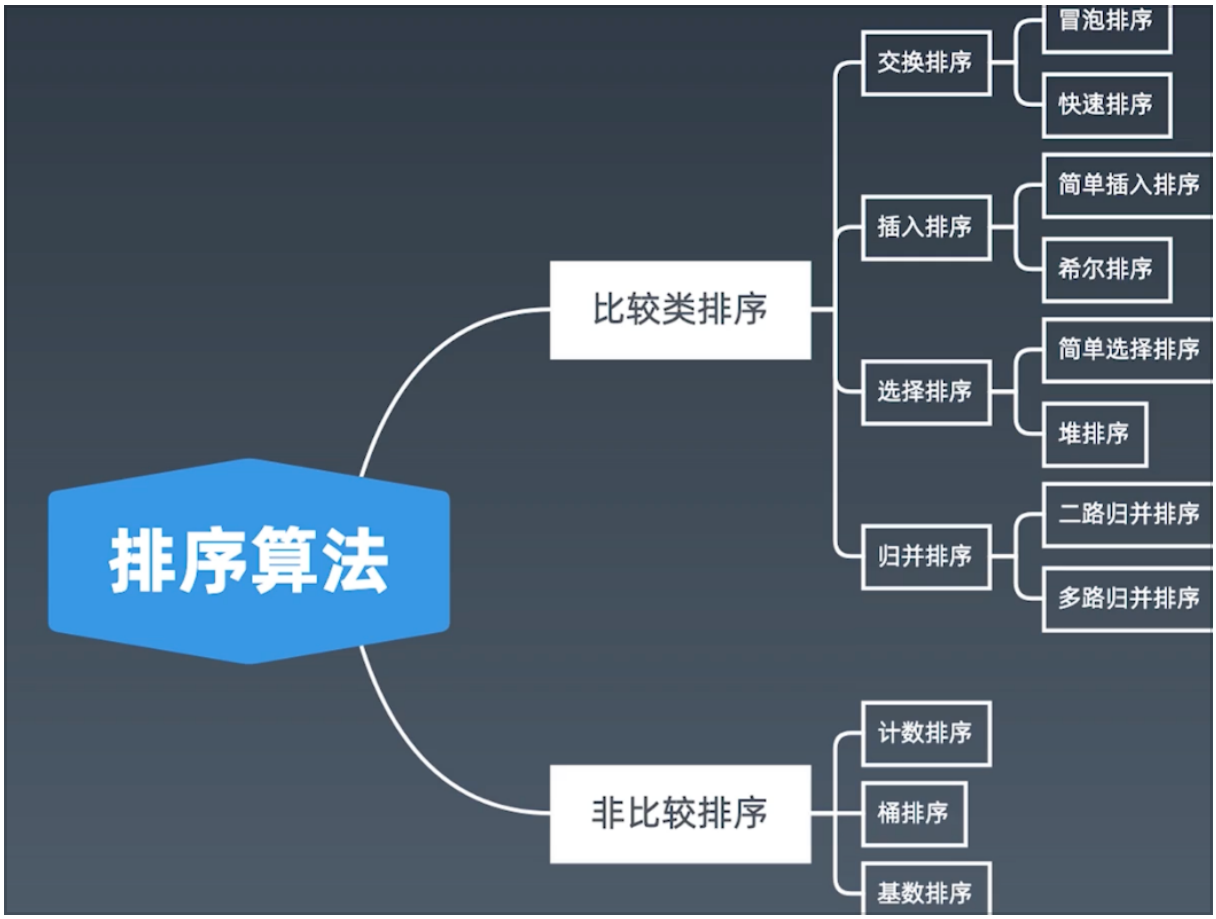
23. 排序算法

十大经典排序算法
9 种经典排序算法可视化动画
6 分钟看完 15 种排序算法动画展示

算法分类

1. 比较类排序
   通过比较来决定元素间的相对次序，由于其时间复杂度不能突破 O(nlogn),因此也称为非线性时间比较类排序.

2. 非比较类排序
   不通过比较来决定元素间的相对次序，它可以突破基于比较排序的时间下届，以线性时间运行，因此也
   称为线性时间非比较类排序.



时间复杂度

面试着重于 $O(log_2n)$ 的排序： 堆排序、快速排序、归并排序.

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

初级排序 - O（n^2）

1. 选择排序（Selection Sort）
   每次找最小值，然后放到待排序数组额度起始位置.

2. 插入排序（Insertion Sort）
   从前往后逐步构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入.

3. 冒泡排序（ Bubble Sort ）
   嵌套循环，每次查看相邻的元素，如果逆序，则交换.

高级排序 - O（N * LogN）

快速排序（ Quick Sort ）

- 数组取标杆 pivot，将小元素放 pivot 左边，大元素放右边，然后依次对右边和右边的子数组继续快排，以达到整个序列有序。

- 注意：正常情况下数组的 prepend 操作的时间复杂度是 O(n), 但是可以进行特殊优化到 O(1). 采用的方式是申请稍大一些的内存空间，然后在数组最开始预留一部分空间，然后 prepend 的操作则是把头下标前移一个位置即可.

快速排序代码示例

Java Code

```java
public static void quickSort(int[] array, int begin, int end) {
    if (end <= begin) return;
    int pivot = partition(array, begin, end);
    quickSort(array, begin, pivot - 1);
    quickSort(array, pivot + 1, end);
}

static int partition(int[] a, int begin, int end) {
    // pivot: 标杆位置, counter: 小于pivot的元素的个数
    int pivot = end, counter = begin;
    for (int i = begin; i < end; i++) {
        if (a[i] < a[pivot]) {
            int temp = a[counter]; a[counter] = a[i]; a[i] = temp;
            counter++;
        }
    }
    int temp = a[pivot]; a[pivot] = a[counter]; a[counter] = temp;
    return counter;
}
```

调用方式： `quickSort(a, 0, a.length - 1)`

归并排序（Merge Sort）– 分治

1. 把长度为 n 的输入序列分成两个长度为 n/2 的子序列；
2. 对这两个子序列分别采用归并排序；
3. 将两个排序好的子序列合并成一个最终的排序序列.

归并排序代码示例

Java Code

```java
public static void mergeSort(int[] array, int left, int right) {

    if (right <= left) return;
    int mid = (left + right) >> 1; // (left + right) / 2

    mergeSort(array, left, mid);
    mergeSort(array, mid + 1, right);
    merge(array, left, mid, right);

}
```

```java
public static void merge(int[] arr, int left, int mid, int right) {

    int[] temp = new int[right - left + 1]; // 中间数组
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        temp[k++] = arr[i] <= arr[j] ? arr[i++] : arr[j++];
    }

    while (i <= mid)   temp[k++] = arr[i++];
    while (j <= right) temp[k++] = arr[j++];

    for (int p = 0; p < temp.length; p++) {
        arr[left + p] = temp[p];
    }
    // 也可以用 System.arraycopy(a, start1, b, start2, length)
}
```

> 归并 和 快排 具有相似性，但步骤顺序相反
>
> 归并：先排序左右子数组，然后合并两个有序子数组
> 快排：先调出左右子数组，然后对于左右子数组进行排序

堆排序（Heap Sort）– 插入堆 O(logN), 取最大/小值 O（1）

1. 数组元素一次建立小顶堆；
2. 依次取堆顶元素，并删除；
3. 推荐使用系统的 priority_queue.

堆排序代码示例

C++ Code

```
void heap_sort(int a[], int len) {

    priority_queue<int,vector<int>,greater<int> > q;

    for(int i = 0; i < len; i++) {
        q.push(a[i]);
    }

    for(int i = 0; i < len; i++) {
        a[i] = q.pop();
    }
}
```

Java Code

```
static void heapify(int[] array, int length, int i) {
    int left = 2 * i + 1, right = 2 * i + 2;
    int largest = i;

    if (left < length && array[left] > array[largest]) {
        largest = leftChild;
    }
    if (right < length && array[right] > array[largest]) {
        largest = right;
    }

    if (largest != i) {
        int temp = array[i]; array[i] = array[largest]; array[largest] = temp;
        heapify(array, length, largest);
    }
}

public static void heapSort(int[] array) {
    if (array.length == 0) return;

    int length = array.length;
    for (int i = length / 2-1; i >= 0; i-)
        heapify(array, length, i);

    for (int i = length - 1; i >= 0; i--) {
        int temp = array[0]; array[0] = array[i]; array[i] = temp;
        heapify(array, i, 0);
    }
}
```

首先上面的代码是一个子函数

特殊排序 - O (n)

- 计数排序（Counting Sort）
  计数排序要求输入的数据必须是有确定范围的整数。将输入的数据值转化为键存储在额外开辟的数组空间中；然后依次把计数大于 1 的填充回原数组

- 桶排序（Bucket Sort）
  桶排序 (Bucket sort)的工作的原理：假设输入数据服从均匀分布，将数据分到有限数量的桶里，每个桶再分别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排）。

- 基数排序（Radix Sort）
  基数排序是按照低位先排序，然后收集；再按照高位排序，然后再收集；依次类推，直到最高位。有时候有些属性是有优先级顺序的，先按低优先级排序，再按高优先级排序。

#Algorithm/Part II : Theory/Algorithm#