排序相关

1选择排序

算法步骤

算法步骤

1. 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；
2. 再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾；
3. 重复第2步，直到所有元素均排序完毕。

```
private static void selectSort(int[] a) {
    for (int i = 0; i < a.length-1; i++) {
        // 这个地方的标志位含义是后面哪一个比当前这个小，把这个后面的标志位置记录出来
        int num = i;
        for (int j = i+1; j < a.length; j++) {
            // 排序这里<length是因为要比较到最后，前面<-1是因为最后一个不用拿出来再做比较
            if (a[j] < a[num]) {
                // 比当前的小就是要交换的，找出最小的那个
                num = j;
            }
        }
        if (num != i){
            int temp = a[i];
            a[i] = a[num];
            a[num] = temp;
        }
    }
    System.out.println(Arrays.toString(a));
}
```

2插入排序

（1）、从第一个元素开始，该元素可以认为已经被排序；
（2）、取出下一个元素，在已经排序的元素序列中从后向前扫描；
（3）、如果该元素（已排序）大于新元素，将该元素移到下一位置；
（4）、重复步骤3，直到找到已排序的元素小于或者等于新元素的位置；

(5)、将新元素插入到该位置后；

(6)、重复步骤2~5

```
    private static void insertSort(int[] a) {
        for (int i = 1; i < a.length; i++) {
            // 第一个数据默认排序，从第二个数据开始比较
            int temp = a[i];
            // 把第二个数据（要插入的数据）拿出来。用来交换
            int j;
            // 这个是要比较的列表
            for (j = i; j >0 && a[j-1] > temp ; j--) {
                // 循环条件。a[j-1]已经排好序的数据都比要排的数据大a[j] = a[i] = temp，则一直往前
                // 因为已经比要插入的数据取出来了，所以空间不变直接往后移动
                a[j] = a[j-1];
            }
            // 直到找到了a[j-1] 数据没有a[j]即a[i]即temp大的,直接插在他后面，因为他后面其他数据都往
            a[j] = temp;
        }
        System.out.println(Arrays.toString(a));
    }
```

3快速排序

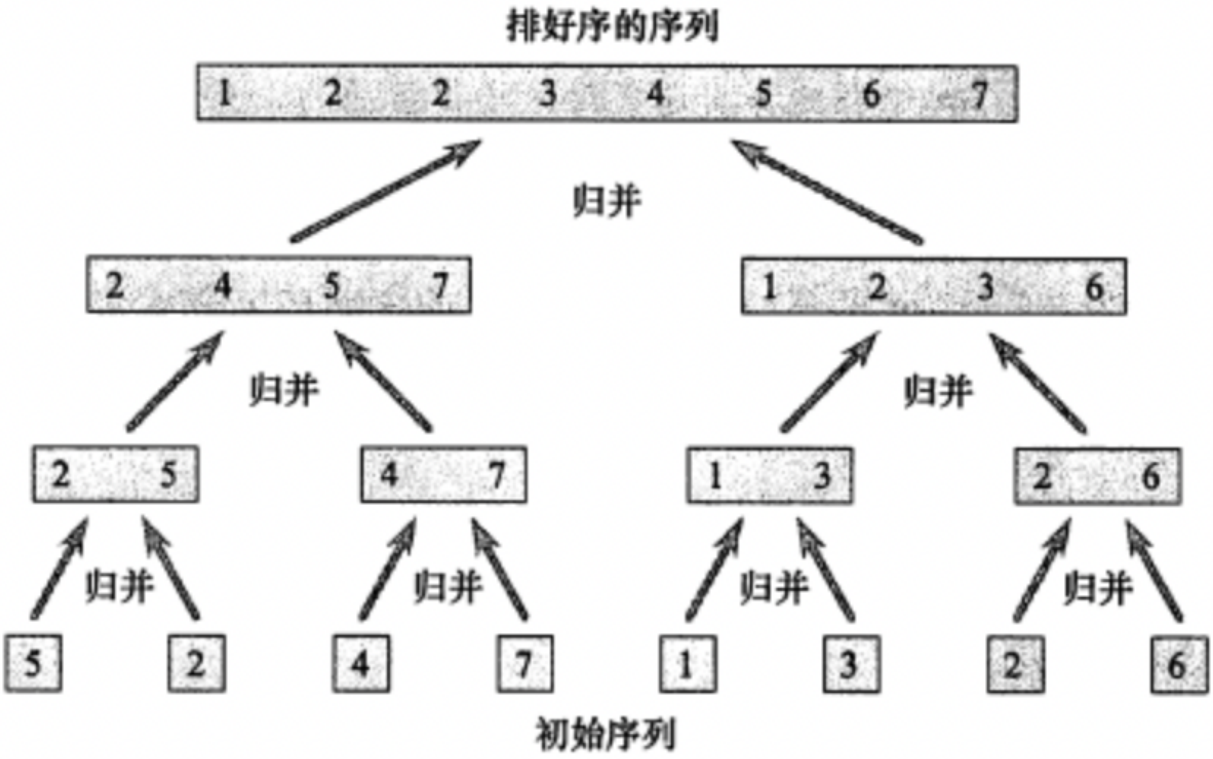
1. 左右两把指针往中间找，找到不属于自己区间的就停下来等和另一个指针交换，交换完成继续往前往后走，
2. 最终两把指针相遇，交换指针和基数，此时指针（基数）左边都比基数小，右边都比基数大，
3. 继续递归（递归不用包含中间值即可）
4. https://blog.csdn.net/shujuelin/article/details/82423852

```
private static void quicksort(int[] a) {
        quicksort(a, 0, a.length-1);
    }

    // 左右两把指针往中间找，找到不属于自己区间的就停下来等和另一个指针交换，交换完成继续往前往后走，最
    private static void quicksort(int[] a, int low, int high) {
        if (low > high) {
            return;
        }
        // 基准位置
        int temp = a[low];
        int i = low;
        int j = high;
        while (i < j) {
            // 右边
            while (a[j] >= temp && i<j) {
                j--;
            }
            // 左边
            while (a[i] <= temp && i<j) {
                i++;
            }
            if (i < j) {
                int trans = a[j];
                a[j] = a[i];
                a[i] = trans;
            }
        }
        // 走到最后i和j指向同一个节点
        a[low] = a[i];
        a[i] = temp;
        quicksort(a, low, i-1);
        quicksort(a, i+1, high);


        System.out.println(Arrays.toString(a));
    }
```

4归并


排好序的序列
1 2 2 3 4 5 6 7
归并
2 4 5 7
1 2 3 6
归并
归并
2 5
4 7
1 3
2 6
归并
归并
归并
归并
5
2
4
7
1
3
2
6
初始序列

```java
import java.util.Arrays;
//归并排序
public class MergeSort_06 {
	public static void main(String[] args) {
		int a[]={3,44,38,5,47,15,36,26,27,2,46,4,19,50,48};
		//int a[]={5,2,4,7,1,3,2,2};
		int temp[]=new int[a.length];
		mergesort(a,0,a.length-1,temp);
		System.out.println(Arrays.toString(a));
	}
	private static void mergesort(int[] a, int left, int right, int[] temp) {
		//分解
		if (left<right) {
			int mid=(left+right)/2;
			//向左递归进行分解
			mergesort(a, left, mid, temp);
			//向右递归进行分解
			mergesort(a, mid+1, right, temp);
			//每分解一次便合并一次
			merge(a,left,right,mid,temp);
		}
	}
	/**
	 *
	 * @param a  待排序的数组
	 * @param left  左边有序序列的初始索引
	 * @param right 右边有序序列的初始索引
	 * @param mid   中间索引
	 * @param temp  做中转的数组
	 */
	private static void merge(int[] a, int left, int right, int mid, int[] temp) {
		int i=left; //初始i，左边有序序列的初始索引
		int j=mid+1;//初始化j，右边有序序列的初始索引（右边有序序列的初始位置即中间位置的后
		int t=0;//指向temp数组的当前索引，初始为0

		//先把左右两边的数据（已经有序）按规则填充到temp数组
		//直到左右两边的有序序列，有一边处理完成为止
		while (i<=mid && j<=right) {
			//如果左边有序序列的当前元素小于或等于右边的有序序列的当前元素，就将左边的元
			if (a[i]<=a[j]) {
				temp[t]=a[i];
				t++;//索引向后移
				i++;//i后移
			}else {
				//反之，将右边有序序列的当前元素填充到temp数组中
				temp[t]=a[j];
				t++;//索引向后移
				j++;//j后移
			}
		}
```

```
                //把剩余数据的一边的元素填充到temp中
                while (i<=mid) {
                        //此时说明左边序列还有剩余元素
                        //全部填充到temp数组
                        temp[t]=a[i];
                        t++;
                        i++;
                }
                while (j<=right) {
                        //此时说明左边序列还有剩余元素
                        //全部填充到temp数组
                        temp[t]=a[j];
                        t++;
                        j++;
                }
                //将temp数组的元素复制到原数组
                t=0;
                int tempLeft=left;
                while (tempLeft<=right) {
                        a[tempLeft]=temp[t];
                        t++;
                        tempLeft++;
                }
        }

}
```

5堆排

第一步：构建初始堆buildHeap， 使用sink(arr,i, length)调整堆顶的值;

第二步：将堆顶元素下沉 目的是将最大的元素浮到堆顶来，然后使用sink(arr, 0,length)调整;

```java
public class Heap_Sort_07 {
        public static void main(String[] args) {
                int a[]={3,44,38,5,47,15,36,26,27,2,46,4,19,50,48};
                sort(a);
                System.out.println(Arrays.toString(a));
        }
    public static void sort(int[] arr) {
        int length = arr.length;
        //构建堆
        buildHeap(arr,length);
        for ( int i = length - 1; i > 0; i-- ) {
            //将堆顶元素与末位元素调换
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            //数组长度-1 隐藏堆尾元素
            length--;
            //将堆顶元素下沉 目的是将最大的元素浮到堆顶来
            sink(arr, 0,length);
        }
    }
    private static void buildHeap(int[] arr, int length) {
        for (int i = length / 2; i >= 0; i--) {
            sink(arr,i, length);
        }
    }

    private static void sink(int[] arr, int index, int length) {
        int leftChild = 2 * index + 1;//左子节点下标
        int rightChild = 2 * index + 2;//右子节点下标
        int present = index;//要调整的节点下标

        //下沉左边
        if (leftChild < length && arr[leftChild] > arr[present]) {
            present = leftChild;
        }

        //下沉右边
        if (rightChild < length && arr[rightChild] > arr[present]) {
            present = rightChild;
        }

        //如果下标不相等 证明调换过了
        if (present != index) {
            //交换值
            int temp = arr[index];
            arr[index] = arr[present];
            arr[present] = temp;

            //继续下沉
```

```
            sink(arr, present, length);
        }
    }
}
```

6冒泡

```
import java.util.Arrays;
//冒泡排序
public class BubbleSort_01 {
        public static void main(String[] args) {
                int a[]={3,44,38,5,47,15,36,26,27,2,46,4,19,50,48};
                //记录比较次数
                int count=0;
                //i=0,第一轮比较
                for (int i = 0; i < a.length-1; i++) {
                        //第一轮，两两比较
                        for (int j = 0; j < a.length-1-i; j++) {
                                if (a[j]>a[j+1]) {
                                        int temp=a[j];
                                        a[j]=a[j+1];
                                        a[j+1]=temp;
                                }
                                count++;
                        }
                }
                System.out.println(Arrays.toString(a));//[2, 3, 4, 5, 15, 19, 26, 27, 3
                System.out.println("一共比较了："+count+"次");//一共比较了：105次
        }
}
```