Linear search (finds the element in the list)
Input: List of nums, target
Output: target position (index)

for each num in array: do

if num = target then
~~return array[~~
return array.index(num)

return -1

Test Case:

① linear_search([1,2,3,4,5],6)
→ element not found

② linear_search([1,12,3,14,5],14)
→ element found at index 3

③ linear_search([],6)
→ The list is empty

④ linear_search((1,4,7,8,5),15)
→ The input are of incorrect datatype.

⑤ linear_search([151,225,31,444,5],225)
→ The element is at index 1.

Binary_Search {
Input: sorted list of nums, target, left, right

Output: Index of target element

binary_search(nums, target, left, right).

mid = (left + right) // 2

if left > right then

return element not found

if element at index mid = target then

return index mid

if element at index mid < target then

return call binary_search with parameters nums, target, mid+1, right

else

return call binary_search with parameters nums, target, left, mid -1

Testcases

① binary_search([1, 2, 3, 4, 5], 6, 0, len([1,2,3,4,5] -1)
→ element not found.

② binary_search([1, 2, 3, 4, 5], 4, 0, len[1,2,3,4,5] -1)
→ unsorted array received.

③ binary_search([], 6, 0, len([]) -1)
→ list is empty.

④ binary_search((1, 4, 7, 8, 5), 5, 0, len((1,4,7,8,5) -1))
→ invalid datatype of inputs.

~~⑤ binary_search([100, 200, 400, 500, 800], 500, 0, len[100,200~~
~~→ element at index 3.~~

⑤ binary_search([100, 200, 400, 500, 800], 500, 0, len([100, 200, 400, 500, 800]) -1)
→ element at index 3.

linear search: input size $= n$

$$C_{wost}(n) = \sum_{i=0}^{n-1} 1 = [(n-1)+1]$$

$$= n$$

$\therefore$ TC $\in O(n)$

binary search: input size $= n$

basic operation: check conditions and make a recursive call

let $A(n)$ be time complexity of algorithm

$A(n) = A(n/2) + C$ [C is the time for checking the condition]
$A(n/2) = A(n/4) + C$

$\therefore A(n) = A(n/4) + 2C$

$\vdots$

$\therefore A(n) = A(n/2^k) + kC$ — ①

$\vdots$

$\therefore A(n) = A(1) + C$ C = constant

~~$\therefore A(n)$~~

$$\frac{n}{2^k} = 1$$

$\therefore n = 2^k$

$\therefore \log_2 n = k$

$\therefore A(n) = A(n/2^{\log_2 n}) + c\log_2 n.$

$$= A(n/n) + c\log_2 n$$
$$= A(1) + c\log_2 n$$
$$= c + c\log_2 n$$

$\therefore A(n) = c(1 + \log_2 n)$

$\therefore$ ~~Ati~~ Time complexity $\in O(\log_2 n)$

The code is written in python as per the PEP 8 coding style.

```
Code:

# Linear Search
def linear_search(nums, target):
    """This function performs linear search on the list inputted
    Time Complexity is O(n)"""

    if type(nums) != list:  # Datatype Check
        return "The inputs are of incorrect datatype!!"

    if len(nums) == 0:  # Empty List Check
        return "The list is empty!!"

    for i in range(len(nums)):  # Main Loop
        if nums[i] == target:
            return "The target is present at index " + str(i)

    return "The target element is not found in the list!!"


# Binary Search
def binary_search(nums, target, left, right):
    """This function performs binary search on the list inputted in
recursive pattern
    Time Complexity is O(log n)"""

    if type(nums) != list:  # Data Type Check
        return "The inputs are of incorrect datatype!!"

    for i in range(1, len(nums) - 1):  # Unsorted Input Check
        if nums[i - 1] > nums[i]:
            return "This is an unsorted array!!"

    if len(nums) == 0:  # Empty List Check
        return "The list is empty!!"

    # Pointers Initialization
    mid = (left + right) // 2

    if left > right:
        return "The target element is not found in the list!!"
    if nums[mid] == target:
        return "The target is present at index " + str(mid)
    elif nums[mid] < target:
        return binary_search(nums, target, mid + 1, right)
    else:
        return binary_search(nums, target, left, mid - 1)
```

```python
# Test Cases
print("Test Case No 1: " + linear_search([1, 2, 3, 4, 5], 6))
print("Test Case No 2: " + linear_search([1, 12, 3, 14, 5], 14))
print("Test Case No 3: " + linear_search([], 6))
print("Test Case No 4: " + linear_search((1, 4, 7, 8, 5), 15))
print("Test Case No 5: " + linear_search([151, 225, 31, 444, 5], 225))

print("Test Case No 1: " + binary_search([1, 2, 3, 4, 5], 6, 0, len([1, 2, 3, 4, 5]) - 1))
print("Test Case No 2: " + binary_search([1, 12, 3, 14, 5], 14, 0, len([1, 12, 3, 14, 5]) - 1))
print("Test Case No 3: " + binary_search([], 6, 0, len([]) - 1))
print("Test Case No 4: " + binary_search((1, 4, 7, 8, 5), 15, 0, len((1, 4, 7, 8, 5)) - 1))
print("Test Case No 5: " + binary_search([100, 200, 400, 500, 800], 500, 0, len([100, 200, 400, 500, 800]) - 1))
```

Output:

```
Run: Test
C:\Users\user\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:/PYTHON/Codes/Misc/Test.py
Test Case No 1: The target element is not found in the list!!
Test Case No 2: The target is present at index 3
Test Case No 3: The list is empty!!
Test Case No 4: The inputs are of incorrect datatype!!
Test Case No 5: The target is present at index 1
Test Case No 1: The target element is not found in the list!!
Test Case No 2: This is an unsorted array!!
Test Case No 3: The list is empty!!
Test Case No 4: The inputs are of incorrect datatype!!
Test Case No 5: The target is present at index 3

Process finished with exit code 0
```

Conclusion:

The code implemented in this lab assignment effectively performs linear search and binary search on the given inputs with appropriate error handling. Linear search has a time complexity of O(n) and checks each element sequentially, while binary search, with O(log n) complexity, requires a sorted list and uses recursive calls to update the pointers and find the element.