools. 2024. DSA Merge Sort [Online] Available at: [https://www.w3schools.com/dsa/dsa\\_algo\\_mergesort.php](https://www.w3schools.com/dsa/dsa_algo_mergesort.php)

Khandaker S. 2021. Taking the Edge off of Edge Cases [Online] Available at: <https://medium.com/swlh/taking-the-edge-off-of-edge-cases-7b3008d83a57>

## Question 2 Answer:

Two different types of searching algorithms are:

1. Linear Search Algorithm: this search algorithm has a simple technique of looking at every element in an array to find (or not) the key search value. The algorithm process works like this: It will begin by looking at the first element in the array, then compares this to the key input value, if it is a match then this will be returned and treated as search found / done, if not a match then will move to the next element in the array to compare this. This process will repeat through all elements in the array until a match is found or all elements have been checked and no match found. Example of selection sort in python code:



The image shows a code editor with a file named `search.py`. The code implements a linear search algorithm. Below the code, the `TERMINAL` tab is active, showing the program's execution. Three green arrows point from the terminal output to the corresponding lines in the code: the first arrow points from the output 'Your number of 22 has been found at array index: 4.' to line 7 (the `return i` statement); the second arrow points from 'Your number of 33 has been found at array index: 7.' to line 7; the third arrow points from 'Sorry that number does not exist' to line 15 (the `print` statement for the 'not found' case).

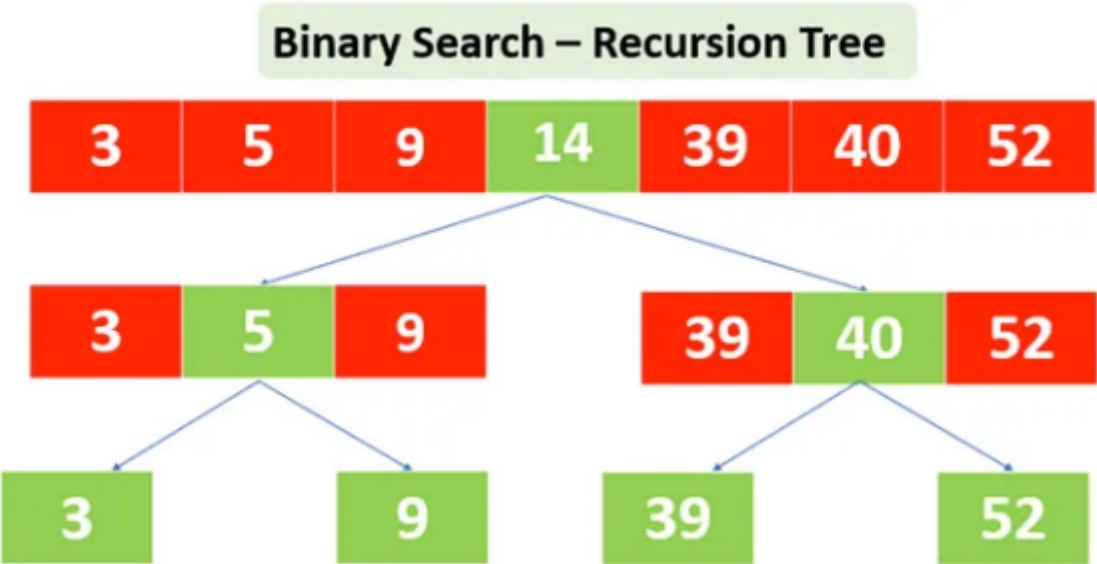
```
1 my_array = [64, 25, 99, 12, 22, 11, 7, 33]
2
3
4 def linearSearch(my_array, targetVal):
5     for i in range(len(my_array)):
6         if my_array[i] == targetVal:
7             return i
8     return -1
9
10 find_value = int(input("Enter the whole number to find: "))
11 search = linearSearch(my_array, find_value)
12 if search != -1:
13     print(f"Your number of {find_value} has been found at array index: {search}. ")
14 else:
15     print("Sorry that number does not exist")
16
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$
Enter the whole number to find: 22
Your number of 22 has been found at array index: 4.
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$
Enter the whole number to find: 33
Your number of 33 has been found at array index: 7.
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$
Enter the whole number to find: 100
Sorry that number does not exist
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book$
```

Efficiency / Performance: the time complexity / worst case Big O notation of this algorithm is  $O(n)$ . If the key input value is found at the first element then the best case time complexity is  $O(1)$ , but if the input value is the last element in the array then this worst case time complexity is  $O(n)$ , where 'n' is the size of the array. This algorithm is preferred for small data sets and for searching unsorted arrays, lists etc.

2. Binary Search Algorithm: this search algorithm can only be applied when the data structure has been sorted. Then the technique used to find the input value is a divide and compare process until found or not. The algorithm process works like this: first it looks at the value in the middle of the array, if the key input value matches = then found, but if the key input value is lower than the middle value = then must search the left half of the array, if the key input value is higher than the middle value = then must search the right half of the array, it then repeats this searching process on the chosen halved array portion to search again and repeats this again, until the input value is matched / found or not found. An image of the algorithm process:



Example of binary search in python code:

```

18 my_array = [64, 25, 99, 12, 22, 11, 7, 33]
19 my_array.sort()
20 def binarySearch(arr, targetVal):
21     left = 0
22     right = len(arr) - 1
23
24     while left <= right:
25         mid = (left + right) // 2
26
27         if arr[mid] == targetVal:
28             return mid
29
30         if arr[mid] < targetVal:
31             left = mid + 1
32         else:
33             right = mid - 1
34
35     return -1
36
37 find_value = int(input("Enter the whole number to find: "))
38 search = binarySearch(my_array, find_value)
39 if search != -1:
40     print(f"Your number of {find_value} has been found at array index: {search}. ")
41 else:
42     print("Sorry that number does not exist")
43

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book
Enter the whole number to find: 22
Your number of 22 has been found at array index: 3.
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book
Enter the whole number to find: 33
Your number of 33 has been found at array index: 5.
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book
Enter the whole number to find: 100
Sorry that number does not exist
smalldazza@DESKTOP-0TK91FI:/mnt/c/Users/Dazza/Documents/Dazza's Stuff/CoderAcademy/testing Python from book

```

Efficiency / Performance: the time complexity / worst case Big O notation of this algorithm is  $O(\log n)$ . Because we are recursively dividing into subarrays until the value is found or not, this will be a time complexity of  $O(\log n)$ . This algorithm is preferred for large data sets where efficiency and speed is required. Used often to search databases for specific values and can also be used for string matching in text search engines to search for a pattern in a text.

Edge Cases For Searching Algorithms:

- in any searching algorithm issues will happen depending on the use case of the particular algorithm used. Using good error handling techniques and allowing for a output to users if the search value is not found. For example: when the search value does not exist in the structure, a search algorithm will just end. How does the user know it ended, does the value exist or not?. Adding a output to display to the user it does not exist can resolve this edge case. Example 2: if going to use binary search what happens if the structure is not sorted? this edge case will produce incorrect results. Correctly implementing a sort function to the structure before the binary search algorithm will resolve this edge case.

## References:

GeekforGeeks. 2024. Introduction to Linear Search Algorithm [Online] Available at:  
<https://www.geeksforgeeks.org/linear-search/>

W3 Schools. 2024. DSA Linear Search [Online] Available at:  
[https://www.w3schools.com/dsa/dsa\\_algo\\_linearsearch.php](https://www.w3schools.com/dsa/dsa_algo_linearsearch.php)

Sharma P. 2023. Top 3 Searching Algorithms [Online] Available at: <https://www.linkedin.com/pulse/top-3-searching-algorithms-parth-sharma>

W3 Schools. 2024. DSA Binary Search [Online] Available at:  
[https://www.w3schools.com/dsa/dsa\\_algo\\_binarysearch.php](https://www.w3schools.com/dsa/dsa_algo_binarysearch.php)

GeekforGeeks. 2024. Binary Search Algorithm – Iterative and Recursive Implementation [Online] Available at:  
<https://www.geeksforgeeks.org/binary-search/>