📘 Lecture Notes: Binary Search

1. Problem Definition

- Input: Sorted list of numbers (or strings, etc.) and a target element `x`.
- Output: Index of `x` if found, otherwise `None` (or `False`).
- *Key requirement:* List must be sorted (ascending or descending).

---

2. Key Idea

- Maintain two pointers: `left` and `right` defining the search region.
- Compute `mid = (left + right) // 2`.
- Compare `list[mid]` with target:
  - If equal → return `mid`.
  - If target < `list[mid]` → search left half.
  - If target > `list[mid]` → search right half.
- Terminate when `left > right`.

---

3. Examples

- List = `[1, 6, 7, 19, 22, 25, 31, 55]`
  - Search for `18` → Not found (`None`).
  - Search for `6` → Found at index `1`.

---

4. Recursive Implementation (Python)

```
def binary_search(arr, target):
    def helper(left, right):
        if left > right:
            return None
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            return helper(mid + 1, right)
        else:
            return helper(left, mid - 1)
    return helper(0, len(arr) - 1)
```

---

5. Correctness Argument

- **Invariant:** At each step, if the element exists in the list, it must be within `[left, right]`.
- Recursive calls preserve this invariant.
- When `left > right`, the invariant guarantees the element is absent.
- Proof can be shown via induction.

---

6. Time Complexity Analysis

- Each iteration halves the search region.
- If size = `n = 2^k`, after `k` steps search region shrinks to size `1`.
- Worst-case number of steps = $\log_2(n) + 1$.
- Complexity:

- **Worst-case:** `O(log n)`
- **Best-case:** `O(1)` (if middle element is target).
- **Space:** `O(1)` for iterative, `O(log n)` for recursive (stack frames).

---

7. Practical Implications

- Searching in 1 million items → ~20 steps.
- Searching in 1 trillion items → ~30 steps.
- Extremely efficient compared to linear search (`O(n)`).

✅ Clear takeaway: **Binary Search is simple, elegant, and very fast—but tricky to implement correctly.**