Searching & Sorting

**658. Find K Closest Elements**

public class Solution {

public List<Integer> findClosestElements(int[] arr, int k, int x) {

int size = arr.length;

int more = size - k;

int left = 0;

int right = arr.length - 1;

while (more > 0) {

if ((x - arr[left]) <= (arr[right] - x)) {

right--;

} else {

left++;

}

more--;

}

List<Integer> ans = new ArrayList<>();

for (int i = left; i < left + k; i++) {

ans.add(arr[i]);

}

return ans;

}

}

**349. Intersection of Two Arrays**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

HashSet<Integer> set1 = new HashSet<Integer>();

HashSet<Integer> set2 = new HashSet<>();

for (int i = 0; i < nums1.length; i++) {

set1.add(nums1[i]);

}

for (int i = 0; i < nums2.length; i++) {

set2.add(nums2[i]);

}

set1.retainAll(set2);

int index = 0;

int[] res = new int[set1.size()];

for (int i : set1) {

res[index] = i;

index++;

}

return res;

}

}

**852. Peak Index in a Mountain Array**

class Solution {

public int peakIndexInMountainArray(int[] arr) {

return helper(arr, 0, arr.length - 1);

}

private int helper(int[] arr, int start, int end) {

if (end - start <= 1) return arr[end] >= arr[start] ? end : start;

int mid = start + (end - start) / 2;

int left = helper(arr, start, mid);

int right = helper(arr, mid + 1, end);

return arr[left] < arr[right] ? right : left;

}

}

**767. Reorganize String**

class Solution {

public String reorganizeString(String S) {

Map<Character, Integer> map = new HashMap<>();

PriorityQueue<Character> heap = new PriorityQueue<>((a, b) -> map.get(b) - map.get(a));

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

map.put(S.charAt(i), map.getOrDefault(S.charAt(i), 0) + 1);

}

for (Character key: map.keySet()) {

heap.offer(key);

}

if (S.length() % 2 == 0 && map.get(heap.peek()) > S.length() / 2) return "";

if (S.length() % 2 == 1 && map.get(heap.peek()) > S.length() / 2 + 1) return "";

char[] answ = new char[S.length()];

int oddLength = S.length() - S.length() / 2;

Character curr = heap.poll();

int counter = map.get(curr);

for (int i = 0; i < oddLength; i++) {

answ[i \* 2] = curr;

counter --;

if (counter == 0) {

curr = heap.poll();

counter = map.get(curr);

}

}

int evenLength = S.length() / 2;

for (int i = 0; i < evenLength; i++) {

answ[i \* 2 + 1] = curr;

counter --;

if (counter == 0 && heap.size() > 0) {

curr = heap.poll();

counter = map.get(curr);

}

oddLength--;

}

String res = new String(answ);

return res;

}

}

**Count Numbers of One:**

private int countOnes(int arr[], int start, int end) {

        if (end >= start) {

            int mid = start + (end - start) / 2;

            if ((mid == end || arr[mid + 1] == 0 && (arr[mid] == 1))

                return mid + 1;

            if (arr[mid] == 1) {

                return countOnes(arr, (mid + 1), end);

            return countOnes(arr, start, (mid - 1));

        }

        return 0;

    }