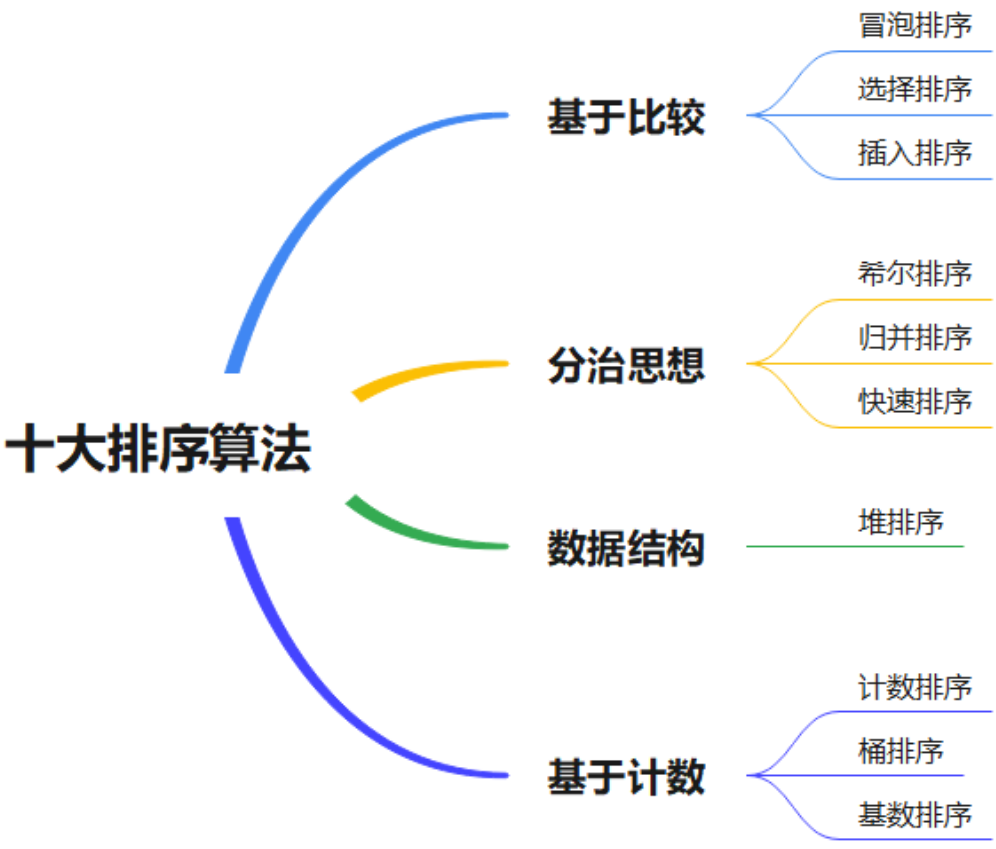
排序算法_面试问答 v1.0

一、基础问题

1.(泛) 请你说一下排序算法

建议**分类**叙述。



2.复杂度与稳定性

2.1十大排序算法复杂度一览

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

速记Tip：

- 非原地——归并、（计数、桶、基数）
- 不稳定——选择、希尔、快排、堆排

  > 记忆：**快希选堆**不稳定（**快些选堆**朋友来嗨吧!）

- 平均时间复杂度——（冒泡、选择、插入）｜（希尔 归并 快排 堆排）｜（计数、桶、基数）

2.2最坏情况下时间复杂度最小的排序算法?

① 归并排序

② 堆排序

最坏时间复杂度都为O(nlogn)

2.3 XXXX与初始序列无关的排序有哪些?

（注：这块知识 *CSDN* 上各种不同的答案，仅供参考！我整理了一遍脑子已经开始乱了痛了）

- （1）**移动次数**与初始序列无关的是：归并排序、基数排序

  > 记忆：归基
  >
  > 分析：因为归并和基数都是**非原地**的，都得移动。
  >
  > ---
  >
  > 选择排序（有关）——特例：全部排好，无需移动

> 插入排序（有关）——就像抓牌，如果全部排好，每次放最右边就行；如果没有排好，每次要移动一些牌再插入。注：折半插入（即采用二分法查找插入位置）可以减少比较次数。

- （2）**比较次数**与初始序列无关是：选择排序、基数排序

> 记忆：选基
>
> 分析：
>
> 选择排序（无关）——哪怕全部排好，也要比较(n-1)!次，选出最值并（交换后）固定
>
> 基数排序（无关）——没有相互比较的过程，无关
>
> ---
>
> 堆排序（有关）——比较次数和**下沉深度**相关。虽然一个元素是从底部拉上来的，但不代表这个元素一定会接着沉到底部，如果沉到中间就停止下沉的话，比较次数就少了。

- （3）**时间复杂度**与初始序列无关的是：选择排序、基数排序、归并排序、堆排序

> 记忆：选基归堆（旋即归队）
>
> 分析：平均、最好、最坏时间复杂度相同 -> 与初始序列无关

- （4）**排序趟数**与初始序列无关的是：插入排序、选择排序、基数排序

> 分析：除了两个**例外**：快排、优泡
>
> ① 快速排序中，排序趟数（即递归深度）与关键字选择（即初始状态）有关；
>
> ② 优化后的冒泡排序中，排序趟数与初始状态有关；

> Q：什么是**优化后的冒泡排序**?
>
> A：查阅CSDN后得知，这里的“**优化**”指：
>
> 在原本n-1轮的两两比较中，如果其中一轮发现已经全部有序，则**提前退出。**
>
> **优化后的冒泡排序**的**核心代码**：

```
bool sorted = false;
while (!sorted) {
  sorted = true;
  for (int i=1; i<n; i++) {
         if (a[i] > a[i+1]) {
              temp = a[i];
              a[i] = a[i+1];
              a[i+1] = temp;
              sorted = false;
         }
     }
     n--;
}
```

3.手撕排序

3.1手撕归并排序

一句话原理：采用递归思想，不断拆分后按序合并。

```
/*
 * 归并排序
 * 特点：时间：O(nlogn)、空间：O(n)--非原地
 */
public class Merge {
    private static void sort(int[] nums, int start, int end) {
        if (start < end) {  // 如果start==end，数组中只有一个元素，则不用排了
            // 分成两半
            int mid = (start + end) / 2;
            // 分别排序
            sort(nums, start, mid);
            sort(nums, mid + 1, end);
            // 进行合并
            merge(nums, start, end);
        }
    }

    // 辅助函数：合并两个有序数组
    private static void merge(int[] nums, int left, int right) {
        int[] tmp = new int[nums.length];       // 辅助数组tmp，暂存有序结果
        int mid = left + (right - left) / 2;
        int p1 = left;
        int p2 = mid + 1;
        int k = left;                           // 注意：k初始化为left（因为tmp与
nums位置要一一对应）
        while (p1 <= mid && p2 <= right) {
            if (nums[p1] <= nums[p2])
                tmp[k++] = nums[p1++];
            else
                tmp[k++] = nums[p2++];
        }
        while (p1 <= mid)
            tmp[k++] = nums[p1++];
        while (p2 <= right)
            tmp[k++] = nums[p2++];
        for (int i = left; i <= right; i++)
            nums[i] = tmp[i];
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums, 0, nums.length - 1);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

3.2手撕快速排序

一句话原理：采用分治思想，选取中轴元素，小的放左大的放右，增大移动距离 以 减小比较次数。

```
/*
 * 快速排序
 * 特点：时间：O(nlogn)、空间：O(logn)、非稳定
 */
public class Quick {
    private static void sort(int[] nums, int start, int end) {
        // 终止条件
        if (start >= end)   return;

        // 核心代码
        // temp记录标记点，left和right在移动；
        // 此时left位置相当于空出，可以放置
        int left = start;
        int right = end;
        int temp = nums[left];
        // 抛三个球
        while (left < right) {
            while (left < right && nums[right] >= temp) // 位置符合
                right--;
            nums[left] = nums[right];                   // 位置不符：放到左边空位，并把自己当前位置空出
            while (left < right && nums[left] <= temp)  // 位置符合
                left++;
            nums[right] = nums[left];                   // 位置不符：放到右边空位，并把自己当前位置空出
        }
        nums[left] = temp;                              // 退出while时left==right，这是个空位，把temp标记点的数放入

        // 递归：现在以left/right为边界，左边是比它小的，右边是比它大的
        sort(nums, start, left - 1);
        sort(nums, left + 1, end);
    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums, 0, nums.length - 1);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

3.3手撕堆排序

一句话原理：构造大顶堆（升序），把堆顶元素（最大）的固定到末尾，将剩余元素继续建堆、固定...

```
/*
 * 堆排序
 * 特点：时间：O(nlogn)、非稳定
 */
public class Heap {
    public static void sort(int[] list) {
        // （1）构造初始堆
        // 从第一个非叶子节点（倒数第二行最后一个）开始调整
        // 左右孩子节点中较大的交换到父节点中
        // 注意这里i是自底往上的!
        for (int i = (list.length) / 2 - 1; i >= 0; i--) {
            headAdjust(list, list.length, i);
        }
        // （2）排序
        // 将最大的节点list[0]放在堆尾list[i]
        // 然后从根节点重新调整
        // 由于（1）的存在，这里每次最大值都会在堆顶那三个里面产生（这个想法不知道对不对，严谨性待证明）
        // 这里的i代表len，即每次把最后一个排好的忽略掉
        for (int i = list.length - 1; i >= 1; i--) {
            int temp = list[0];
            list[0] = list[i];
            list[i] = temp;
            headAdjust(list, i, 0);
        }
    }

    // 辅助函数：调整堆
    // 参数说明：list代表整个二叉树、len是list的长度、 i代表三个中的根节点
    private static void headAdjust(int[] list, int len, int i) {
        int index = 2 * i + 1; // 左孩子

        // 这步while的意义在于把较小的沉下去，把较大的提上来
        //使得最大值总在最顶上三个里面产生（这个想法不知道对不对，严谨性待证明）
        while (index < len) {
            // （1）index指向左右孩子较大的那个
            if (index + 1 < len) { // 说明还有右孩子
                if (list[index] < list[index + 1]) {
                    index = index + 1;
                }
            }
            // （2）比较交换大孩子和根节点
            if (list[index] > list[i]) {
                //交换
                int temp = list[i];     // temp暂存，现在根空了
                list[i] = list[index];  // 更新根，现在list[index]空了
                list[index] = temp;
                //更新
                i = index;
                index = 2 * i + 1;      // index指向孩子的孩子，继续
            } else {
                break;
```

```
            }
        }

    }

    // 测试代码
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 3, 7, 8, 2, 9, 6, 0, 5, 4 };
        sort(nums);
        for (int num : nums)
            System.out.print(num + " ");
    }
}
```

3.4手撕其他排序

打个广告，嘿嘿，详见@Lemon 的 `代码小抄：十大排序算法（Java实现）v1.1.pdf`

二、拓展问题

1.奇偶排序

- **背景**：因为奇数对彼此独立，每一刻可以用不同的处理器比较和交换**奇数对**（或偶数对），可以实现非常快速的排序。
- **思路**：利用**并行**特性，轮流对奇数对和偶数对排序（注：冒泡排序的两两有重叠，奇偶排序每对之间数据无关）
- **举例**：参考https://blog.csdn.net/zhizhengguan/article/details/95897884
- **代码**：

```
public class OddEven {
    private static void sort(int[] arr) {
        if (arr == null || arr.length < 2)
            return;

        boolean flag = false;    // 用于判断是否排序完成

        while (!flag) {
            flag = true;         // 先默认已经排序完了
            // 进行 奇数对 排序
            for (int i = 0; i < arr.length; i += 2) {
                if ((i + 1) < arr.length && arr[i] > arr[i + 1]) {
                    int t = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = t;
                    flag = false;    // 有变动，未排完
                }
            }
            // 进行 偶数对 排序
            for (int i = 1; i < arr.length; i += 2) {
                if ((i + 1) < arr.length && arr[i] > arr[i + 1]) {
                    int t = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = t;
                    flag = false;    // 有变动，未排完
                }
            }
        }
    }

    //测试代码
    public static void main(String[] args) {
        int[] arr = new int[] { 1, 6, 2, 4, 3, 5, 8, 0, 9, 7 };
        sort(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

三、应用问题

1.快排思想：找第k大的数

- **快排**：采用分治思想，选取中轴元素，小的放左大的放右，增大移动距离 以 减小比较次数。
- **思路**：
  - 找TopK大的元素：即**从大到小**排序后，下标为**k-1**的元素
  - 快排while循环的每一轮，可以确定中轴元素的**最终下标**
    - 每进行一轮，比较 本轮固定好的中轴元素下标 和 k-1
    - 根据情况分类，选择继续递归左半边/右半边
- **代码**：

```
/*
 * 快排思想找TopK大的数
 */
public class QuickTopK {
    //找TopK大的元素：即从大到小排序后，下标为k-1的元素
    static int findTopK(int[] nums, int k, int left, int right) {
        int pos = quick(nums, left, right); //进行一轮快排，pos为本轮固定的中轴元
素下标
        if (k-1 < pos) {        //情况一：k-1在pos左边
            return findTopK(nums, k, left, pos - 1);
        } else if (k-1 > pos) { //情况二：k-1在pos右边
            return findTopK(nums, k, pos + 1, right);
        } else {                //情况三：k-1即pos位置
            return nums[pos];
        }
    }

    // 单轮快排函数：以nums[left]为中轴，进行一轮快速排序（从大到小）
    static int quick(int[] nums, int left, int right) {
        int temp = nums[left];  //中轴元素
        while (left < right) {
            //下面的while循环，从右找第一个比中轴元素大的元素下标
            while (left < right && nums[right] <= temp) {
                right--;
            }
            //放到左边
            nums[left] = nums[right];
            //下面的while循环，从左找第一个比中轴元素小的元素下标
            while (left < right && nums[left] >= temp) {
                left++;
            }
            //放到右边
            nums[right] = nums[left];
        }
        nums[left] = temp;
        return left;
    }

    //测试代码：
    public static void main(String[] args) {
        int[] nums = new int[] { 1, 6, 2, 4, 3, 5, 8, 0, 9, 7 };
```

```
        // 从大到小：9，8，7，6，5，4，3，2，1，0
        System.out.println(findTopK(nums, 3, 0, nums.length - 1));
        // nums中第 3 大的应该为：7
    }
}
```

变式：快排思想：找无序数组的中位数

- 是“快排思想：找第k大的数”的**变式**
- 找中位数 -> 找第k大的数，其中`k=nums.length/2+1`

> 写在最后：
>
> 感谢录友们提供的面试问题、感谢卡哥帮助宣传！
>
> 我也是菜鸟，也是学一点写一点，如有错误，烦请指正。
>
> 内啥，看完了回主题给我点个赞吧？嘿嘿。