Step 2 : Learn About Sorting Techniques

1.1 Selection Sort

Difficulty: **Easy**Accuracy: **64.33%**Submissions: **205K+**Points: **2**Average Time: **15m**

Given an array **arr**, use **selection sort** to sort arr[] in increasing order.

Examples :

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Explanation: Maintain sorted (in bold) and unsorted subarrays. Select 1. Array becomes **1** 4 3 9 7. Select 3. Array becomes **1 3** 4 9 7. Select 4. Array becomes **1 3 4** 9 7. Select 7. Array becomes **1 3 4 7** 9. Select 9. Array becomes **1 3 4 7 9**.

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input: arr[] = [38, 31, 20, 14, 30]

Output: [14, 20, 30, 31, 38]

Constraints:
$1 \le$ arr.size() $\le 10^3$
$1 \le$ arr[i] $\le 10^6$

1.2 **Bubble Sort**

Difficulty: **Easy**Accuracy: **59.33%**Submissions: **295K+**Points: **2**Average Time: **15m**

Given an array, **arr[]**. Sort the array using bubble sort algorithm.

Examples :

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input: arr[] = [1, 2, 3, 4, 5]

Output: [1, 2, 3, 4, 5]
Explanation: An array that is already sorted should remain unchanged after applying bubble sort.

Constraints:
1 <= arr.size() <= 10^3
1 <= arr[i] <= 10^3

1.3 **Insertion Sort**

Difficulty: **Easy**Accuracy: **66.61%**Submissions: **233K+**Points: **2**Average Time: **15m**

The task is to complete the **insertsort()** function which is used to implement Insertion Sort.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]
Explanation: The sorted array will be [1, 3, 4, 7, 9].

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Explanation: The sorted array will be [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

Input: arr[] = [4, 1, 9]

Output: [1, 4, 9]
Explanation: The sorted array will be [1, 4, 9].

Constraints:
1 <= arr.size() <= 1000
1 <= arr[i] <= 1000

1.4 **Merge Sort**

Difficulty: **Medium**Accuracy: **54.1%**Submissions: **245K+**Points: **4**Average Time: **15m**

Given an array arr[], its starting position l and its ending position r. Sort the array using the merge sort algorithm.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input: arr[] = [1, 3 , 2]

Output: [1, 2, 3]

Constraints:
1 <= arr.size() <= 10^5
1 <= arr[i] <= 10^5

1.5 **Quick Sort**

Difficulty: **Medium**Accuracy: **55.23%**Submissions: **272K+**Points: **4**Average Time: **15m**

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr**[] in ascending order. Given an array, **arr**[], with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Note: The **low** and **high** are inclusive.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]

Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.

Input: arr[] = [5, 5, 5, 5]

Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.

Constraints:
1 <= arr.size() <= 10^5
1 <= arr[i] <= 10^5