（1）在二叉排序树中，每个结点的值均大于其左子树上所有结点的值，小于其右子树上所有结点的值，对二叉排序树进行中序遍历得到一个有序序列。所以，二叉排序树是结点之间满足一定次序关系的二叉树；

　　堆是一个完全二叉树，并且每个结点的值都大于或等于其左右孩子结点的值（这里的讨论以大根堆为例），所以，堆是结点之间满足一定次序关系的完全二叉树。

　　具有n个结点的二叉排序树，其深度取决于给定集合的初始排列顺序，最好情况下其深度为log n（表示以2为底的对数），最坏情况下其深度为n；

　　具有n个结点的堆，其深度即为堆所对应的完全二叉树的深度log n 。

在二叉排序树中，某结点的右孩子结点的值一定大于该结点的左孩子结点的值；在堆中却不一定，堆只是限定了某结点的值大于（或小于）其左右孩子结点的值，但没有限定左右孩子结点之间的大小关系。

在二叉排序树中，最小值结点是最左下结点，其左指针为空；最大值结点是最右下结点，其右指针为空。在大根堆中，最小值结点位于某个叶子结点，而最大值结点是大根堆的堆顶（即根结点）。

二叉排序树是为了实现动态查找而设计的数据结构，它是面向查找操作的，在二叉排序树中查找一个结点的平均时间复杂度是O(log n)；

堆是为了实现排序而设计的一种数据结构，它不是面向查找操作的，因而在堆中查找一个结点需要进行遍历，其平均时间复杂度是O(n)。

（2）堆排序

（3）先将A从小到大排序，易知要使n1-n2的绝对值最小，且s1-s2的绝对值最大，假设n为偶数，则n1=n2=n/2，假设n为奇数，则n1=(n-1)/2，n2=(n+1)/2。又发现其实不需要将A从小到大排序，只需要找到一个数，使得它前面的都比他小，数量为n/2，它后面的都比他大即可。于是可以借助快速排序的思想：

#include <iostream>

using namespace std;

int partition(int a[], int n)

{

int i = 0, j = n - 1; // 指向下界和上界

int i0 = 0, j0 = n - 1; // 指向新表的下界和上界

int s1 = 0, s2 = 0; // A1和A2的和

int f = 1; // 标记是否划分成功

int mid = n / 2;

while (f)

{

int pivot = a[i];

while (i < j)

{

while (i < j && a[j] >= pivot)

{

j--;

}

if (i != j)

{

a[i] = a[j];

}

while (i < j && a[i] <= pivot)

{

i++;

}

if (i != j)

{

a[j] = a[i];

}

}

a[i] = pivot;

// 如果是中间位置

if (i == mid - 1)

{

f = 0;

}

else

{

// 如果在中间前面的位置，那么0~i的元素属于A1，从i+1后面继续查找

if (i < mid - 1)

{

i0 = ++i;

j = j0;

}

else

{

// 如果在中间后面的位置，那么在j~n-1的元素属于A2,从j-1之前查找

j0 = --j;

i = i0;

}

}

}

// 计算A1

for (int i = 0; i < mid; ++i)

{

s1 += a[i];

}

// 计算A2

for (int i = mid; i < n; ++i)

{

s2 += a[i];

}

return s2 - s1;

}

int main()

{

int a[100];

int n;

cin >> n;

for (int i = 0; i < n; ++i)

{

cin >> a[i];

}

cout << partition(a, n) << endl;

}

时间复杂度O(n)，空间复杂度O(1)。