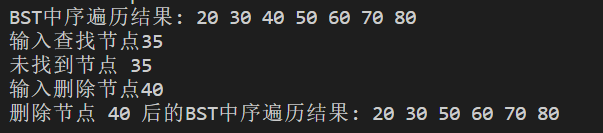
BST查找结构与折半查找方法的实现与实验比较：

1. BST 的左右链存储结构，并实现BST插入（建立）、删除、查找和排序算法：(5.3.c)



1. 折半查找算法（5.2.c）

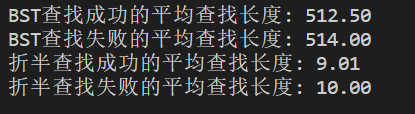


1. 实验比较(5.c)

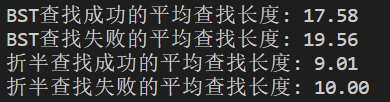
本程序利用源文件5.1.c生成n=1024个已排序的整数序列（0至2048之间的奇数），存储至"odd\_numbers.txt"，和随机打乱顺序的序列，存储至"shuffled\_numbers.txt"。

然后利用5.c中的程序进行实验比较：

第一组数据：



第二组数据：



由于折半查找的序列是由BST中序遍历而来，所以两组数据折半查找的结果相同。

实验结论与理论结果基本符合。

2.

1. 堆是一种完全二叉树的结构，堆对父节点和左右子节点的大小关系有要求，而对左，右子节点间的大小关系没有要求。

二叉排序树要求左子树上所有节点的值均小于根节点的值而且右子树上所有节点的值均大于根节点的值；



堆排序。该算法无需将全部元素排序，即可得到最小的k个元素。

具体实现步骤如下：

构建一个大小为k的最小堆，将序列的前k个元素插入堆中。

遍历序列的剩余元素，如果当前元素小于堆顶元素，则替换为，并调整堆。

遍历完成后，堆中的元素即为序列中的k个最小元素。

1. 基本思想：利用快速排序的第一层（不进行递归），将较小的n/2个元素放在A1中，将较大的n/2个元素放在A2中，即可满足题意。

int FindMax(int\* data,int size)

{

int low = 0,high = size-1;

int low0 = 0,high0 = size-1;

int k = size/2;

bool flag = true;

while(flag){

int pivot = data[low]; //确定基准元素

while(low < high){

//找到右边第一个比枢纽小的数

while(low < high && data[high] > pivot){

high--;

}

if(low != high){//如果还划分没有结束

data[low] = data[high];

}

//找到左边第一个比枢纽大的数

while(low < high && data[low] < pivot){

low++;

}

if(low != high){//如果还划分没有结束

data[high] = data[low];

}

data[low] = pivot;//把枢纽放到合适的位置

if(low == k-1){//如果low刚好是中间位置

flag = false;

}else if(low < k-1){//现在枢纽位置小于中间位置

low0 = low++;//说明前low个数是已经是最小的了

high = high0;

}else{

high0 = high--;//说明后high个数是已经是最大的了

low = low0;

}

}

}

int s1 = 0,s2 = 0;

for(int i = 0 ; i < k ; i++){

s1 += data[i];

}

for(int i = k ; i < size ; i++){

s2 += data[i];

}

return s2-s1;

}

该算法无需将全部元素排序，其平均时间复杂度为O(n),空间复杂度为O(1)