Clean Code

**Searching Algorithms –**

1. **Linear Search -** It is a simple searching algorithm that sequentially checks each element in a list or array until the desired element is found or the entire list has been traversed

**Flowchart** –

**Algorithm** –

1. Start the algorithm.
2. Define a function linear\_search that takes an array arr and a target element target as input.
3. Initialize a for loop with index i, which iterates in the range of 0 to the length of the array arr.
4. Check if arr[i] is equal to target.
5. If the condition is satisfied, return the index i as the result.
6. End the for loop.
7. Return -1 as the result, indicating that the target element was not found.
8. End the function.
9. Call the function linear\_search with the input array my\_list and the target element target\_element.
10. Store the returned result in the variable result.
11. Check if result is equal to -1.
12. If the condition is satisfied, print "Element not found".
13. Otherwise, print "Element found at index" followed by the value of result.
14. End the algorithm.

**Code** –

def linear\_search(arr, target):

    for i in range(len(arr)):

        if arr[i] == target:

            return i  # Return the index of the target element

    return -1  # Return -1 if the target element is not found

my\_list = [4, 2, 8, 5, 1, 9, 3]

target\_element = 5

result = linear\_search(my\_list, target\_element)

if result == -1:

    print("Element not found")

else:

    print("Element found at index", result)

**Output** - Element found at index 3

1. **Binary Search -** Binary search is a searching algorithm that efficiently finds the position of a target value within a sorted array. It works by repeatedly dividing the search space in half until the target element is found or determined to be absent.

**Flowchart** –

**Algorithm** –

1. Start the algorithm.
2. Define a function binary\_search that takes a sorted array arr and a target element target as input.
3. Initialize the left variable to 0, representing the left boundary of the search space.
4. Initialize the right variable to len(arr) - 1, representing the right boundary of the search space.
5. Enter a while loop that continues until the left boundary is less than or equal to the right boundary.
6. Calculate the mid index as the integer division of the sum of left and right by 2.
7. Check if the element at index mid of the array arr is equal to the target element.
8. If the condition is satisfied, return the mid index as the result.
9. Check if the element at index mid is less than the target element.
10. If the condition is satisfied, update the left boundary to mid + 1, discarding the left half of the search space.
11. Otherwise, update the right boundary to mid - 1, discarding the right half of the search space.
12. End the while loop.
13. Return -1 as the result, indicating that the target element was not found.
14. End the function.
15. Create a sorted list called my\_list with some elements.
16. Define the target element as the value you want to search for.
17. Call the binary\_search function with the input array my\_list and the target element.
18. Store the returned result in the variable result.
19. Check if result is equal to -1.
20. If the condition is satisfied, print "Element not found".
21. Otherwise, print "Element found at index" followed by the value of result.
22. End the algorithm.

**Code** –

def binary\_search(arr, target):

    left = 0

    right = len(arr) - 1

    while left <= right:

        mid = (left + right) // 2

        if arr[mid] == target:

            return mid  # Return the index of the target element

        elif arr[mid] < target:

            left = mid + 1  # Update the left boundary

        else:

            right = mid - 1  # Update the right boundary

    return -1  # Return -1 if the target element is not found

my\_list = [1, 2, 3, 4, 5, 8, 9]

target\_element = 5

result = binary\_search(my\_list, target\_element)

if result == -1:

    print("Element not found")

else:

    print("Element found at index", result)

**Output** - Element found at index 4