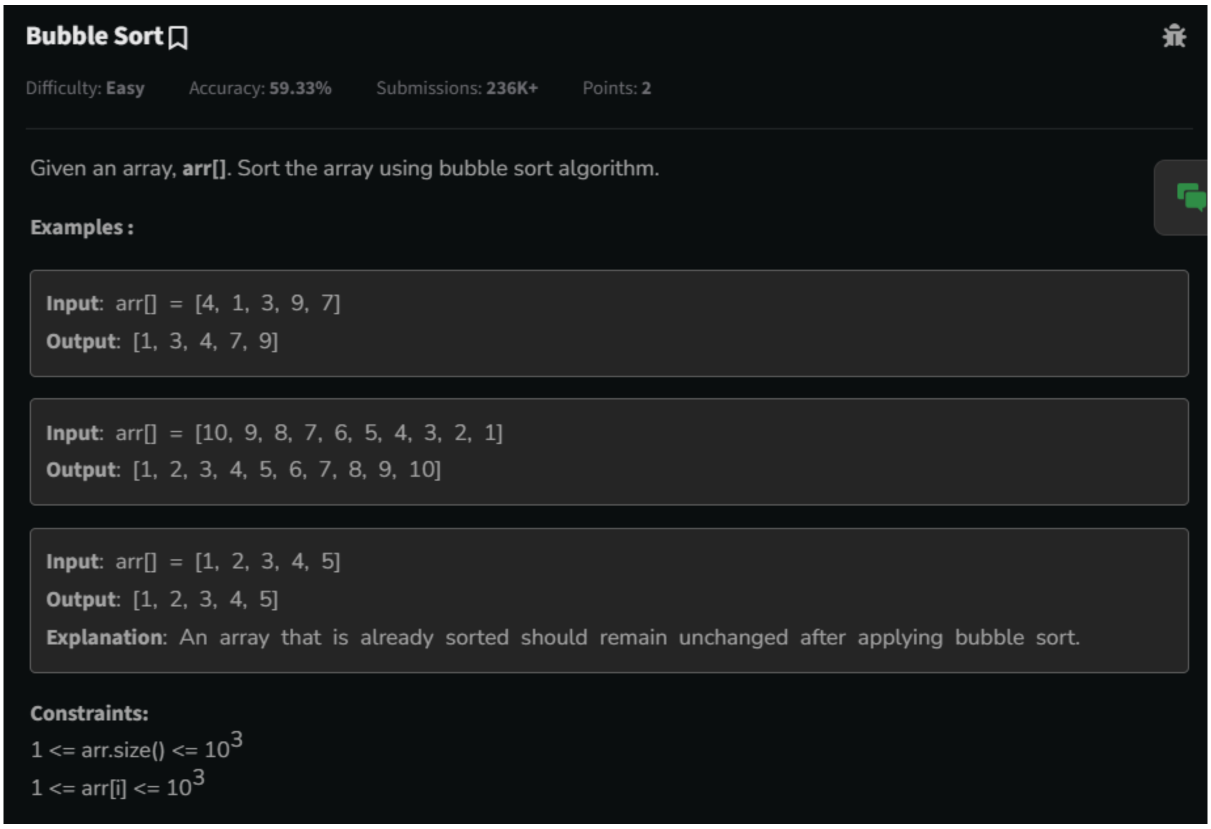
Coding Practice Set-6

1. **Bubble Sort**



Bubble Sort

Difficulty: **Easy** Accuracy: **59.33%** Submissions: **236K+** Points: **2**

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]
```

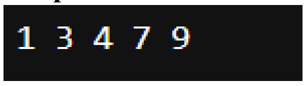
```
Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
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

Code:
```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {4, 1, 3, 9, 7};
        bubbleSort(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

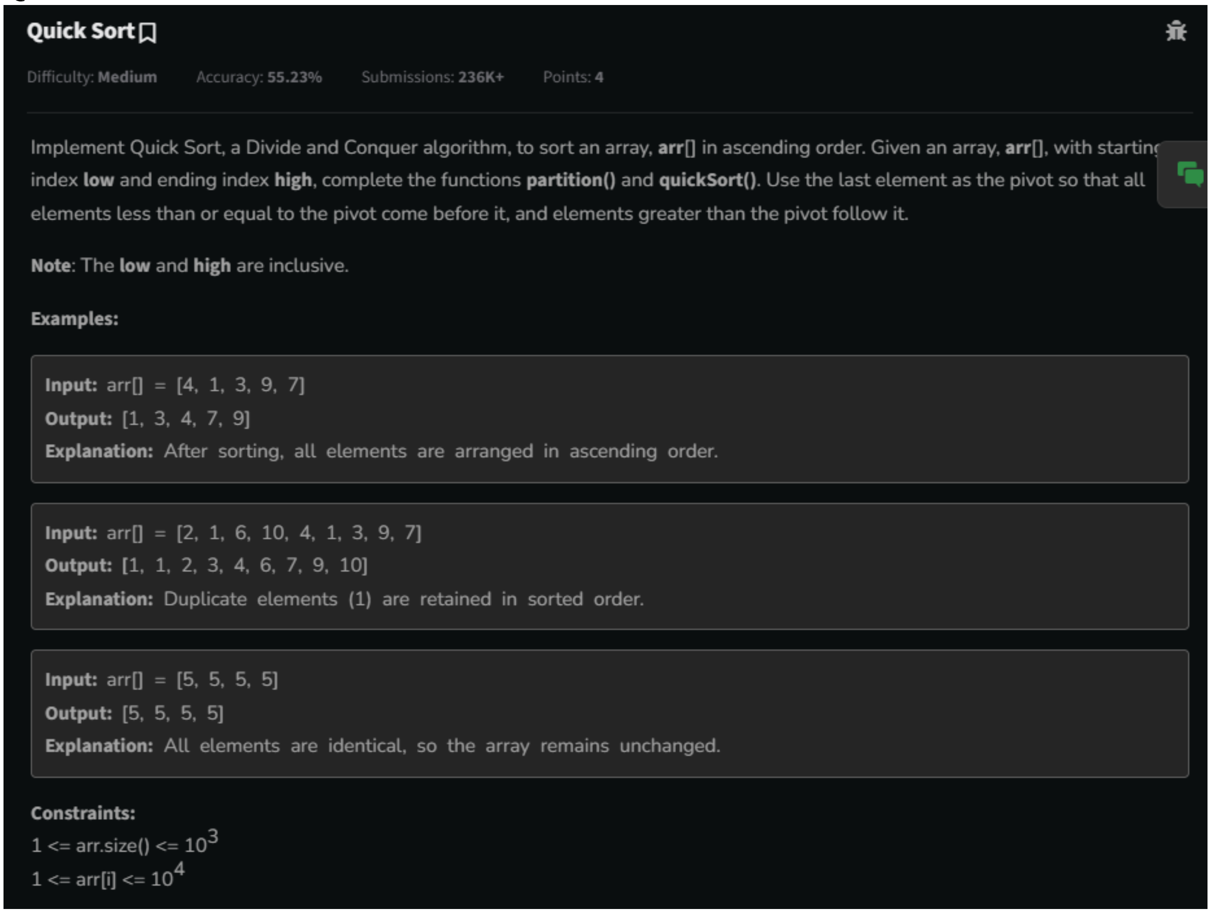
Output:



```
1 3 4 7 9
```

Time Complexity: $O(n^2)$

2. Quick Sort


Quick Sort

Difficulty: **Medium** Accuracy: **55.23%** Submissions: **236K+** Points: **4**

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr[]** in ascending order. Given an array, **arr[]**, with startin 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.
```

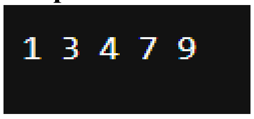
```
Input: arr[] = [5, 5, 5, 5]
Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.
```

Constraints:
1 <= arr.size() <= 10^3
1 <= arr[i] <= 10^4

Code:

```
public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {4, 1, 3, 9, 7};
        quickSort(arr, 0, arr.length - 1);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

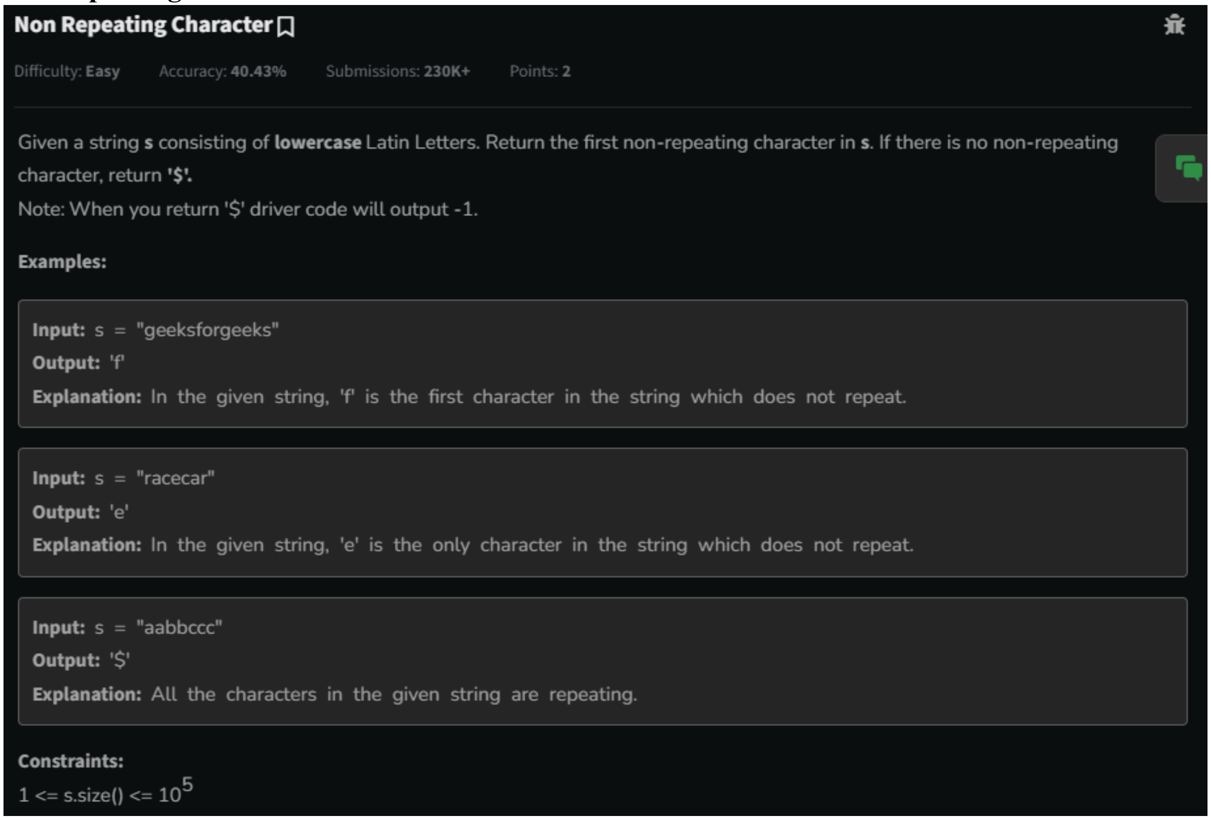
Output:



```
1 3 4 7 9
```

Time Complexity: O(logn)

3. Non repeating character



Non Repeating Character

Difficulty: **Easy** Accuracy: **40.43%** Submissions: **230K+** Points: **2**

Given a string **s** consisting of **lowercase** Latin Letters. Return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'**.
Note: When you return '$' driver code will output -1.

Examples:

Input: s = "geeksforgeeks"
Output: 'f'
Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Input: s = "racecar"
Output: 'e'
Explanation: In the given string, 'e' is the only character in the string which does not repeat.

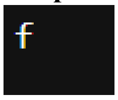
Input: s = "aabbccc"
Output: '$'
Explanation: All the characters in the given string are repeating.

Constraints:
1 <= s.size() <= 10^5

Code:

```
import java.util.LinkedHashMap;
import java.util.Map;

public class NonRepeatingCharacter {
    public static char firstNonRepeatingChar(String s) {
        Map<Character, Integer> charCount = new LinkedHashMap<>();
        for (char c : s.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }
        for (Map.Entry<Character, Integer> entry : charCount.entrySet()) {
            if (entry.getValue() == 1) {
                return entry.getKey();
            }
        }
        return '$';
    }

    public static void main(String[] args) {
        String s = "geeksforgeeks";
        System.out.println(firstNonRepeatingChar(s));
    }
}
```

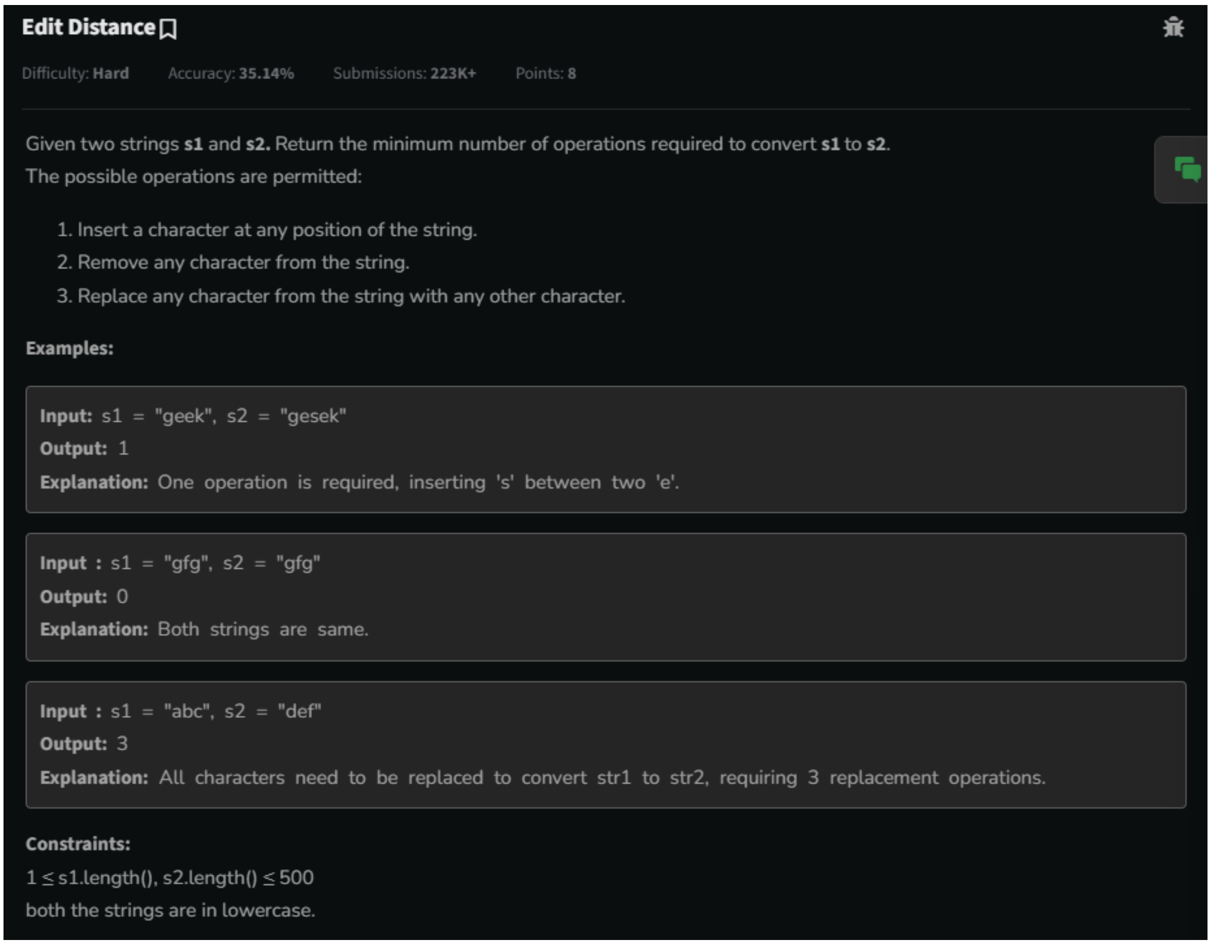
Output:



```
f
```

Time Complexity: O(n)

4. Edit Distance


Edit Distance

Difficulty: **Hard** Accuracy: **35.14%** Submissions: **223K+** Points: **8**

Given two strings **s1** and **s2.** Return the minimum number of operations required to convert **s1** to **s2**.
The possible operations are permitted:

1. Insert a character at any position of the string.
2. Remove any character from the string.
3. Replace any character from the string with any other character.

Examples:

```
Input: s1 = "geek", s2 = "gesek"
Output: 1
Explanation: One operation is required, inserting 's' between two 'e'.
```

```
Input : s1 = "gfg", s2 = "gfg"
Output: 0
Explanation: Both strings are same.
```

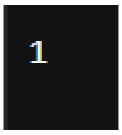
```
Input : s1 = "abc", s2 = "def"
Output: 3
Explanation: All characters need to be replaced to convert str1 to str2, requiring 3 replacement operations.
```

Constraints:
1 ≤ s1.length(), s2.length() ≤ 500
both the strings are in lowercase.

Code:

```
public class EditDistance {
  public static int minDistance(String s1, String s2) {
    int m = s1.length(), n = s2.length();
    int[][] dp = new int[m + 1][n + 1];

    for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
        if (i == 0) {
          dp[i][j] = j;
        } else if (j == 0) {
          dp[i][j] = i;
        } else if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = 1 + Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1]));
        }
      }
    }
    return dp[m][n];
  }

  public static void main(String[] args) {
        String s1 = "geek", s2 = "gesek";
        System.out.println(minDistance(s1, s2));
    }
}
```

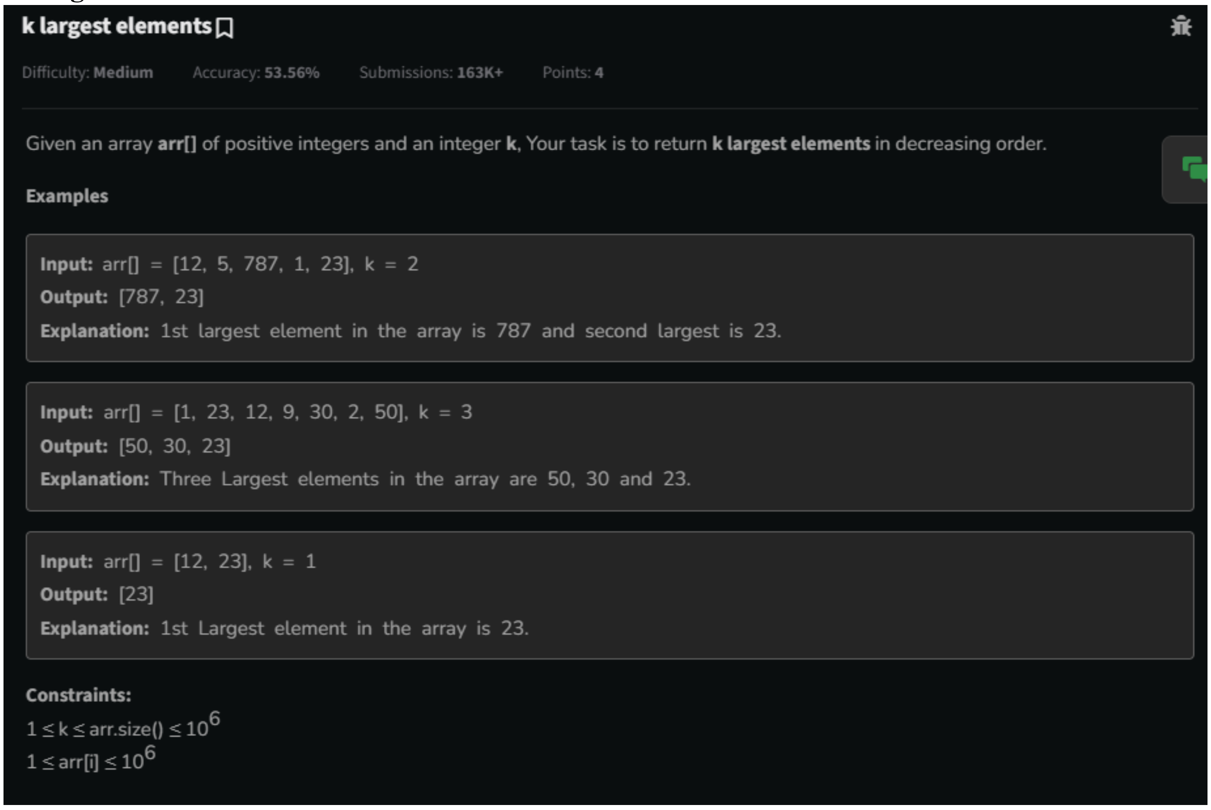
Output:



```
1
```

Time Complexity: O(m*n)

5. K largest elements



k largest elements

Difficulty: **Medium** Accuracy: **53.56%** Submissions: **163K+** Points: **4**

Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements** in decreasing order.

Examples

```
Input: arr[] = [12, 5, 787, 1, 23], k = 2
Output: [787, 23]
Explanation: 1st largest element in the array is 787 and second largest is 23.
```

```
Input: arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3
Output: [50, 30, 23]
Explanation: Three Largest elements in the array are 50, 30 and 23.
```

```
Input: arr[] = [12, 23], k = 1
Output: [23]
Explanation: 1st Largest element in the array is 23.
```

Constraints:

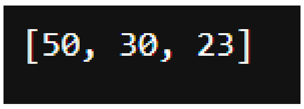
$1 \le k \le arr.size() \le 10^6$

$1 \le arr[i] \le 10^6$

Code:

```
import java.util.*;

public class KLargestElements {
    public static List<Integer> kLargest(int[] arr, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int num : arr) {
            minHeap.add(num);
            if (minHeap.size() > k) {
                minHeap.poll();
            }
        }
        List<Integer> result = new ArrayList<>(minHeap);
        result.sort(Collections.reverseOrder());
        return result;
    }

    public static void main(String[] args) {
        int[] arr = {1, 23, 12, 9, 30, 2, 50};
        int k = 3;
        System.out.println(kLargest(arr, k));
    }
}
```

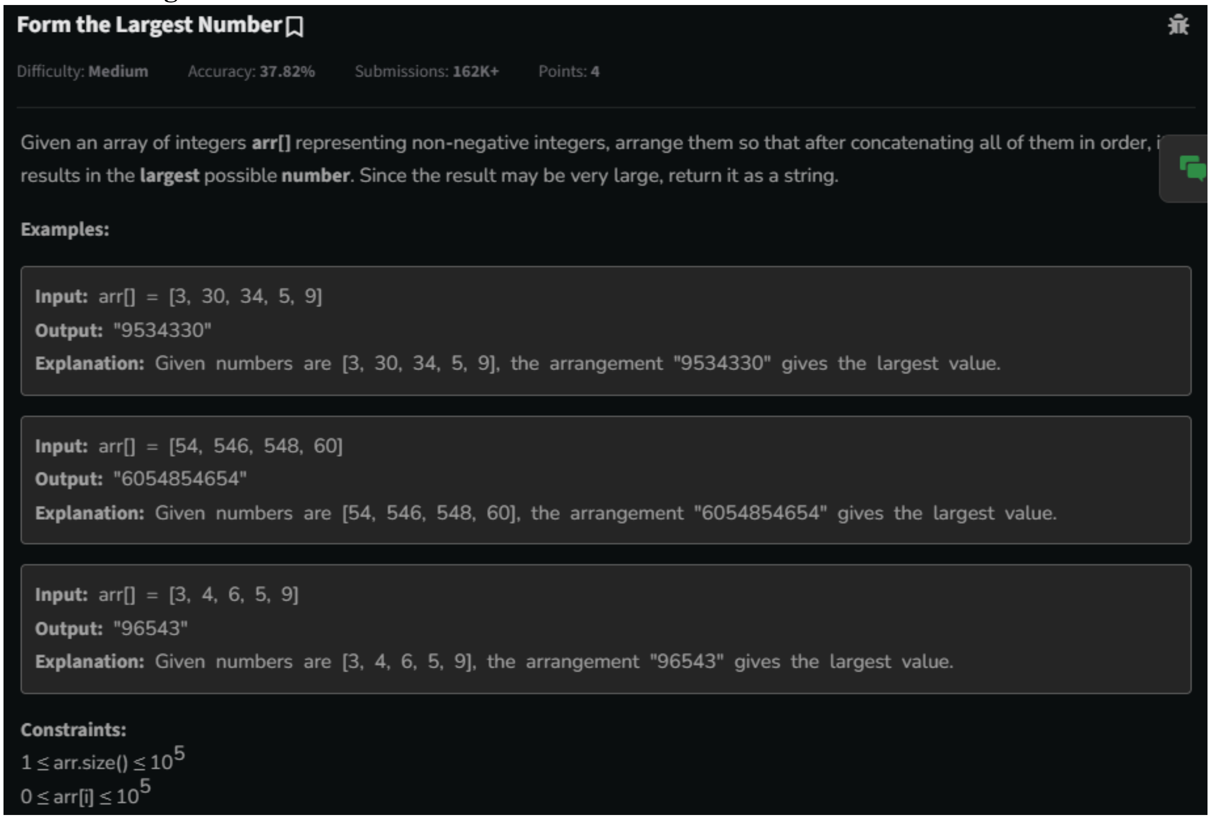
Output:



```
[50, 30, 23]
```

Time Complexity: O(n logk)

6. Form the largest Number



Form the Largest Number

Difficulty: **Medium** Accuracy: **37.82%** Submissions: **162K+** Points: **4**

Given an array of integers **arr[]** representing non-negative integers, arrange them so that after concatenating all of them in order, i
results in the **largest** possible **number**. Since the result may be very large, return it as a string.

Examples:

```
Input: arr[] = [3, 30, 34, 5, 9]
Output: "9534330"
Explanation: Given numbers are [3, 30, 34, 5, 9], the arrangement "9534330" gives the largest value.
```

```
Input: arr[] = [54, 546, 548, 60]
Output: "6054854654"
Explanation: Given numbers are [54, 546, 548, 60], the arrangement "6054854654" gives the largest value.
```

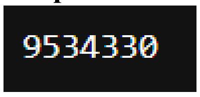
```
Input: arr[] = [3, 4, 6, 5, 9]
Output: "96543"
Explanation: Given numbers are [3, 4, 6, 5, 9], the arrangement "96543" gives the largest value.
```

Constraints:
$1 \le arr.size() \le 10^5$
$0 \le arr[i] \le 10^5$

Code:

```java
import java.util.*;

public class LargestNumber {
    public static String largestNumber(int[] arr) {
        String[] strArr = Arrays.stream(arr)
                                .mapToObj(String::valueOf)
                                .toArray(String[]::new);

        Arrays.sort(strArr, (a, b) -> (b + a).compareTo(a + b));

        if (strArr[0].equals("0")) return "0";

        StringBuilder result = new StringBuilder();
        for (String num : strArr) {
            result.append(num);
        }

        return result.toString();
    }

    public static void main(String[] args) {
        int[] arr = {3, 30, 34, 5, 9};
        System.out.println(largestNumber(arr));
    }
}
```

Output:



```
9534330
```

Time Complexity: O(n logn)