Basic Algorithms: Sorting and Searching
Introduction to Sorting – Bubble Sort
What is Sorting?
Sorting means arranging data in a specific order:

- **Ascending** (A → Z, small → large)
- **Descending** (Z → A, large → small)

What is Bubble Sort?
Bubble Sort is a simple sorting algorithm that:

- Compares **adjacent elements**
- Swaps them if they are in the wrong order
- Repeats until the list is sorted

Larger elements “bubble up” to the end.

Bubble Sort Logic

1. Start from the first element
2. Compare with the next element
3. Swap if needed
4. Repeat for all elements
5. Continue passes until sorted

Bubble Sort Example (Numbers)

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

numbers = [5, 2, 9, 1, 3]
print("Sorted list:", bubble_sort(numbers))
```

Time Complexity

- Worst case: **O(n²)**
- Best case: **O(n)** (already sorted)

Searching – Linear Search
What is Searching?
Searching means finding the position of a specific element in a list.

What is Linear Search?
Linear Search:

- Checks elements **one by one**

- Works on **unsorted lists**
- Simple but slower for large data

Linear Search Logic

1. Start from first element
2. Compare with target value
3. If match found → return index
4. If not found → return -1

Linear Search Example

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

items = [10, 20, 30, 40, 50]
result = linear_search(items, 30)

if result != -1:
    print("Element found at index:", result)
else:
    print("Element not found")
```

⏱ **Time Complexity**

- Best case: **O(1)**
- Worst case: **O(n)**

Activity: Sorting a List of Names

Problem Statement

Sort a list of student names alphabetically and search for a specific name.

Step 1: Bubble Sort for Names

```
def bubble_sort_names(names):
    n = len(names)
    for i in range(n):
        for j in range(0, n - i - 1):
            if names[j] > names[j + 1]:
                names[j], names[j + 1] = names[j + 1], names[j]
    return names

names = ["Athar", "Zain", "Imran", "Bilal", "Ayaan"]
sorted_names = bubble_sort_names(names)
print("Sorted Names:", sorted_names)
```

Step 2: Linear Search on Names

```
def linear_search_name(names, target):
    for i in range(len(names)):
        if names[i] == target:
            return i
    return -1

search_name = "Imran"
index = linear_search_name(sorted_names, search_name)

if index != -1:
    print(search_name, "found at position", index)
else:
    print(search_name, "not found")
```

Real-World Use Cases

Algorithm	Use Case
Bubble Sort	Small datasets, teaching basics
Linear Search	Searching contacts, roll numbers
Sorting Names	Student lists, employee records

Key Notes

- **Bubble Sort** compares adjacent elements repeatedly
- **Linear Search** checks elements sequentially
- Bubble Sort is **not efficient** for large data
- Linear Search works on **both sorted & unsorted lists**