1. Searching Basics

- Searching is the most common operation in databases.
- SQL's `SELECT` statement is very versatile and complex.
- **Linear Search** is the baseline for efficiency:
    - Starts at beginning of list
    - Continues element by element until:
        - Match found
        - End reached without finding match

2. Key Terminology

- **Record**: A row in a table (collection of attribute values for an entity).
- **Collection**: A set of records of the same type (a table).
- **Search Key**: Attribute(s) used to search; can be one or more attributes.

3. Record Storage Structures

Contiguous Allocation (Array)

- Requires n * x bytes for n records of size x.
- Fast random access.
- Slow insertions (especially not at the end).

Linked List

- Each record uses x bytes + memory for address pointers.
- Records linked via memory addresses.
- Fast insertions.
- Slow random access.

4. Binary Search Algorithm

- **Precondition**: Array must be sorted.
- **Goal**: Find index of target value or return -1 if not found.

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Time Complexity

- **Linear Search**:
    - Best: O(1) (first element)
    - Worst: O(n)
- **Binary Search**:
    - Best: O(1) (middle element)
    - Worst: $O(\log_2 n)$

5. Database Search Considerations

- Data typically stored sorted by `id` → fast lookup by `id`.
- Searching other columns (e.g., `specialVal`) requires **linear scan**.
- Can't sort disk data by both `id` and `specialVal` without duplicating data → **space inefficient**.

6. External Structures for Search Optimization

Option 1: Array of Tuples `(specialVal, rowNumber)` (Sorted)

- Supports binary search (fast lookup).
- Slow insertions (inserting into sorted array is expensive).

Option 2: Linked List of Tuples `(specialVal, rowNumber)` (Sorted)

- Fast insertions.
- Slow searches (linear scan needed).

7. Ideal Structure: Binary Search Tree (BST)

- Tree structure where:
    - Left subtree < parent
    - Right subtree > parent
- Supports:
    - **Fast searches**
    - **Fast insertions** (on average)
- Balanced BSTs (e.g., AVL, Red-Black Trees) offer guaranteed performance.