排序

排序算法比较

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

- 快速排序递归

```
package sort.quicksort;

import java.util.Arrays;

/**
* 快速排序递归
*/
public class QuickSortRecursion {

    private static void quicksort(int[] arr) {
        int n = arr.length;
        quicksort(arr, 0, n - 1);
    }

    private static void quicksort(int[] arr, int start, int end) {
        if (arr == null || start >= end) {
        return;
    }
    int i = start, j = end;
    int pivotKey = arr[start];
    while (i < j) {
        while (i < j && arr[j] >= pivotKey) {
        j--;
        }
        if (i < j) {
            arr[i++] = arr[j];
```

```
        }
        while (i < j && arr[i] <= pivotKey) {
            i++;
        }
        if (i < j) {
            arr[j--] = arr[i];
        }
    }
    arr[i] = pivotKey;
    quicksort(arr, start, i - 1);
    quicksort(arr, i + 1, end);
}

    public static void main(String[] args) {
        int[] arr = {10, 5, 2, 3};
        quicksort(arr);
        System.out.println(Arrays.toString(arr)); // [2, 3, 5, 10]
    }
}
```

- 归并排序递归

```
package sort.mergesort;

import java.util.Arrays;

/**
 * 归并排序递归
 */
public class MergeSortRecursion {
    public static void sort(int[] arr) {
        int n = arr.length;
        sort(arr, 0, n - 1);
    }

    // 递归使用归并排序,对arr[l...r]的范围进行排序
    private static void sort(int[] arr, int l, int r) {
        if (l >= r)
            return;

        int mid = (l + r) / 2;
        sort(arr, l, mid);
        sort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }

    // 将arr[l...mid]和arr[mid+1...r]两部分进行归并
    private static void merge(int[] arr, int l, int mid, int r) {
        int[] aux = Arrays.copyOfRange(arr, l, r + 1);

        // 初始化，i指向左半部分的起始索引位置l；j指向右半部分起始索引位置mid+1
        int i = l, j = mid + 1;
        for (int k = l; k <= r; k++) {

            if (i > mid) {  // 如果左半部分元素已经全部处理完毕
                arr[k] = aux[j - l];
                j++;
```

```
            } else if (j > r) {   // 如果右半部分元素已经全部处理完毕
                arr[k] = aux[i - l];
                i++;
            } else if (aux[i - l] - aux[j - l] < 0) {  // 左半部分所指元素 < 右半部分所指元素
                arr[k] = aux[i - l];
                i++;
            } else {  // 左半部分所指元素 >= 右半部分所指元素
                arr[k] = aux[j - l];
                j++;
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 5, 2, 3};
        sort(arr);
        System.out.println(Arrays.toString(arr)); // [2, 3, 5, 10]
    }
}
```

- 堆排序

```
package sort.heapsort;

import java.util.Arrays;

public class HeapSort {

    public static void heapsort(int[] arr) {
        int n = arr.length;
        // 注意，此时我们的堆是从0开始索引的
        // 从(最后一个元素的索引-1)/2开始
        // 最后一个元素的索引 = n-1
        for (int i = (n - 1 - 1) / 2; i >= 0; i--)
            shiftDown(arr, n, i);

        for (int i = n - 1; i > 0; i--) {
            swap(arr, 0, i);
            shiftDown(arr, i, 0);
        }
    }

    // 交换堆中索引为i和j的两个元素
    private static void swap(int[] arr, int i, int j) {
        int t = arr[i];
        arr[i] = arr[j];
        arr[j] = t;
    }

    // 优化的shiftDown过程，使用赋值的方式取代不断的swap，
    // 该优化思想和我们之前对插入排序进行优化的思路是一致的
    private static void shiftDown(int[] arr, int n, int k) {

        int e = arr[k];
        while (2 * k + 1 < n) {
            int j = 2 * k + 1;
```

```
            if (j + 1 < n && arr[j + 1] - arr[j] > 0)
                j += 1;

            if (e - arr[j] >= 0)
                break;

            arr[k] = arr[j];
            k = j;
        }
        arr[k] = e;
    }

    // 测试 HeapSort
    public static void main(String[] args) {
        int[] arr = {10, 5, 2, 3};
        heapsort(arr);
        System.out.println(Arrays.toString(arr)); // [2, 3, 5, 10]
    }
}
```