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 | 算法分析与设计实验3 | | | 系 别 | 计算机科学与技术 |
| 姓 名 | 林旋华 | 学 号 | 181543306 | 班 级 | 1815433 |
| 实验地点 | 电教503 | 实验日期 | 2020.09.24 | 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 2. 掌握递归与分治法的基本思想及基本原理。 3. 掌握使用分治法求解问题的一般特征及步骤。 4. 掌握分治法的设计方法及复杂性分析方法。 5. 掌握基于分治策略的二分搜索算法、快速排序算法的算法设计思想、算法设计的实现及时间复杂性分析。 | | | | | |
| 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）   1、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：   1. 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。 2. 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。 3. 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*。   2、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。   1. 写出算法实现代码并截屏程序的运行结果。 2. 分析快速排序算法在最好以及最坏情况下的时间复杂性。   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。 | | | | | |
| 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）   1.二分搜索算法  1)public class binarySearch {  public static void main(String[] args) {  // TODO 自动生成的方法存根  int[] array = new int[] { 1, 8, 12, 15, 16, 21, 30, 35, 39 };  int result = binarySearch(array, 30, array.length);  if (result == -1) {  System.out.println("无此元素");  } else {  System.out.println("此元素在" + result + "位置");  }  }  public static int binarySearch(int[] a, int x, int n) {  int left = 0;  int right = n - 1;  while (left <= right) {  int middle = (left + right) / 2;  if (x == a[middle]) {  return middle;  }  if (x > a[middle]) {  left = middle + 1;  } else {  right = middle - 1;  }  }  return -1;  }  }    2)public static void main(String[] args) {  // TODO 自动生成的方法存根  int[] array = new int[] { 1, 8, 12, 15, 16, 21, 30, 35, 39 };  int result = binarySearch(array, 20, array.length);  if (result == -1) {  System.out.println("无此元素");  } else {  System.out.println("此元素在" + result + "位置");  }  }    3)public class QHSearch {  public static void main(String[] args) {  // TODO 自动生成的方法存根  int[] arr = new int[] { 1, 8, 12, 15, 16, 21, 30, 35, 39 };  int[] res = binarySearch(arr, 10, arr.length);  int l = 0;  int r = 0;  if (res[0] == -1) {  l = res[1] - 1;  r = res[1];  System.out.println("无此元素");  System.out.println("小于x的最大元素位置："+l);  System.out.println("大于x的最小元素位置："+r);  } else {  System.out.println("此元素在" + res[0] + "位置");  }  }  public static int[] binarySearch(int[] a, int x, int n) {  int left = 0;  int right = n - 1;  int[] res = new int[] { -1, 0 };  while (left <= right) {  int middle = (left + right) / 2;  if (x == a[middle]) {  res[0] = middle;  }  if (x > a[middle]) {  left = middle + 1;  } else {  right = middle - 1;  }  }  return res;  }  }    2.快速排序算法  1)public class qSort {  public static int[] QuickSort(int[] array, int start, int end) {  if ((array.length < 1) || (start < 0) || (end >= array.length) ||  (start > end)) {  return null;  }  int smallIndex = partition(array, start, end);  if (smallIndex > start) {  QuickSort(array, start, smallIndex - 1);  }  if (smallIndex < end) {  QuickSort(array, smallIndex + 1, end);  }  return array;  }  /\*\*  \* 快速排序算法——partition  \*/  public static int partition(int[] array, int start, int end) {  int pivot = start ;  int smallIndex = start - 1;  swap(array, pivot, end);  for (int i = start; i <= end; i++)  if (array[i] <= array[end]) {  smallIndex++;  if (i > smallIndex) {  swap(array, i, smallIndex);  }  }  return smallIndex;  }  /\*\*  \* 交换数组内两个元素  \*/  public static void swap(int[] array, int i, int j) {  int temp = array[i];  array[i] = array[j];  array[j] = temp;  }  public static void display(int[] array) {  for (int i = 0; i < array.length; i++) {  System.out.print(array[i] + " ");  }  System.out.println();  }  public static void main(String[] args) {  int[] li = { 8, 4, 3, 7, 1, 5, 6, 2 };  System.out.println("原数组：");  display(li);  QuickSort(li, 0, li.length - 1);  System.out.println("快排后：");  display(li);  }  }    2)   * 最坏情况：   ①两个区域分别包含n-1个元素和1个元素。  ②时间复杂度：T(n)=  a=2,b=2,k=1  T(n)=O()   * 最好情况：   ①每次划分都产生两个大小为n/2的区域。  ②时间复杂度：T(n)=  T(n)=O()   * 快速排序算法的性能取决于划分的对称性   在数组还没有被划分时，在array[start,end]中随机选出一个元素作为划分基准，可以使划分基准的选择是随机的，从而可以期望划分是较对称的。  public static int partition(int[] array, int start, int end) {  int pivot = (int) (start + (Math.random() \* (end - start + 1)));  int smallIndex = start - 1;  swap(array, pivot, end);  for (int i = start; i <= end; i++)  if (array[i] <= array[end]) {  smallIndex++;  if (i > smallIndex) {  swap(array, i, smallIndex);  }  }  return smallIndex;  } | | | | | |
| 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）   通过本次实验，我掌握了递归与分治法的基本思想及基本原理，学习了如何使用分治法来求解问题，以及使用分治策略来实现二分搜索算法、快速排序算法，同时掌握了其时间复杂性分析方法。  在算法优化的过程中，让我明白快速排序算法的性能取决于划分的对称性，我们可以在array[start,end]中随机选出一个元素作为划分基准，提高算法的效率，从而增强算法的处理问题的能力。 | | | | | |
| 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 | | | | | |