1.Kth smallest element [Amazon]
Given an array of n-elements and an integer k, find the kth smallest element present in an array.
For example:
arr = [40, 25, 68, 79, 52, 66, 89, 97]
k = 2
result = 40

```
import { MaxPriorityQueue } from 'datastructures-js';

function ksmallest(arr, k) {
    const maxHeap = new MaxPriorityQueue();

    arr.forEach(element => {
        maxHeap.push(element);
        if (maxHeap.size() > k) {
            maxHeap.pop()
        }
    });

    return maxHeap.pop()
}

const arr = [40, 25, 68, 79, 52, 66, 89, 97];
const k = 2;

const result = ksmallest(arr, k);
console.log(result)

// Time Complexity : O(nlogn)
```

2.Sort Colors[Amazon]
Given array nums with n objects colored red, white, or blue, sort them in place so that the objects of the same color are adjacent, with the colors in the order red, white, and blue.
We will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
Solve this question without using the library sort function.
For example:
Nums = [2,0,2,1,1,0]
Result = [0,0,1,1,2,2]

```
function sortColor(arr, p, q) {
    if (p < q) {
        const mid = sort(arr, p, q);
        sortColor(arr, p, mid - 1);
        sortColor(arr, mid + 1, q);
    }
}


function sort(arr, p, q) {
    const pivot = arr[p];
    let i = p;
    for (let j = i + 1; j <= q; j++) {
        if (arr[j] < pivot) {
            i += 1;
            swap(arr, i, j);
        }
    }
    swap(arr, i, p);
    return i;
}

const arr = [2, 0, 2, 1, 1, 0];
const p = 0;
const q = arr.length - 1;
console.log(arr)
sortColor(arr, p, q)
console.log(arr)

// Time Compexity : O(nlogn) -> Quick Sort
// Space Complexity : O(1) -> Inplace Sorting
```

3. Kth Largest Element in an array [Facebook]
Given an integer array nums and an integer k, return the kth largest element present in an array.
For example:
arr = [40, 25, 68, 79, 52, 66, 89, 97]
k = 2
result = 89

```
import { MinPriorityQueue } from 'datastructures-js';

function klargest(arr, k) {
    const minHeap = new MinPriorityQueue();

    arr.forEach(element => {
        minHeap.push(element);
        if (minHeap.size() > k) {
            minHeap.pop()
        }
    });

    return minHeap.pop()
}

const arr = [40, 25, 68, 79, 52, 66, 89, 97];
const k = 2;

const result = klargest(arr, k);
console.log(result)

// Time Complexity : O(nlogn)
```

4. Majority Element [Amazon, Google]
Given array nums of size n, return the majority element present in the array.
Assume that the majority element always exists in an array.
For example:
Nums = [2, 2, 1, 1, 1, 2, ,2]
Output: 2

```
function majority(arr) {
    const map = arr.reduce((acc, e) => acc.set(e, (acc.get(e) || 0) + 1), new
Map());

    const sortMap = new Map([...map].sort().reverse());

    const highestPair = [...sortMap.entries()][0];

    return highestPair[0];
}
const arr = [2, 2, 1, 1, 1, 2, 2];
const res = majority(arr);
console.log(res)
// Time Complexity : O(nlogn) -> sorting of array
// Space Complexity : O(n) -> map
```

5. Find of peak element [Facebook]
The peak element is the element that is strictly greater than its neighbors. If an array contains multiple peak elements, return the index of any of the peak elements.
For example: [1,2,3,1]
Output: 2

```
function binarySearch(nums, p, q) {
    if (p <= q) {
        const mid = Math.floor(p + (q - p) / 2);

        if (mid > 0 && mid < nums.length - 1) {
            if (nums[mid - 1] < nums[mid] && nums[mid] > nums[mid + 1]) {
                return mid;
            } else if (nums[mid] < nums[mid + 1]) {
                return binarySearch(nums, mid + 1, q);
            } else if (nums[mid] < nums[mid - 1]) {
                return binarySearch(nums, p, mid - 1);
            }
        } else if (mid === 0) {
            if (nums[0] > nums[mid + 1]) {
                return 0;
            } else {
                return 1;
            }
        } else if (mid === nums.length - 1) {
            if (nums[nums.length - 1] > nums[nums.length - 2]) {
                return nums.length - 1;
            } else {
                return nums.length - 2;
            }
        }
    }
}

const arr = [6, 5, 4, 3, 2, 3, 2];
const p = 0;
const q = arr.length - 1;
const peak = binarySearch(arr, p, q);
console.log(peak)

// Time Complexity : O(logn)
// Space Compelxity : O(logn) (height of tree)
```