Insertion sort:

```
def insertion_sort(arr):


  for i in range(1, len(arr)):  ----------------------------------------------------> n-1 times
    key = arr[i]  # key is the current element to be compared ------------------------> S1


    j = i-1


    while j >= 0 and key < arr[j]: ----------------------------------------------> n(n-1)/2 times
      arr[j + 1] = arr[j]
      j -= 1
    arr[j + 1] = key
```

Time complexity:

Best case:

n-1 times for loop

Worst case:

n-1 times for loop

n(n-1)/2 times ------- > While loop

Space complexity:

Best case:

S1

Worst case:

S1

>>>

Linear Search:

```
def linear_search(arr, target):

    for i in range(len(arr)): ------------------------------------------------------- > n
        if arr[i] == target:
            return i  # Return the index of the target element if found
    return -1  # If target element is not found, return -1

# Example usage:
arr = [4, 2, 7, 1, 9, 5]
target = 7 -----------------------------------------> S1
index = linear_search(arr, target)
if index != -1:
    print(f"Target element {target} found at index {index}.")
else:
    print(f"Target element {target} not found in the array.")
```

Time complexity:

<u>Best case:</u>

1 time for loop

<u>Worst case:</u>

n times for loop

Space complexity :

Best case:

1

Worst case:

1

>>

Binary Search:

```
def binary_search(arr, target):

  left = 0-------------------------------- > S 1
  right = len(arr) - 1 --------------------> S 2

  while left <= right: ---------------->  2x=n
    mid = (left + right) // 2 -------------> S 3

    # Check if the target is equal to the middle element
    if arr[mid] == target:
      return mid

    # If the target is less than the middle element, search the left half
    elif arr[mid] > target:
      right = mid - 1

    # If the target is greater than the middle element, search the right half
    else:
      left = mid + 1
```

```
    # If the target element is not found in the array
    return -1

# Example usage:
arr = [1, 3, 5, 7, 9, 11, 13, 15]
target = 9 ----------------------------------------------> S4
index = binary_search(arr, target)
if index != -1:
    print(f"Target element {target} found at index {index}.")
else:
    print(f"Target element {target} not found in the array.")
```

Time complexity:

Best case:

Mathematically:

$2x=4$

$22=4$

$2x = n$

Worst case scenario:

$2n$

Best case:

1 – Equal to mid element

Space complexity:

Best case :

S4

Worst case:

S4