1. **Understand Search Algorithms:**

**Explain linear search and binary search algorithms:**

**Linear Search Algorithm**

**Definition**:  
Linear Search is a simple search technique where each element in the array or list is checked **one by one** until the desired element is found or the list ends.

**How it works**:

1. Start from the first element of the array.
2. Compare each element with the target value.
3. If a match is found, return the index.
4. If the end is reached without a match, return -1.

**Example**:

int[] arr = {10, 20, 30, 40};

int key = 30;

for (int i = 0; i < arr.length; i++) {

if (arr[i] == key) {

System.out.println("Element found at index " + i);

break;

}

}

**UseCase**:  
Use Linear Search when the list is **unsorted** or **small in size**

**Binary Search Algorithm**

**Definition**:  
Binary Search is an efficient algorithm used to find an element in a **sorted** list by repeatedly dividing the search interval in half.

**How it works**:

1. Find the middle element of the array.
2. If the middle element matches the target, return the index.
3. If the target is less than the middle, repeat the search in the left half.
4. If the target is greater, search in the right half.
5. Repeat until the target is found or the subarray becomes empty.

**Example**:

int[] arr = {10, 20, 30, 40, 50};

int key = 30;

int low = 0, high = arr.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid] == key) {

System.out.println("Element found at index " + mid);

break;

} else if (key < arr[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}}

**UseCase**:  
Use Binary Search when the list is **sorted** and you need to search quickly in large datasets.

**4.Analysis**

**Compare the time complexity of linear and binary search:**

| **Aspect** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Best Case** | O(1) | O(1) |
| **Average Case** | O(n) | O(log n) |
| **Worst Case** | O(n) | O(log n) |
| **Data Requirement** | Works on **unsorted** data | Requires **sorted** data |
| **Efficiency** | Less efficient for large data | Highly efficient for large sorted data |
| **Use Case** | Small or unsorted arrays | Large and sorted arrays |
| **Implementation** | Simple | Slightly complex |

**Discuss when to use each algorithm based on the data set size and order:**

**Linear Search: When to Use**

* Use **linear search** when the **data set is small**, such as a list with fewer than 20 elements.
* It is the best choice when the data is **unsorted**, because linear search does not require any order in the data.
* If the search operation is done only **once or rarely**, linear search is simpler and faster to implement.
* It is useful when the **data changes frequently**, as there's no need to re-sort the data before searching.
* It's also ideal for situations where **simplicity** is more important than speed, such as in small programs or educational examples.

**Binary Search: When to Use**

* Use **binary search** when the **data set is large**, typically with 100 or more elements.
* The data must be **sorted** in order to use binary search. If it's not sorted, you must sort it first.
* It is ideal when you need to perform **frequent searches**, as it is much faster than linear search for large datasets.
* Binary search is most efficient in **read-heavy** environments where the data does not change often.
* It provides a significant performance boost because it reduces the number of comparisons using the **divide-and-conquer** method.