2009853E-I011-0084

毛一凡

# 回溯排序

public class MergeSort {  
  
 public static void main(String[] args) {  
 int arr[] = { 8, 4, 5, 7, 1, 3, 6, 2,37,43}; *//* System.*out*.println("排序前" + Arrays.*toString*(arr));  
   
 int temp[] = new int[arr.length]; *//归并排序需要一个额外空间  
 mergeSort*(arr, 0, arr.length - 1, temp);  
   
   
 System.*out*.println("归并排序后=" + Arrays.*toString*(arr));  
 }  
   
 分解的方法  
 public static void mergeSort(int[] arr, int start, int end, int[] temp) {  
 if(start < end) {  
 int mid = (start + end) / 2; *//中间索引  
 //向左递归进行分解  
 mergeSort*(arr, start, mid, temp);  
 *//向右递归进行分解  
 mergeSort*(arr, mid + 1, end, temp);  
 *//合并  
 merge*(arr, start, mid, end, temp);  
   
 }  
 }  
   
 合并的方法public static void merge(int[] arr, int start, int mid, int end, int[] temp) {  
   
 int i = start;

int j = mid + 1;

int t = 0; *// 指向temp数组的当前索引* while (i <= mid && j <= end) {*//继续* if(arr[i] <= arr[j]) {  
 temp[t] = arr[i];  
 t += 1;  
 i += 1;  
 } else { *//反之,将右边有序序列的当前元素，填充到temp数组* temp[t] = arr[j];  
 t += 1;  
 j += 1;  
 }  
 }  
   
 while( i <= mid) { *//左边的有序序列还有剩余的元素，就全部填充到temp* temp[t] = arr[i];  
 t += 1;  
 i += 1;   
 }  
   
 while( j <= end) { *//右边的有序序列还有剩余的元素，就全部填充到temp* temp[t] = arr[j];  
 t += 1;  
 j += 1;   
 }  
   
   
 *//(三)* t = 0;  
 int tempLeft = start; *//*

while(tempLeft <= end) {  
 arr[tempLeft] = temp[t];  
 t += 1;  
 tempLeft += 1;  
 }  
   
 }  
  
}

主要依靠分解方法递归将数组分为小的部分，靠合并方法将两子部分按大小合并

## 实现

# 文本, 信件 描述已自动生成

# 简单排序

public class QuickSort {  
  
 public static void main(String[] args) {  
 int[] arr = {8, 4, 5, 7, 1, 3, 6, 2,37,43};  
   
 System.*out*.println("排序前" + Arrays.*toString*(arr));  
   
 *quickSort*(arr, 0, arr.length-1);  
 System.*out*.println("arr=" + Arrays.*toString*(arr));  
 }  
  
 public static void quickSort(int[] arr,int left, int right) {  
 int l = left; *//左下标* int r = right; *//右下标  
 //pivot 中轴值* int pivot = arr[(left + right) / 2];  
 int temp = 0; *//临时变量，作为交换时使用  
 //while循环的目的是让比pivot 值小放到左边  
 //比pivot 值大放到右边* while( l < r) {   
 *//在pivot的左边一直找,找到大于等于pivot值,才退出* while( arr[l] < pivot) {  
 l += 1;  
 }  
 *//在pivot的右边一直找,找到小于等于pivot值,才退出* while(arr[r] > pivot) {  
 r -= 1;  
 }

if( l >= r) {  
 break;  
 }  
   
 *//交换* temp = arr[l];  
 arr[l] = arr[r];  
 arr[r] = temp;  
   
 }  
   
 *// 如果 l == r, 必须l++, r--, 否则为出现栈溢出的报错？* if (l == r) {  
 l += 1;  
 r -= 1;  
 }  
 *//向左递归* if(left < r) {  
 *quickSort*(arr, left, r);  
 }  
 *//向右递归* if(right > l) {  
 *quickSort*(arr, l, right);  
 }  
   
   
 }  
}

## .实现

## 