班级 计算机203 学号 202007020625 姓名 於俊涛 报告日期 2020年12月1日

**实验题目：查找及排序数据结构的关键算法实现及性能分析**

**一、实验目的**

1.能够阐明基于线性结构、树结构乃至散列结构的不同种类的查找的工作原理；  
2.能够阐明不同类型排序算法的工作原理；  
3.能够实现典型的查找和排序算法；  
4.能够运用集成开发环境编写和调试C语言程序；  
5.能够生成不同规模的数据集，评估不同类型查找（或排序）算法之间的性能差异。

二、实验原理

1.查找及排序数据结构的基本原理

（1）二分查找

首先，假设表中元素是按升序排列，将表中间位置记录的关键字与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的记录，使查找成功，或直到子表不存在为止，此时查找不成功。

1. 查找二叉排序树

一棵空树，或者是具有下列性质的二叉树：  
若左子树不空，则左子树上所有结点的值均小于它的根结点的值；若右子树不空，则右子树上所有结点的值均大于它的根结点的值；左、右子树也分别为二叉排序树；没有键值相等的结点。

1. 希尔排序

希尔排序法(缩小增量法)属于插入类排序，是将整个无序列分割成若干小的子序列分别进行插入排序的方法。

1. 快速排序

设要排序的数组是A[0]……A[N-1]，首先任意选取一个数据（通常选用数组的第一个数）作为关键数据，然后将所有比它小的数都放到它左边，所有比它大的数都放到它右边，这个过程称为一趟快速排序。值得注意的是，快速排序不是一种稳定的排序算法，也就是说，多个相同的值的相对位置也许会在算法结束时产生变动。

1. 堆排序

堆排序（英语：Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

1. 关键算法设计原理及性能影响因素分析
2. 二分查找

一个运用二分查找算法的程序的时间复杂度是“对数级别”。二分查找是一种效率较高的查找方法，算法复杂度即是while循环的次数，时间复杂度可以表示“O(h)=O(log2n)”

1. 希尔排序

时间复杂度：介于O(n^2)和O(nlogn)之间空间复杂度：O(1)不稳定

1. 快速排序

时间复杂度平均O(nlog2n)最好O(nlog2n)最坏O(n^2)空间复杂度O(1)不稳定

1. 堆排序

**三、实验方案设计**

1.存储方案设计

二分查找：采用有序顺序存储结构实现，操作依赖下标，所以不适合用链表

查找二叉排序树：采用树型结构，利用链式存储结构，方便树的相关寻址操作

顺序存储可能会浪费空间(在非完全二叉树的时候)，但是读取某个指定的节点的时候效率比较高

链式存储相对二叉树比较大的时候浪费空间较少，但是读取某个指定节点的时候效率偏低

线性探测法查找哈希表： 采用顺序存储结构，地址增量+1，用下标更方便描述

快速排序，冒泡排序，插入排序，希尔排序均使用顺序表存储

便于用下标访问数据元素进行比较和排序。

2.算法的设计和实现

二分查找：

Position BinarySearch( List L, ElementType X ){

int low,mid,high;

low=1;high=L->Last;

while(low<=high){

mid=(low+high)/2;

if(L->Data[mid]==X){

return mid;

}else if(L->Data[mid]<X){

low=mid+1;

//return NotFound;

}else {

high=mid-1;

}

}

return NotFound;

}

查找二叉排序树：

int SearchBST(BSTNode \*bt,KeyType k){

if(!bt)

return 0;

printf("%d ",bt->key);

if(k == bt->key)

return 1;

else if(k < bt->key)

return SearchBST(bt->lchild,k);//在左子树中继续查找

else

return SearchBST(bt->rchild,k);//在右子树中继续查找

}

快速排序：

int Partition (SqList L,int low,int high){

L.elem[0]=L.elem[low];

int pivotkey=L.elem[0];

while(low<high){

while(low<high&&L.elem[high]>=pivotkey)high--;

L.elem[low]=L.elem[high];

while(low<high&&L.elem[low]<=pivotkey)low++;

L.elem[high]=L.elem[low];

}

L.elem[low]=L.elem[0];

return low;

}

希尔排序：

void ShellInsert(SqList L,int dk){

int i;

for (i=1+dk;i<=L.Length;i++){

if(L.elem[i]<L.elem[i-dk]){

L.elem[0]=L.elem[i];

L.elem[i]=L.elem[i-dk];

int j;

for (j=i-dk;j>0&&L.elem[0]<L.elem[j];j-=dk){

L.elem[j+dk]=L.elem[j];

}

L.elem[j+dk]=L.elem[0];

}

}

}

插入排序：

void insertionSort(int arr[], int n)

{

int i,j,t,p,k;

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(arr[j]<arr[i])

{

p=arr[j];

for(k=j;k>i;k--)

{

arr[k]=arr[k-1];

}

arr[k]=p;

}

}

}

}

3.测试方案

二分查找：

输入样例1：

5

12 31 55 89 101

31

输出样例1：

2

输入样例2：

3

26 78 233

31

输出样例2：

0

二叉查找树：

输入样例1：

9

4 9 0 1 8 6 3 5 7

6

输出样例1：

4 9 8 6 :Found

输入样例2：

9

4 9 0 1 8 6 3 5 7

10

输出样例2：

4 9 :NOT Found

合并两个有序数组

输入样例：

7 11

1 2 14 25 33 73 84

5 6 17 27 68 68 74 79 80 85 87

输出样例：

1 2 5 6 14 17 25 27 33 68 68 73 74 79 80 84 85 87

希尔排序

输入样例：

10

5 2 4 1 8 9 10 12 3 6

输出样例：

1 2 3 4 5 6 8 9 10 12

快速排序

输入样例：

10

5 2 4 1 8 9 10 12 3 6

输出样例：

1 2 3 4 5 6 8 9 10 12