Q1. Merge sorted array

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m - 1;
        int j = n - 1;
        int k = m + n - 1;
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }
    }
};
```

Q2. Find peak element

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {

                right = mid;
```

```
        } else {
            left = mid + 1;
        }
    }

    return left;
  }
};
```

Q3.Kth largest element

```
class Solution {
public:
  int findKthLargest(std::vector<int>& nums, int k) {
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;

    for (int num : nums) {
      minHeap.push(num);
      if (minHeap.size() > k) {
        minHeap.pop();
      }
    }

    return minHeap.top();
  }
};
```

Q4. Median od two sorted aaray

```
class Solution {
public:
  double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    int m = nums1.size(), n = nums2.size();
```

```
        vector<int> merged;
        int i = 0, j = 0;

        while (i < m && j < n) {
            if (nums1[i] < nums2[j]) merged.push_back(nums1[i++]);
            else merged.push_back(nums2[j++]);
        }

        while (i < m) merged.push_back(nums1[i++]);
        while (j < n) merged.push_back(nums2[j++]);

        int total = m + n;
        if (total % 2 == 1) return merged[total / 2];
        return (merged[total / 2] + merged[(total / 2) - 1]) / 2.0;
    }
};
```

Q5. Top K frequency element

```
#include <vector>
#include <unordered_map>

class Solution {
public:
    std::vector<int> topKFrequent(std::vector<int>& nums, int k) {
        std::unordered_map<int, int> freqMap;
        for (int num : nums) {
            freqMap[num]++;
        }

        // Bucket array where index represents frequency
        std::vector<std::vector<int>> buckets(nums.size() + 1);
```

```
        for (auto& [num, freq] : freqMap) {
            buckets[freq].push_back(num);
        }

        // Collect top k elements
        std::vector<int> result;
        for (int i = buckets.size() - 1; i >= 0 && result.size() < k; --i) {
            for (int num : buckets[i]) {
                result.push_back(num);
                if (result.size() == k) return result;
            }
        }

        return result;
    }
};
```