十大排序算法分析

排序算法是面试中常问的算法，大厂中排序算法问的深度很深，对排序算法的理解有多个层次

- 知道常用算法的写法，了解各种算法的时间复杂度，空间复杂度和稳定性
- 了解每种算法的性能瓶颈
- 对于每种算法知道如何优化
- 知道每种算法的应用场景

1. 选择排序

- 算法思想

  将一组数据分为两部分，前面是已排序部分，后面是未排序部分，初始状态可认为位置 0 为已排序部分 (数组下标从0开始)，其余为未排序部分，每一次都从未排序部分选择一个最小元素放在已排序部分的末尾，然后已排序部分增加一个元素，未排序部分减少一个元素，直到数据全部有序。

- 时间复杂度

  选择排序无论数据初始是何种状态，均需要在未排序元素中选择最小或最大元素与未排序序列中的首尾元素交换，因此它的最好、最坏、平均时间复杂度均为 O(n^2)。

- 空间复杂度

  空间复杂度为O(1)

- 稳定性

  直接选择排序是不稳定的。因为每次遍历比较完后会使用本次遍历选择的最小元素和无序区的第一个元素交换位置，所以如果无序区第一个元素后面有相同元素的，则可能会改变相同元素的相对顺序（**稳定性：能保证两个相等的数,经过排序之后,其在序列的前后位置顺序不变**）

- 优化思路

  1. 每次查找时不仅找出最小值，还找出最大值，分别插到前面和后面，可以减少一半的查询时间。
  2. 如果数组元素重复率高，可以考虑使用辅助空间在每一次循环的时候，将本次选择的数及相同元素的索引记录下来，一起处理。

- 代码

```
public class SelectionSort {
    public static void main(String[] args) {
        int[] arr={1,4,3,2,3,2,1,2,6};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    public static void sort(int[] arr){
        for (int i=0;i<arr.length-1;i++){
            int minPos=i;
            for (int j = minPos+1; j < arr.length; j++) {
                if (arr[j]<arr[minPos]){
                    minPos=j;
                }
            }
            swap(arr,i,minPos);
        }
    }
```

```
    private static void swap(int[] arr,int i,int j){
        int temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
    }
}
```

2. 冒泡排序

- 算法思想

  通过比较相邻的两个元素，将大的元素或者小的元素交换到后面，这样越大或者越小的元素都会交换到数组的后端。

- 时间复杂度

  时间复杂度是O(n^2)

- 空间复杂度

  空间复杂度为O(1)

- 稳定性

  稳定

- 优化思路
  - 用一个计数器记录交换的次数，当某一轮交换次数为0则表示数组已经有序，那么就不用继续进行了。
  - 记录最后一次交换的位置，该位置之后没有进行交换说明是有序的了，下一轮只用遍历该位置即可。
- 代码

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr={1,4,23,2,1,1,23,2,1,8};
        sort(arr);
        for (int i=0;i<arr.length;i++){
            System.out.print(arr[i]+" ");
        }
    }
    public static void sort(int[] arr){
        for (int i=0;i<arr.length-1;i++){
            for (int j=0;j<arr.length-i-1;j++){
                if (arr[j]>arr[j+1]){
                    swap(arr,j,j+1);
                }
            }
        }
    }
    private static void swap(int[] arr,int i,int j){
        int temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
    }
}
```

3. (直接)插入排序

- 算法思想

  每趟将一个元素，按照其关键字的大小插入到它前面已经排序的子序列中，依此重复，直到插入全部元素。

- 时间复杂度

  时间复杂度为O(n^2)

- 空间复杂度

  空间复杂度为O(1)

- 稳定性

  稳定

- 优化思路
  - 希尔排序
  - 二分查找插入排序
    - 二分查找插入排序的原理：是直接插入排序的一个变种，区别是：在有序区中查找新元素插入位置时，为了减少元素比较次数提高效率，采用二分查找算法进行插入位置的确定。

- 代码

```
public class InsertionSort {
    public static void main(String[] args) {
        int[] arr={1,5,4,3,2,6,7,8,9};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    public static void sort(int[] arr){
        for (int i=0;i<arr.length;i++){
            int temp=arr[i];
            int j=i-1;
            for (;j>=0&&arr[j]>temp;j--){
                arr[j+1]=arr[j];
            }
            arr[j+1]=temp;
        }
    }
}
```

4. 希尔排序

- 算法思路

  希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的关键词越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止。

- 时间复杂度

  时间复杂度取决于增量序列的选择O(n^1.3)

- 空间复杂度
- 稳定性

不稳定

- 代码

```
public class ShellSort {
    public static void main(String[] args) {
        int[] arr={1,3,2,4,6,5,7,9,8,0};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }

    public static void sort(int [] arr){
        // 增量序列
        int[] ds={1,3,7};
        for (int i=ds.length-1;i>=0;i--){
            insertSort(arr,1);
        }
    }

    /**
     * 按增量分组进行直接插入排序
     * @param arr 数组
     * @param d 增量
     */
    private static void insertSort(int[] arr,int d){
        for (int i=0;i<d;i++){
            for (int j=i;j<arr.length;j+=d){
                int k=j-d;
                int temp=arr[j];
                for (;k>=i&&arr[k]>temp;k-=d){
                    arr[k+d]=arr[k];
                }
                arr[k+d]=temp;
            }
        }
    }
}
```

5. 堆排序

- 算法思路

  堆是这样一种数据结构，首先堆是一个完全二叉树，其父节点一定大于其所有的子节点。堆排序就是利用堆这种数据结构进行排序，首先构造一个堆，堆顶元素就是最大或最小的元素，把他与堆的最后一个元素交换，这样堆顶元素就调整到了排序后的顺序，而堆的元素个数减少了一个且结构发生了变化，只需要重新调整堆就可以了。循环这个步骤数组就变成有序的了。

- 时间复杂度

  nlogn -> n表示第几轮 logn为调整堆的时间复杂度

- 空间复杂度

  O(1)

- 稳定性

不稳定

- 代码

```
public class HeapSort {
    public static void main(String[] args) {
        int[] arr={1,4,3,2,5,7,6,8,9,0};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    public static void sort(int[] arr){
        // 首先要对整个数组进行heapify操作
        for (int i=arr.length/2;i>=0;i--){
            heapify(arr,arr.length,i);
        }
        for (int i=1;i<=arr.length;i++){
            swap(arr,0,arr.length-i);
            heapify(arr,arr.length-i,0);
        }
    }
    private static void swap(int[] arr,int i,int j){
        int temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
    }
    private static void heapify(int[] arr,int len,int i){
        int left=i*2+1;
        int right=i*2+2;
        int max=i;
        if (left<len&&arr[left]>arr[max]){
            max=left;
        }
        if (right<len&&arr[right]>arr[max]){
            max=right;
        }
        if (max!=i){
            swap(arr,i,max);
            heapify(arr,len,max);
        }
    }
}
```

6. 归并排序

- 算法思路

  基于二路归并算法，将数组分成两个部分，对每一部分递归采取归并排序，这样数组两个部分就排好序了，对于两个已排好序的数组，只需要进行二路归并就可以得到一个有序的数组。

- 时间复杂度

  O(nlogn)

- 空间复杂度

  O(n)

- 稳定性

稳定

- 优化思路
  - 原地归并

    因为用归并将一个大数组排序时，需要进行多次归并，而且每次归并会都创建一个新数组来存储排序结果会带来问题。由于原地归并排序不需要额外的空间，所以空间复杂度为O(1)。
  - 当递归到规模足够小时，利用插入排序
- 代码

```
public class MergeSort {
    private static int[] tempArr;
    public static void main(String[] args) {
        int[] arr={1,2,5,4,3,6,9,7,8,0};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    public static void sort(int[] arr){
        tempArr=new int[arr.length];
        sort(arr,0,arr.length-1);
    }

    private static void sort(int[] arr,int start,int end){
        if (start==end){
            return;
        }
        int mid=start+(end-start)/2;
        sort(arr,start,mid);
        sort(arr,mid+1,end);
        merge(arr,start,mid,end);
    }

    private static void merge(int[] arr,int start,int mid,int end){
        System.arraycopy(arr,0,tempArr,0,arr.length);
        int i=start;
        int j=mid+1;
        int k=start;
        while (i<=mid&&j<=end){
            if (tempArr[i]<tempArr[j]){
                arr[k++]=tempArr[i++];
            }else{
                arr[k++]=tempArr[j++];
            }
        }
        while (i<=mid){
            arr[k++]=tempArr[i++];
        }
        while (j<=end){
            arr[k++]=tempArr[j++];
        }
    }
}
```

7. 快速排序

- 算法思路

  快速排序使用分治法策略来把一个序列分为较小和较大的2个子序列，然后递归地排序两个子序列。具体步骤是选择一个元素作为基准元素，将比它小的元素放置在它的左边，将比它大的元素排它的后面，这样基准元素的位置就确定了，然后分别对基准元素左边和右边的序列进行相同操作，每次都能将一个元素排放到正确的位置。

- 时间复杂度

  O(nlogn) 递归的过程O(n) partition过程O(n)->O(nlogn)

- 空间复杂度

  空间复杂度为logn

- 优化思路

- 代码

```
public class QuickSort {
    public static void main(String[] args) {
        int[] arr={1,4,2,3,6,5,7,9,8,0};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    public static void sort(int[] arr){
        quickSort(arr,0,arr.length-1);
    }

    private static void quickSort(int[] arr,int start,int end){
        if (start>=end){
            return;
        }
        int pos=partition(arr,start,end);
        quickSort(arr,start,pos-1);
        quickSort(arr,pos+1,end);
    }

    private static int partition(int[] arr,int start,int end){
       int left=start-1;
       int k=start;
       while (k<=end){
           if(arr[k]>arr[start]){
               k++;
           }else{
               swap(arr,++left,k++);
           }
       }
       swap(arr,start,left);
       return left;
    }
    private static void swap(int[] arr,int i,int j){
        int temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
    }
}
```

8. 桶排序

- 算法思路

  这是一种算法思想，基于非比较的排序算法，时间复杂度比较低但是一般需要额外的空间，将数组中的元素分配到不同的桶，桶与桶之间是有顺序的，桶的内部元素无序，对每个不为空的桶进行排序（可以使用别的排序算法），然后将不为空的桶中的元素进行输出就可以完成排序。

- 代码

```
public class BucketSort {
    public static void main(String[] args) {
        int[] arr={1,3,2,5,4,6,9,8,7,0};
        sort(arr);
        for (int i : arr) {
            System.out.printf(i+" ");
        }
    }

    public static void sort(int[] arr){
        int max=arr[0];
        int min=arr[0];
        for (int value : arr) {
            min = Math.min(value, min);
            max = Math.max(value, max);
        }
        ArrayList<Integer>[] buckets=new ArrayList[max/10-min/10+1];
        for (int i=0;i<buckets.length;i++){
            buckets[i]=new ArrayList<Integer>();
        }
        for (int value : arr) {
            buckets[(value-min)/10].add(value);
        }
        for (int i=0;i<buckets.length;i++){
            if (buckets[i].size()!=0){
                Collections.sort(buckets[i]);
            }
        }
        int k=0;
        for (int i=0;i<buckets.length;i++){
            if (buckets[i].size()!=0){
                for (int j=0;j<buckets[i].size();j++){
                    arr[k++]=buckets[i].get(j);
                }
            }
        }
    }
}
```

9. 基数排序

- 算法思想

  基数排序是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。是桶排序思想的一种。是一种多关键字排序。

- 代码

```
public class RadixSort {
    public static void main(String[] args) {
        int[] arr={123,43,231,24,56,432,124};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }

    public static void sort(int[] arr){
        int[] result=new int[arr.length];
        int[] count=new int[10];
        int maxLen=maxLen(arr);
        // 分别对十位，个位，百位.....进行排序
        for (int i=0;i<=maxLen;i++){
            int temp= (int) Math.pow(10,i);
            for (int value : arr) {
                count[value / temp % 10]++;
            }
            for (int j=1;j<count.length;j++){
                count[j]+=count[j-1];
            }
            for (int j=arr.length-1;j>=0;j--){
                result[--count[arr[j]/temp%10]]=arr[j];
            }
            System.arraycopy(result,0,arr,0,arr.length);
            Arrays.fill(count,0);
        }
    }

    private static int maxLen(int[] arr){
        int max=arr[0];
        for (int i : arr) {
            max=Math.max(i,max);
        }
        int len=0;
        while (max!=0){
            max/=10;
            len++;
        }
        return len;
    }
}
```

10. 计数排序

- 算法思路

  准备一个桶，桶的长度为n，且待排序数组的范围都在0到n-1之间，这样数组中的数i就可以放入桶的第i项，桶只要记住每个数字出现的次数，然后扫描桶就可以得到排序后的序列，适合数组范围不大的元素。

- 代码

```
public class CountSort {
    public static void main(String[] args) {
```

```
        int[] arr={1,3,2,4,5,6,8,7,9,0};
        sort(arr);
        for (int i : arr) {
            System.out.print(i+" ");
        }
    }
    private static void sort(int[] arr){
        int[] count=new int[10];// 计数数组，数组的数的范围落在0-count.length-1
        for (int value : arr) {
            count[value]++;
        }
        int k=0;
        for (int i=0;i<count.length;i++){
            while (count[i]>0){
                count[i]--;
                arr[k++]=i;
            }
        }
    }
}
```

常用排序一览表(需要熟记，并且时间复杂度空间复杂度怎么算的，为什么稳定/不稳定都得知道)

中文名称	英文名称	平均 时间复杂度	最坏 时间复杂度	最好 时间复杂度	空间复杂度	稳定性
选择排序	Selection	n^2	n^2	n^2	1	不稳
冒泡排序	Bubble	n^2	n^2	n	1	稳
插入排序	Insertion	n^2	n^2	n	1	稳
堆排序	heap	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	1	不稳
希尔排序	Shell	$n^{1.3}$	n^2	n	1	不稳
归并排序	Merge	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	n	稳
快速排序	Quick	$n\log_2 n$	n^2	$n\log_2 n$	$n\log_2 n$	不稳
桶排序	Bucket	$n+k$	n^2	n	$n+k$	稳
计数排序	Counting	$n+k$	$n+k$	$n+k$	$n+k$	稳
基数排序	Radix	$n*k$	$n*k$	$n*k$	$n+k$	稳

常见面试问题总结

- 告诉你某种算法，比如很明确的问你某种算法然后不断追问，如快速排序（字节跳动一面）
    1. 说一下快速排序的思想（先考你知不知道思想）
    2. 快速排序的时间复杂度/空间复杂度/稳定(考一下你会不会分析算法或者说这个算法是不是你背下来的实际上你并不知道或不理解)
    3. 快速排序不适合什么样的数据（考你某种算法的缺点，进一步看你理不理解这个算法）
    4. 如果要排上面的算法，怎么优化（基于某个问题，要求你进行优化，这类问题最深应该就问到这一步了，后面也问不下去了）
- 不明确告诉你某种算法，问你排某一类特征的数据应该用什么算法排序(字节跳动一面)
    - 这类问题难在要逆向思考，你得对所有算法都得足够的熟悉
    - 问题：排一组比较有序的数组用什么算法，为什么可以用这种算法

总结

- 一共有四种不稳定的排序算法，选择排序，堆排序，希尔排序，快速排序，其他的都是稳定的排序。
- 数据的特征大概有：数据基本有序，数据范围不大，数据逆序
- 分析数据的时间复杂度必须知道他的循环嵌套关系，以及每层循环的时间复杂度，优化也是这么思考的，看每一层循环这么优化
- 空间复杂度比较好分析，但是注意递归的情况，比如快速排序的空间时间复杂度不是O(1)而是logn(上面的表示错误的)
- 要理解算法的稳定性有什么影响，不稳定为什么不好？