广东金融学院实验报告

课程名称：算法分析与设计

装订线

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验编号  及实验名称 | 实验2 | | | 系 别 | 互联网金融与信息工程 |
| 姓 名 | 谢潮锋 | 学 号 | 17B543157 | 班 级 | 1715431 |
| 实验地点 | 实503 | 实验日期 | 2019.9.26 | 实验时数 | 2 |
| 指导教师 | 郭艺辉 | 同组其他成员 | 无 | 成 绩 |  |
| 1. 实验目的及要求   1) 掌握递归与分治算法的基本思想以及基本原理。  2) 掌握使用分治算法求解问题的一般特征以及步骤。  3) 掌握分治算法设计方法以及复杂性分析方法。   1. 掌握二分搜索术以及快速排序问题描述、算法设计思想、算法设计过程以及程序编码实现。 | | | | | |
| 1. 实验环境及相关情况（包含使用软件、实验设备、主要仪器及材料等）   1) 操作系统：Windows操作系统  2) 开发工具：Eclipse、JDK  3) 开发语言：Java | | | | | |
| 1. 实验内容及步骤（包含简要的实验步骤流程） 2. 设计一个递归算法生成5个元素{1, 2, 3, 4, 5}的全排列，并将算法编程实现。 3. 给定数组a[0 : 8]={1, 8, 12, 15, 16, 21, 30, 35, 39}。采用二分搜索算法完成下述任务： 4. 查找是否有元素30，若有返回元素在数组中的位置；如没有返回无此元素； 5. 查找是否有元素20，若有返回元素在数组中的位置；如没有返回无此元素； 6. 当待搜索元素*x*=10不在数组中时，返回小于*x*的最大元素位置*i*和大于*x*的最小元素位置*j*。 7. （选作）给定两个数组*a*[0 : 6]={10, 15, 21, 22, 36, 38}与*a*[0 : 8]={2, 3, 8,16, 24, 27, 38, 40}，采用二分法找出这两个有序线性序列第7小的元素。 8. 给定数组*a*[ ]={8，4，3，7，1，5，6，2}，使用快速排序算法对其进行排序，将算法编程实现。 9. 快速排序算法的性能与划分是否对称有关，设计随机化的快速排序算法解决划分对称性问题，将算法编程实现；并回答：对于一个随机化算法，为什么只分析其平均情况下的性能，而不分析其最坏情况下的性能？ | | | | | |
| 1. 实验结果（包括程序或图表、结论陈述、数据记录及分析等，可附页）  * **第1题 运行截图及代码如下：**   [ 1 2 3 4 5 ] [ 1 2 3 5 4 ] [ 1 2 4 3 5 ] [ 1 2 4 5 3 ] [ 1 2 5 4 3 ] [ 1 2 5 3 4 ]  [ 1 3 2 4 5 ] [ 1 3 2 5 4 ] [ 1 3 4 2 5 ] [ 1 3 4 5 2 ] [ 1 3 5 4 2 ] [ 1 3 5 2 4 ]  [ 1 4 3 2 5 ] [ 1 4 3 5 2 ] [ 1 4 2 3 5 ] [ 1 4 2 5 3 ] [ 1 4 5 2 3 ] [ 1 4 5 3 2 ]  [ 1 5 3 4 2 ] [ 1 5 3 2 4 ] [ 1 5 4 3 2 ] [ 1 5 4 2 3 ] [ 1 5 2 4 3 ] [ 1 5 2 3 4 ]  [ 2 1 3 4 5 ] [ 2 1 3 5 4 ] [ 2 1 4 3 5 ] [ 2 1 4 5 3 ] [ 2 1 5 4 3 ] [ 2 1 5 3 4 ]  [ 2 3 1 4 5 ] [ 2 3 1 5 4 ] [ 2 3 4 1 5 ] [ 2 3 4 5 1 ] [ 2 3 5 4 1 ] [ 2 3 5 1 4 ]  [ 2 4 3 1 5 ] [ 2 4 3 5 1 ] [ 2 4 1 3 5 ] [ 2 4 1 5 3 ] [ 2 4 5 1 3 ] [ 2 4 5 3 1 ]  [ 2 5 3 4 1 ] [ 2 5 3 1 4 ] [ 2 5 4 3 1 ] [ 2 5 4 1 3 ] [ 2 5 1 4 3 ] [ 2 5 1 3 4 ]  [ 3 2 1 4 5 ] [ 3 2 1 5 4 ] [ 3 2 4 1 5 ] [ 3 2 4 5 1 ] [ 3 2 5 4 1 ] [ 3 2 5 1 4 ]  [ 3 1 2 4 5 ] [ 3 1 2 5 4 ] [ 3 1 4 2 5 ] [ 3 1 4 5 2 ] [ 3 1 5 4 2 ] [ 3 1 5 2 4 ]  [ 3 4 1 2 5 ] [ 3 4 1 5 2 ] [ 3 4 2 1 5 ] [ 3 4 2 5 1 ] [ 3 4 5 2 1 ] [ 3 4 5 1 2 ]  [ 3 5 1 4 2 ] [ 3 5 1 2 4 ] [ 3 5 4 1 2 ] [ 3 5 4 2 1 ] [ 3 5 2 4 1 ] [ 3 5 2 1 4 ]  [ 4 2 3 1 5 ] [ 4 2 3 5 1 ] [ 4 2 1 3 5 ] [ 4 2 1 5 3 ] [ 4 2 5 1 3 ] [ 4 2 5 3 1 ]  [ 4 3 2 1 5 ] [ 4 3 2 5 1 ] [ 4 3 1 2 5 ] [ 4 3 1 5 2 ] [ 4 3 5 1 2 ] [ 4 3 5 2 1 ]  [ 4 1 3 2 5 ] [ 4 1 3 5 2 ] [ 4 1 2 3 5 ] [ 4 1 2 5 3 ] [ 4 1 5 2 3 ] [ 4 1 5 3 2 ]  [ 4 5 3 1 2 ] [ 4 5 3 2 1 ] [ 4 5 1 3 2 ] [ 4 5 1 2 3 ] [ 4 5 2 1 3 ] [ 4 5 2 3 1 ]  [ 5 2 3 4 1 ] [ 5 2 3 1 4 ] [ 5 2 4 3 1 ] [ 5 2 4 1 3 ] [ 5 2 1 4 3 ] [ 5 2 1 3 4 ]  [ 5 3 2 4 1 ] [ 5 3 2 1 4 ] [ 5 3 4 2 1 ] [ 5 3 4 1 2 ] [ 5 3 1 4 2 ] [ 5 3 1 2 4 ]  [ 5 4 3 2 1 ] [ 5 4 3 1 2 ] [ 5 4 2 3 1 ] [ 5 4 2 1 3 ] [ 5 4 1 2 3 ] [ 5 4 1 3 2 ]  [ 5 1 3 4 2 ] [ 5 1 3 2 4 ] [ 5 1 4 3 2 ] [ 5 1 4 2 3 ] [ 5 1 2 4 3 ] [ 5 1 2 3 4 ]  public class FullPermutation {  static int count = 0;//计算有多少种排列  public static void perm(int[] arr, int start , int end) {  if(start == end ) {//如果没有交换的数就输出一种排列，并且计数器+1  System.out.print("[ ");  for(int i = 0 ;i < arr.length ;i++)  System.out.print(arr[i] + " ");  System.out.print("]");  System.out.println();  count++;  return;  }else {  for(int i = start ; i <= end ; i++) {  swap(arr , i , start);  perm(arr , start + 1 , end);  swap(arr , i , start);  }  }  }  //交换素组元素  public static void swap(int[] arr , int start ,int end ) {  int temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;   }  public static void main(String[] args) {  // TODO Auto-generated method stub  int [] arr = new int[] {1, 2, 3, 4, 5};  perm(arr , 0 , arr.length-1);  System.out.println("共有：" + count + " 种排列方式");  } }   * **第2题 运行截图及代码如下：**     package com.dichotomy;  public class Dischotomy {  public static void main(String[] args) { //【测试程序】  int[] arr = {1,8,12,15,16,21,30,35,39};  int value = 30; //所要查找的值  System.out.print("查找元素30的下标为：");  printout(arr , 30);  value = 20; //所要查找的值  System.out.print("查找元素20的下标为：");  printout(arr , 20);  }  //对输出进行筛选后再进行输出，目的在于屏蔽错误返回值-1  static void printout(int[] arr ,int value){  //获得返回的下标值  int re = serach(arr,value,0,arr.length-1) ;  //如果下标值为错误值-1，则不输出该值  if (re != -1){  System.out.println(serach(arr,value,0,arr.length-1));  }else { }  }  //递归调用，二分法查找  static int serach(int[] arr, int value ,int left, int right){  //二分法，取列表中间为基准进行比较  int mid = (left + right )/ 2;  //判断是否已经变成 单元素  if (left == right){  //若干该元素正好为查找值，则返回下标  if (arr[mid] == value){  return mid ;  //否则就是列表中无该元素。  }else {  System.out.println("无此元素。");  //待搜索元素x=10不在数组中时，  //返回小于x的最大元素位置i和大于x的最小元素位置j。  if (arr[mid] < value){  System.out.println("小于"+value+"的最大元素位置i为：" + mid);  System.out.println("大于"+value+"的最小元素位置j为：" + (mid +1));  }else {  //相反过来，则左边元素为小于value的最大值  System.out.println("小于"+value+"的最大元素位置i为：" + (mid-1));  System.out.println("大于"+value+"的最小元素位置j为：" + mid);  }  //返回错误值-1，否则会执行下面的代码，导致栈溢出  return -1;  }  }  //进行拆分  //首先判断是否中值就是所找值  if (arr[mid] == value){  return mid;  }else{  //比较查找值与中值的关系  if (arr[mid] < value){  //扫描右半部分  left = mid + 1;  //递归返回的值赋给最外层的mid  mid = serach(arr,value,left,right);  }else { //扫描左半部分  right = mid - 1;  //递归返回的值赋给最外层的mid  mid = serach(arr,value,left,right);  }  }  //返回最终的中值 （单元素值）  return mid;  }  }   * **第4题 运行截图及代码如下：**     //【测试程序】  package com.recursion;  import java.util.Arrays;  public class main {  public static void main(String[] args) {  int[] arr = {8,4,3,7,1,5,6,2};  System.out.println("输入的数组为："+ Arrays.toString(arr));  QuickSort qs = new QuickSort();  qs.quickSort(arr,0,7);  System.out.println("排序后数组为："+Arrays.toString(arr));  }  }  //QuickSort.java代码如下：  package com.recursion;  public class QuickSort{  void quickSort(int[] a ,int p , int right){  if (p < right){  int q = devide(a,p,right);  quickSort(a, p,q);  quickSort(a, q+1,right);  }  }  int devide(int[] a ,int point ,int right){  int i = point;  int j = right;  int x = a[point];  while (true){  while(a[i] - x <= 0 && i < right){ i++;}  while (a[j] - x > 0){ j--; }  if (i >= j) break;  int temp;  temp = a[i];  a[i] = a[j];  a[j] = temp;  }  x = a[j];  a[j] = a[point];  a[point] = x;  return j;  }  }   * **第5题 运行截图及代码如下：**   答：快速排序最坏情况是输入一个已经有序列的递增数组，时间复杂度O(n2)，而随机化算法因为产生随机数正好为0的情况，比例是1/2n ，这个比例随着问题规模n的增加，已经接近无限小，所以不需要分析最坏情况。    package com.recursion;  import java.util.Arrays;  public class QuickSort{  int randomdevide(int[] a ,int p , int r){  int i =(int) (p + (Math.random() \* (r-p)));  int temp = a[p];  a[p] = a[i];  a[i] = temp;  return devide(a,p,r);  }  void quickSort(int[] a ,int p , int right){  // p = (int) (Math.random() \* right);  if (p < right){  int q = randomdevide(a,p,right);  quickSort(a, p,q-1);  quickSort(a, q+1,right);  }  }  int devide(int[] a ,int point ,int right){  int i = 0;  int j = right;  int x = a[point];  while (true){  while(a[i] - x <= 0 && i < right){  i++;  }  while (a[j] - x > 0){  j--;  }  if (i >= j) break;  int temp;  temp = a[i];  a[i] = a[j];  a[j] = temp;  }  a[point] = a[j];  a[j] = x;  return j;  }  public static void main(String[] args) {  int[] arr = {8,4,3,7,1,5,6,2};  QuickSort qs = new QuickSort();  qs.quickSort(arr,0,arr.length-1);  System.out.println(Arrays.toString(arr));  }  } | | | | | |
| 1. 实验总结（包括心得体会、问题回答及实验改进意见，可附页）   随机化快速排序思路为：产生一个介于当前子区间的随机数，定位随机数对应下标的元素，将该元素与首元素对换后执行以首元素为基准元素的快速排序。  快速排序最坏情况是输入一个已经有序列的递增数组，时间复杂度O(n2)，而随机化算法因为产生随机数正好为0的情况，比例是1/2n ，这个比例随着问题规模n的增加，已经接近无限小，所以不需要分析最坏情况 | | | | | |
| 六、教师评语  1、完成所有规定的实验内容，实验步骤正确，结果正确；  2、完成绝大部分规定的实验内容，实验步骤正确，结果正确；  3、完成大部分规定的实验内容，实验步骤正确，结果正确；  4、基本完成规定的实验内容，实验步骤基本正确，所完成的结果基本正确；  5、未能很好地完成规定的实验内容或实验步骤不正确或结果不正确。  6、其它：  评定等级：优秀 良好 中等 及格 不及格  教师签名：郭艺辉 | | | | | |