# Leetcode 题解 - 排序

* [Leetcode 题解 - 排序](#leetcode-题解---排序)
	+ [快速选择](#快速选择)
	+ [堆](#堆)
		- [1. Kth Element](#1-kth-element)
	+ [桶排序](#桶排序)
		- [1. 出现频率最多的 k 个元素](#1-出现频率最多的-k-个元素)
		- [2. 按照字符出现次数对字符串排序](#2-按照字符出现次数对字符串排序)
	+ [荷兰国旗问题](#荷兰国旗问题)
		- [1. 按颜色进行排序](#1-按颜色进行排序)

## 快速选择

用于求解 **Kth Element** 问题，也就是第 K 个元素的问题。

可以使用快速排序的 partition() 进行实现。需要先打乱数组，否则最坏情况下时间复杂度为 O(N2)。

## 堆

用于求解 **TopK Elements** 问题，也就是 K 个最小元素的问题。使用最小堆来实现 TopK 问题，最小堆使用大顶堆来实现，大顶堆的堆顶元素为当前堆的最大元素。实现过程：不断地往大顶堆中插入新元素，当堆中元素的数量大于 k 时，移除堆顶元素，也就是当前堆中最大的元素，剩下的元素都为当前添加过的元素中最小的 K 个元素。插入和移除堆顶元素的时间复杂度都为 log2N。

堆也可以用于求解 Kth Element 问题，得到了大小为 K 的最小堆之后，因为使用了大顶堆来实现，因此堆顶元素就是第 K 大的元素。

快速选择也可以求解 TopK Elements 问题，因为找到 Kth Element 之后，再遍历一次数组，所有小于等于 Kth Element 的元素都是 TopK Elements。

可以看到，快速选择和堆排序都可以求解 Kth Element 和 TopK Elements 问题。

### 1. Kth Element

215. Kth Largest Element in an Array (Medium)

[Leetcode](https://leetcode.com/problems/kth-largest-element-in-an-array/description/) / [力扣](https://leetcode-cn.com/problems/kth-largest-element-in-an-array/description/)

Input: [3,2,1,5,6,4] and k = 2
Output: 5

题目描述：找到倒数第 k 个的元素。

**排序** ：时间复杂度 O(NlogN)，空间复杂度 O(1)

public int findKthLargest(int[] nums, int k) {
 Arrays.sort(nums);
 return nums[nums.length - k];
}

**堆** ：时间复杂度 O(NlogK)，空间复杂度 O(K)。

public int findKthLargest(int[] nums, int k) {
 PriorityQueue<Integer> pq = new PriorityQueue<>(); // 小顶堆
 for (int val : nums) {
 pq.add(val);
 if (pq.size() > k) // 维护堆的大小为 K
 pq.poll();
 }
 return pq.peek();
}

**快速选择** ：时间复杂度 O(N)，空间复杂度 O(1)

public int findKthLargest(int[] nums, int k) {
 k = nums.length - k;
 int l = 0, h = nums.length - 1;
 while (l < h) {
 int j = partition(nums, l, h);
 if (j == k) {
 break;
 } else if (j < k) {
 l = j + 1;
 } else {
 h = j - 1;
 }
 }
 return nums[k];
}

private int partition(int[] a, int l, int h) {
 int i = l, j = h + 1;
 while (true) {
 while (a[++i] < a[l] && i < h) ;
 while (a[--j] > a[l] && j > l) ;
 if (i >= j) {
 break;
 }
 swap(a, i, j);
 }
 swap(a, l, j);
 return j;
}

private void swap(int[] a, int i, int j) {
 int t = a[i];
 a[i] = a[j];
 a[j] = t;
}

## 桶排序

### 1. 出现频率最多的 k 个元素

347. Top K Frequent Elements (Medium)

[Leetcode](https://leetcode.com/problems/top-k-frequent-elements/description/) / [力扣](https://leetcode-cn.com/problems/top-k-frequent-elements/description/)

Given [1,1,1,2,2,3] and k = 2, return [1,2].

设置若干个桶，每个桶存储出现频率相同的数。桶的下标表示数出现的频率，即第 i 个桶中存储的数出现的频率为 i。

把数都放到桶之后，从后向前遍历桶，最先得到的 k 个数就是出现频率最多的的 k 个数。

public int[] topKFrequent(int[] nums, int k) {
 Map<Integer, Integer> frequencyForNum = new HashMap<>();
 for (int num : nums) {
 frequencyForNum.put(num, frequencyForNum.getOrDefault(num, 0) + 1);
 }
 List<Integer>[] buckets = new ArrayList[nums.length + 1];
 for (int key : frequencyForNum.keySet()) {
 int frequency = frequencyForNum.get(key);
 if (buckets[frequency] == null) {
 buckets[frequency] = new ArrayList<>();
 }
 buckets[frequency].add(key);
 }
 List<Integer> topK = new ArrayList<>();
 for (int i = buckets.length - 1; i >= 0 && topK.size() < k; i--) {
 if (buckets[i] == null) {
 continue;
 }
 if (buckets[i].size() <= (k - topK.size())) {
 topK.addAll(buckets[i]);
 } else {
 topK.addAll(buckets[i].subList(0, k - topK.size()));
 }
 }
 int[] res = new int[k];
 for (int i = 0; i < k; i++) {
 res[i] = topK.get(i);
 }
 return res;
}

### 2. 按照字符出现次数对字符串排序

451. Sort Characters By Frequency (Medium)

[Leetcode](https://leetcode.com/problems/sort-characters-by-frequency/description/) / [力扣](https://leetcode-cn.com/problems/sort-characters-by-frequency/description/)

Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

public String frequencySort(String s) {
 Map<Character, Integer> frequencyForNum = new HashMap<>();
 for (char c : s.toCharArray())
 frequencyForNum.put(c, frequencyForNum.getOrDefault(c, 0) + 1);

 List<Character>[] frequencyBucket = new ArrayList[s.length() + 1];
 for (char c : frequencyForNum.keySet()) {
 int f = frequencyForNum.get(c);
 if (frequencyBucket[f] == null) {
 frequencyBucket[f] = new ArrayList<>();
 }
 frequencyBucket[f].add(c);
 }
 StringBuilder str = new StringBuilder();
 for (int i = frequencyBucket.length - 1; i >= 0; i--) {
 if (frequencyBucket[i] == null) {
 continue;
 }
 for (char c : frequencyBucket[i]) {
 for (int j = 0; j < i; j++) {
 str.append(c);
 }
 }
 }
 return str.toString();
}

## 荷兰国旗问题

荷兰国旗包含三种颜色：红、白、蓝。

有三种颜色的球，算法的目标是将这三种球按颜色顺序正确地排列。它其实是三向切分快速排序的一种变种，在三向切分快速排序中，每次切分都将数组分成三个区间：小于切分元素、等于切分元素、大于切分元素，而该算法是将数组分成三个区间：等于红色、等于白色、等于蓝色。

### 1. 按颜色进行排序

75. Sort Colors (Medium)

[Leetcode](https://leetcode.com/problems/sort-colors/description/) / [力扣](https://leetcode-cn.com/problems/sort-colors/description/)

Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

题目描述：只有 0/1/2 三种颜色。

public void sortColors(int[] nums) {
 int zero = -1, one = 0, two = nums.length;
 while (one < two) {
 if (nums[one] == 0) {
 swap(nums, ++zero, one++);
 } else if (nums[one] == 2) {
 swap(nums, --two, one);
 } else {
 ++one;
 }
 }
}

private void swap(int[] nums, int i, int j) {
 int t = nums[i];
 nums[i] = nums[j];
 nums[j] = t;
}