**《算法设计与分析》课程实验报告**

**青岛大学 数据科学与软件工程学院**

实验序号：二 实验项目名称：分治法实验二

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 李自家 | 同组人员 |  | | |
| 班级 | J03 | 指导教师 | 李晓娜 | 完成日期 | 11.20 |
| 一、实验题目  （1）给定n个整数序列<a1,a2,......,an>,输出该序列的一个排列 <a1’,a2’,......,an’>，满足a1’<=a2’<=......<=an’。  （2）（ 选做题，可以任选一题或两题都完成）  实现一个时间复杂度为θ（nlogn)的算法，该算法对给定n个整数的集合S和另一个整数x，确定S中是否存在两个其和刚好为x的元素。  给一列数<a1,a2,......,an>,求它的逆序对，即有多少个有序对（i.j）,使得i<j且ai<aj; n 可以高达106  二、实验目的  （1）进一步理解分治法解决问题的思想及步骤  （2）熟练掌握归并排序和快速排序算法  （3）（选做题）灵活运用分治法及排序算法解决实际问题，练习算法思想的描述，加深对分治算法解决排序问题的理解并恰当灵活运用。  三、实验要求  必做题：（1）参照教材分别给出合并排序的递归和非递归的两种实现以及快速排序的递归及随机化两种实现。  （2）根据你的数据结构设计测试数据，并对每一组数据测试两种实现，记录实验结果。  （3）比较两种实现的时间效率，并分析每种实现的优缺点。  选做题：（1）描述你的算法思想、分析其时间复杂度  （2）实现你的算法、设计测试数据并记录实验结果。     1. 实验过程（算法设计思想、源码） 2. 合并排序 3. 递归   首先递归的将将要排列的数列分割直至每一组都只剩一个数，在将它们两两排序后放回上层，直到排序完成。  代码：  #include<stdio.h>  //int b[50];  //数组复制函数  void Copy(int a[50],int b[50],int left,int right){  for(int i = 0;i<=(right-left);i++){  a[left+i] = b[left+i];  }  }  //合并排序合并两个部分  void Merge\_1(int a[50],int b[50],int left,int i,int right){  int base = left;  int mid = i;  for(int j = 0;j<=(right-left);j++){  if(a[left]<=a[i+1]){  b[base+j] = a[left];  if(left==mid&&i==(right-1)){  b[right] = a[i+1];  }  left++;  }  else if(a[left]>a[i+1]){  b[base+j] = a[i+1];  if(left==mid&&i==(right-1)){  b[right] = a[left];  }  i++;  }  }  }    //合并排序合并两个部分  void Merge\_2(int arr[],int low,int mid,int high)  {  int left\_low=low; //左边序列的左下标  int left\_high=mid;  int right\_low=mid+1; //右边序列的左下标  int right\_high=high;  int tmp[50],k=0,i; //数组tmp用来存放序列元素  while(left\_low<=left\_high&&right\_low<=right\_high) //当左右序列都有元素时执行循环  {  if(arr[left\_low]<arr[right\_low])  tmp[k++]=arr[left\_low++]; //元素小的放进数组tmp  else  tmp[k++]=arr[right\_low++];  }  if(left\_low<=left\_high){ //左边序列还有元素时  for(i=left\_low;i<=left\_high;i++)  tmp[k++]=arr[left\_low++];  }  if(right\_low<=right\_high){ //右边序列还有元素时  for(i=right\_low;i<=right\_high;i++)  tmp[k++]=arr[right\_low++];  }    for(i=0;i<high-low+1;i++){ //将数组tmp中的元素赋给数组arr，从而数组arr有序  arr[low+i]=tmp[i];  }  }  //合并排序，递归方案  void MergeSort\_1(int a[50],int left,int right){  if(left<right){  int i = (left+right)/2;  MergeSort\_1(a,left,i);  MergeSort\_1(a,i+1,right);  Merge\_2(a,left,i,right);  // Copy(a,b,left,right);  }      }  int main(){    //储存将要输入的数组的个数  int n = 0;  int a[50];    // for(int i = 0;i<1;i++){  // printf("%d ",i);  // }    printf("请输入待排序数组的数字个数（少于50个数字）：");  scanf("%d",&n);  for(int i = 0;i<n;i++){  printf("请输入待排序数组的第%d数字（少于50个数字）：",i+1);  scanf("%d",&a[i]);  }  MergeSort\_1(a,0,n-1);  printf("排序完成后的数组：");  for(int i = 0;i<n;i++){  printf("%d ",a[i]);  }  // printf("\n");  // for(int i = 0;i<n;i++){  // printf("%d ",b[i]);  // }      return 0;  }   1. 非递归   不使用递归的方式分割数组，而是先两个两个的排序，然后将排好序的两两序列排序，直至完成排序。  代码：  #include<stdio.h>  #include<string.h>  #include<stdlib.h>  void mergeAdd1(int arr[], int left, int mid, int right, int \*temp){  int i = left;  int j = mid + 1;  int k = left;//临时下标  while (i <= mid&&j <= right){  if (arr[i] < arr[j]){  temp[k++] = arr[i++];  }  else{  temp[k++] = arr[j++];  }  }  while (i <= mid){  temp[k++] = arr[i++];  }  while (j <= right){  temp[k++] = arr[j++];  }  //把temp中的内容拷给arr数组中  //进行归并的时候，处理的区间是arr[left,right),对应的会把  //这部分区间的数组填到tmp[left,right)区间上  memcpy(arr + left, temp + left, sizeof(int)\*(right - left + 1));  }  void mergeSort2(int arr[],int len,int\* tmp){  if (len <= 1){  return;  }  //定义一个步长gap，初始值为1，相当于每次只合并两个长度为1的元素  int gap = 1;  for (; gap <= len; gap \*= 2){  int i = 0;  for (; i <= len; i += 2 \* gap){  int beg = i;  int mid = (gap - 1) + i;  if (mid >= len){  mid = len;  }  int end = mid + gap;  if (end >= len){  end = len;  }  mergeAdd1(arr, beg, mid, end, tmp);  }  }  }  int main(){    int arr[50];  int n = 0;  printf("请输入待排序数组的数字个数（少于50个数字）：");  scanf("%d",&n);  for (int i = 0; i < n; i++){  printf("请输入待排序数组的第%d数字（少于50个数字）：",i+1);  scanf("%d",&arr[i]);  }    int \*temp = (int\*)malloc(sizeof(int)\*n);    mergeSort2(arr, (n - 1), temp);  free(temp);  for (int i = 0; i < n; i++){  printf("%d ", arr[i]);  }  return 0;  }   1. 快速排序 2. 递归   设定输入数组的最左侧的数字为基准数字，首先从最右侧的指针开始寻找小于基准数字的数，找到后在从左游标开始寻找大于基准数字的数，都找到后相互调换位置，如果两游标相遇，则调换基准数字和此时两游标所指数字。然后向基准数字两侧递归调用方法。  代码：  #include <stdio.h>  int a[101],n; //定义两个全局变量    void QuickSort(int left,int right){  int i,j,temp,t;  if(left>right)  return;  temp=a[left];  i=left;  j=right;  while(i!=j)  {  while(a[j]>=temp&&i<j)  {  j--;  }  while(a[i]<=temp&&i<j){  i++;  }  if(i<j) //交换两个数在数组中的位置  {  t=a[i];  a[i]=a[j];  a[j]=t;  }  }  //最终将基数定位  a[left]=a[i];  a[i]=temp;  QuickSort(left,i-1); //递归处理左侧  QuickSort(i+1,right); //递归处理右侧  }    int main(){  int i,j,t;  printf("请输入待排序数组的数字个数（少于50个数字）：");  scanf("%d",&n);  for(i=0;i<n;i++)  {  printf("请输入待排序数组的第%d数字（少于50个数字）：",i+1);  scanf("%d",&a[i]);  }    QuickSort(0,n-1); //快速排序调用    for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  return 0;  }   1. 随机   对于基准数字的选择变成随机化，将随机选择的基准数字与最左侧数字兑换，之后步骤与递归方式相同。  代码：  #include<iostream>  #include<cstdio>  #include<stdio.h>  #include<stdlib.h>  using namespace std;  int Partition(int \*array,int low,int high){    int k=low+rand()%(high-low+1);  int tmp;  tmp=array[k];array[k]=array[low];array[low]=tmp;//随机选择一个点，然后将它与第low个点互换。  int key=array[low];  while(low<high){  while(low<high&&array[high]>key) high--;  array[low]=array[high];  while(low<high&&array[low]<key) low++;  array[high]=array[low];  }  array[low]=key;  return low;  }  void qsort(int \*array,int low,int high){  if(low<high){  int po=Partition(array,low,high);  qsort(array,low,po-1);  qsort(array,po+1,high);  }  }  int main(){  int q[50];    int n;  printf("请输入待排序数组的数字个数（少于50个数字）：");  scanf("%d",&n);  for(int i=0;i<n;i++)  {  printf("请输入待排序数组的第%d数字（少于50个数字）：",i+1);  scanf("%d",&q[i]);  }    qsort(q,0,n-1);  for(int i=0;i<n;i++)  printf("%d ",q[i]);  }   1. 测试数据、结果、分析、讨论： 2. 合并排序 3. 递归   输入奇数个数数组：3/4/2/5/6  1  可见已成功排序；  输入偶数个数数组：3/2/4/5  2  可见已成功排序；   1. 非递归   输入奇数个数数组：3/4/2/5/6  3  可见已成功排序；  输入偶数个数数组：3/4/2/54/5  可见已成功排序；  递归方式时间复杂度为O（nlogn）,而非递归方式时间复杂度为O(n),  所以在所需排序数字个数大的情况下，非递归方式更加节省时间。   1. 快速排序 2. 递归   输入奇数个数数组：3/4/2/5/6  5  可见已成功排序；  输入偶数个数数组：3/4/2/5  6  可见已成功排序；   1. 随机   输入奇数个数数组：3/4/2/5/6  7  可见已成功排序；  输入偶数个数数组：3/4/2/5  8    可见已成功排序；  因为快速排序的平均时间复杂度为：O(nlogn)，所以快速排序在整体上来说比合并排序更加快速。 | | | | | |
| 教师评语及成绩：  签名： 日期： | | | | | |