1️⃣ Heap Sort in Java

◆ What is Heap Sort?

Heap Sort is a **comparison-based sorting algorithm** that uses a **Binary Heap** data structure.

- A **heap** is a **complete binary tree**
- In **Max Heap** → Parent ≥ Children
- In **Min Heap** → Parent ≤ Children

☞ For **ascending order**, we use **Max Heap**

◆ Why use Heap Sort?

✓ Fast
✓ No extra memory (in-place)
✓ Guaranteed **O(n log n)** time

◆ Steps of Heap Sort

1. Convert array into **Max Heap**
2. Swap first element (largest) with last
3. Reduce heap size
4. Heapify again
5. Repeat until sorted

◆ Heap Sort Example

Input:
```
[4, 10, 3, 5, 1]
```

Output:
```
[1, 3, 4, 5, 10]
```

◆ Heap Sort Java Code

```
class HeapSort {

  // Function to heapify
```

```
  static void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
      largest = left;

    if (right < n && arr[right] > arr[largest])
      largest = right;

    if (largest != i) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;

      heapify(arr, n, largest);
    }
  }

  // Heap Sort
  static void heapSort(int arr[]) {
    int n = arr.length;

    for (int i = n / 2 - 1; i >= 0; i--)
      heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
      int temp = arr[0];
      arr[0] = arr[i];
      arr[i] = temp;

      heapify(arr, i, 0);
    }
  }

  public static void main(String[] args) {
    int arr[] = {4, 10, 3, 5, 1};

    heapSort(arr);

    for (int i : arr)
      System.out.print(i + " ");
  }
}
```

◆ Heap Sort Time & Space

Case	Time
Best	O(n log n)
Average	O(n log n)
Worst	O(n log n)

Space: **O(1)**
Stable: ✕ No

---

2 Binary Sort (Binary Insertion Sort)

◆ What is Binary Sort?

Binary Sort is **Insertion Sort + Binary Search**.

- Uses **binary search** to find correct position
- Shifts elements like insertion sort

☞ Reduces comparisons but **not shifting time**

◆ Why use Binary Sort?

✓ Better than normal insertion sort
✓ Easy to implement
✕ Still slow for large data

◆ Binary Sort Example

Input:
```
[5, 2, 9, 1]
```

Output:
```
[1, 2, 5, 9]
```

◆ Binary Sort Java Code

```
class BinarySort {

   static int binarySearch(int arr[], int item, int low, int high) {
```

```
    if (high <= low)
      return (item > arr[low]) ? (low + 1) : low;

    int mid = (low + high) / 2;

    if (item == arr[mid])
      return mid + 1;

    if (item > arr[mid])
      return binarySearch(arr, item, mid + 1, high);

    return binarySearch(arr, item, low, mid - 1);
  }

  static void binarySort(int arr[]) {
    int n = arr.length;

    for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;

      int loc = binarySearch(arr, key, 0, j);

      while (j >= loc) {
        arr[j + 1] = arr[j];
        j--;
      }
      arr[j + 1] = key;
    }
  }

  public static void main(String[] args) {
    int arr[] = {5, 2, 9, 1};

    binarySort(arr);

    for (int i : arr)
      System.out.print(i + " ");
  }
}
```

◆ Binary Sort Time & Space

Case	Time
Best	O(n log n)
Average	$O(n^2)$
Worst	$O(n^2)$

Space: **O(1)**
Stable: ✅ Yes

🔥 Heap Sort vs Binary Sort

Feature	**Heap Sort**	**Binary Sort**
Speed	Fast	Slow
Space	O(1)	O(1)
Stable	❌	✅
Large Data	✅ Best	❌ No