八大排序算法：

一.各排序算法时间复杂度对比表

各类排序算法时间复杂度和空间复杂度对比表

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数。

*修正：快排的空间复杂度为O(logn)

二.插入排序

1.直接插入排序：将一个已有记录插入到排好序的有序表中，正序：比较n-1次，无需移动，反序：比较（n+2)(n-1）/2次，记录移动（n+4）(n-1)/2次，时间复杂度为$O(n^2)$次，空间复杂度O(1)（如果设置了监视哨，则可以利用设置监视哨的方式避免数组越界并且减少比较次数）

java实现代码：

```
public static void InsertOrder(int[] arr){
        for(int i=1;i<arr.length;i++){
            if(arr[i]<arr[i-1]){
                int temp=arr[i];
                int j;
                arr[i]=arr[i-1];
                for(j=i-2;j>=0&&temp<arr[j];j--)
```

```
                arr[j+1]=arr[j];
            arr[j+1]=temp;
        }
    }

}
public static void main(String[] args) {
    int[] arr={49,38,65,97,76,13,27,49};
    InsertOrder(arr);
    for(int i:arr){
        System.out.println(i);
    }
```

2.希尔排序：将待排序记录按照增量值分成增量个子序列，然后分别在子序列中进行直接插入排序，直至增量值最后减为1，在进行直接插入排序时注意序列中间隔变为增量值，希尔排序时间复杂度与增量有关，最好为O(n1.3），最差为O（n^2），空间复杂度为O(1);），不稳定 （原因：因为是分组进行直接插入排序，原来相同的两个数字可能会被分到不同的组去，可能会使得后面的数字会排到前面，使得两个相同的数字排序前后位置发生变化。 不稳定举例: 4 3 3 2 按2为增量分组，则第二个3会跑到前面）

```
public static void shellOrder(int [] arr,int dk){
    while(dk!=0){
        for(int i=0;i<dk;i++){
            for(int j=i+dk;j<arr.length;j+=dk){
                if(arr[j]<arr[j-dk]){
                    int temp=arr[j];
```

```
                int h;
                for(h=j-dk;h>=0&&temp<arr[h];h-=dk)
                    arr[h+dk]=arr[h];
                arr[h+dk]=temp;
            }
        }
    }
    dk=dk/2;
  }
}
```

三.选择排序

1.简单选择排序

首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后每次从剩余未排序元素中继续寻找最小（大）元素放到已排序序列的末尾。以此类推，直到所有元素均排序完毕 时间复杂度：最坏:O(n²) 最好: O(n²) 平均: O(n²) 空间复杂度：O(1) 稳定性：不稳定 （例如数组 2 2 1 3 第一次选择的时候把第一个2与1交换使得两个2的相对次序发生了改变）

```
public static void SselctSort(int[] arr){
    for(int i=0;i<arr.length;i++){
        int index=i;
        for(int j=i+1;j<arr.length;j++){
            if(arr[j]<arr[index])
                index=j;
        }
```

```
            if(i!=index){
                int temp=arr[i];
                arr[i]=arr[index];
                arr[index]=temp;
            }
        }
    }
```

2.堆排序

思想：堆排序是利用堆的性质进行的一种选择排序，先将排序元素构建一个大顶堆，每次从堆中取出最大的元素并调整堆，将该取出的最大元素放到已排好序的序列前面。 时间复杂度：最坏:O(nlogn) 最好: O(nlogn) 平均: O(nlogn) 空间复杂度：O(1) 稳定性：不稳定（例如 5 10 15 10。 如果堆顶5先输出，则第三层的10(最后一个10)的跑到堆顶，然后堆稳定，继续输出堆顶，则刚才那个10跑到前面了，所以两个10排序前后的次序发生改变）

```
public static void heapSort(int[] arr){
        for(int i=(arr.length-1)>>1;i>=0;i--)
            HeapAdjust(arr,i,arr.length-1);
        for(int j=arr.length-1;j>0;j--){
            int temp=arr[0];
            arr[0]=arr[j];
            arr[j]=temp;
            HeapAdjust(arr,0,j-1);
        }

    }
```

```
public static void HeapAdjust(int[] arr,int s,int m){
    for(int i=2*s+1;i<=m;i=i*2+1){
        int rc=arr[s];
        if(i<m&&arr[i]<arr[i+1])
            ++i;
        if(rc>=arr[i])
            break;
        int temp=arr[s];
        arr[s]=arr[i];
        arr[i]=temp;
        s=i;
        }
}
```

四.交换排序

1.冒泡排序

两两比较，然后分别将最大的数依次放置，直至最后一趟过程中已没有交换操作，时间复杂度为O(n^2）空间复杂度为O(1）稳定（相邻的关键字两两比较，如果相等则不交换，所以排序前后的相等数字相对位置不变）

```
//冒泡排序（设置尾边界为最后一次交换处，结束循环的标志位不再交换）
public static void BubbleSort(int[] arr){
    int flag=arr.length;
    while(flag>0){
        int k=flag;
        flag=0;
```

```
            for(int j=0;j<k;j++){
                if(arr[j+1]<arr[j]){
                    int temp=arr[j+1];
                    arr[j+1]=arr[j];
                    arr[j]=temp;
                    flag=j;
                }
            }
        }
    }
```

2.快速排序

通过设置枢纽，一趟排序后在枢纽位置处分割（分成比它大的部分以及比它小的部分），然后再分别进行排序（递归思想）改进思想：可以针对于一次划分方法，在low-1或者high+1的同时进行冒泡操作，分别设置标志表示是否移动过程中是否交换过，若没有交换过，则无需再对其进行某低端或高端子序列排序 基准元素的选择对快速排序的性能影响很大，所有一般会想打乱排序数组选择第一个元素或则随机地从后面选择一个元素替换第一个元素作为基准元素。时间复杂度：最好: O(nlogn) 最坏：$O(n^2)$ 平均: O(nlogn) 空间复杂度：O(nlogn)用于方法栈 稳定性：不稳定 （快排会将大于等于基准元素的关键词放在基准元素右边，加入数组 1 2 2 3 4 5 选择第二个2 作为基准元素，那么排序后 第一个2跑到了后面，相对位置发生变化 ）

```
    public static void QuickSort(int[] arr,int low,int high){
        if(low<high){
            int pivortloc=Partition( arr,low,high);
            QuickSort(arr,low,pivortloc-1);
```

```
            QuickSort(arr,pivortloc+1,high);
        }
    }
    public static int Partition(int[] arr,int low,int high){
        int key=arr[low];
        while(low<high){
            while(low<high&&key<=arr[high])
                high--;
            arr[low]=arr[high];
            while(low<high&&arr[low]<=key)
                low++;
            arr[high]=arr[low];
        }
        arr[low]=key;
        return low;
    }
```

五.归并排序

归并排序采用了分治算法，首先递归将原始数组划分为若干子数组，对每个子数组进行排序。然后将排好序的子数组递归合并成一个有序的数组。时间复杂度：最坏:O(nlogn) 最好: O(nlogn) 平均: O(nlogn) 空间复杂度：O(n) 稳定性：稳定

```
public static void MergeSort(int[] arr,int left,int right){
        if(left<right){
            int mid=(left+right)/2;
```

```
            MergeSort(arr,left,mid);
            MergeSort(arr,mid+1,right);
            Merge(arr,left,mid,right);
        }

    }
    public static void Merge(int[] arr,int left,int middle,int
right){
    int[] temp=new int[arr.length];
    int i=left;
    int j=middle+1;
    int k=left;
    while(i<=middle&&j<=right){
        if(arr[i]<arr[j])
            temp[k++]=arr[i++];
        else
            temp[k++]=arr[j++];
    }
    while(i<=middle)
        temp[k++]=arr[i++];
    while(j<=right)
        temp[k++]=arr[j++];
    for(int m=left;m<=right;m++)
```

```
        arr[m]=temp[m];
}
```

六.基数排序(桶排序)

排序通过“分配”和“收集”过程来实现排序，若待排元素为数字，首先根据数字的个位的数将数字放入0-9号桶中，然后将所有桶中所盛数据按照桶号由小到大，桶中由底至顶依次重新收集起来，得到新的元素序列，然后再分别对十位、百位这些高位采用同样的方式分配收集，直到最高位完成收集最终得到有序数组，若待排元素为单词，可以根据单词位数进行分解，然后根据单词在低位的字母分别单词分配到A-Z桶中，然后将所有桶中所盛单词按照桶号由小到大，桶中由底至顶依次重新收集起来，得到新的元素序列，然后再分别对高位采用同样的方式分配收集，直到最高位完成收集得到最终有序序列。

中心思想：将待排元素看成由若干关键字复合而成，从最低数位关键字起，按关键字的不同值将元素分配到不同桶中后再收集之，如此重复d次。其中基指的就是桶号的取值范围

时间复杂度：最坏:O(d(r+n)) 最好:O(d(r+n)) 平均: O(d(r+n)) 空间复杂度：O(dr+n)　n个记录，d个关键码，关键码的取值范围为r (基为r)　稳定性：稳定 (基数排序基于分别排序，分别收集)

```
public static void RedixSort(int[] arr){
        int max=arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i]>max) {
                        max=arr[i];
            }
        }
        int d=1;
        while (max>0) {
                max/=10;
```

```
    d*=10;
}
int n=1;
int k=0;
int[][] bucket=new int[10][arr.length];
int[] order=new int[10];
while(n<d){
    for(int num:arr){
        int digit=(num/n)%10;
        bucket[digit][order[digit]]=num;
        order[digit]++;
    }
    for(int i=0;i<10;i++){
                    if(order[i]!=0){
            for(int j=0;j<order[i];j++)
            {
              arr[k]=bucket[i][j];
              k++;
            }
        }
        order[i]=0;
    }
    k=0;
```

```
        n*=10;
    }
}
```