数据结构 A
第九章作业参考答案

题 目 范 围 ：查找
授 课 老 师 ：何璐璐
助 教 ：王思怡（撰写人），赵守玺
联 系 邮 箱 ：2021302111095@whu.edu.cn

目录

1. 单选题

题目1

下列关键字序列不可能是二叉排序树的查找路径的为 _____

A.95，22，91，24，94，71

B.92，20，91，34，88，35

C.21，89，77，29，36，38

D.12，25，71，68，33，34

答案：A

解析：对于二叉排列树的任一子树, 它的左子树上的节点都大于根，右子树上的节点都小于根。A: 通过“95，22，91，24”可以判断出关键字大于24且小于91，所以 95,22,91,24,94 不可能构成某二叉排列树中一条查找路径的序列.

题目2

下列选项中，错误的是（ _____ ）。

A.红黑树中，任何一个结点的左右子树高度只差不超过2倍

B.AVL树中，任何一个结点的左右子树高度之差不超过1

C.红黑树的查找效率要优于AVL树

D.红黑树和AVL树的查找、插入和删除操作的最坏时间复杂度相同

答案：C

解析：广度优先搜索时会依次访问同一表结点的所有节点，所以类似于依次访问同一层的所有节点，也就是类似于层次遍历。

题目3

已知一3阶B-树中有2047个关键字，则树的最大高度是 _____

A.11

B.12

C.13

D.14

答案：B

解析：

3阶B-树也就是我们平常讲的B树，该树有一个特点就是：叶子节点可以看成外部节点，不包含任何信息（即不包含关键字）；而题目说的是包含叶节点共有2047个关键字；所以我们算出11层之后还应该加上1层------不包含任何信息的叶子节点层；

题目4

从19个元素中查找其中某个元素，如果最多进行5次元素之间的比较，则采用的查找方法只可能是 _____ 。

A.折半查找

B.分块查找

C.顺序查找

D.都不可能

答案：A。

解析：n=19，折半查找的元素最多比较次数=$\lceil \log_2(n+1) \rceil$=5，顺序查找和分块查找所需元素比较次数会更多。

题目5

采用分块查找，如果线性表一共有625个元素，查找每个元素的概率相同，假设采用顺序查找来确定结点所在的块，每块分为 _____ 个结点最佳

A.9

B.25

C.6

D.625

答案: B

解析：

采用分块查找的时候，我们的目的是最小化整体查找的平均时间。在这种情况下，我们假设线性表被分成若干个块，并且采用顺序查找来确定元素所在的块，然后在该块内使用顺序查找来找到具体的元素。

如果线性表共有 n 个元素，分成 b 个块，每个块大概有 k 个元素，其中 $k = \frac{n}{b}$。平均查找时间 T 可以表示为确定块的时间加上在块内查找的时间：

$$T = \frac{b}{2} + \frac{k}{2}$$

我们希望最小化这个时间。将 k 代入 $(\frac{n}{b})$ 得：

$$T = \frac{b}{2} + \frac{\frac{n}{b}}{2}$$

为了找到最优的 b（即每个块的大小 k），我们对 T 进行求导，并令导数等于 0 找到最小值。

$$\frac{d}{db}\left(\frac{b}{2} + \frac{\frac{n}{b}}{2}\right) = 0$$

计算后得：

$$\frac{1}{2} - \frac{n}{2b^2} = 0$$

$$b^2 = n$$

$$b = \sqrt{n}$$

在本题中 $n = 625$，所以 $b = \sqrt{625} = 25$。这表示最佳的情况是每块包含 25 个元素。因此，选择B是正确的，即每块分为 25 个节点是最优的。

题目6

数据元素有序元素个数较多，且元素固定不变的情况下，应该采用 _____ 法

A.折半查找

B.分块查找

C.二叉排序树查找

D.顺序查找

答案：A

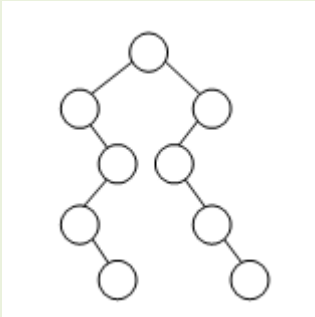
解析：

在处理有序且固定不变的大量数据元素时，折半查找（也称为二分查找）是非常高效的查找方法，其时间复杂度为$O(logn)$。这是因为折半查找可以在每一步将查找范围缩小到一半，从而大幅度减少比较的次数。

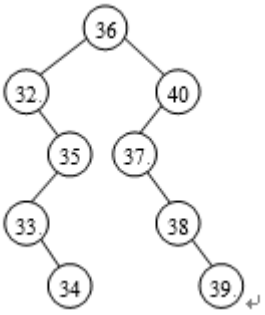
2. 简答题

题目：二叉排序树

一棵二叉排序树的结构如图1所示，其中各结点的关键字依次为32～40，请标出各结点的关键字。



答案：



3. 编程题

3.1. 求中位数

【问题描述】

给定n个整数x1，x2，…，xn，计算每对整数的差值|xi-xj|（1≤i＜j≤n），可以得到C(n，2)个差值，现在你的任务是尽快找到这些差值的中位数。注意在此问题中，如果差值的个数m为偶数，则中位数定义为第m/2个最小数，例如m=6时需要求第3个最小数。

【输入形式】

在每个测试用例中，第一行给出n，然后给出n个整数，分别代表x1，x2，…，xn（xi≤1，000，000，000，3≤n≤1，00，000）。

【输出形式】

在单独的行中输出中位数

【样例输入】

4

1 3 2 4

【样例输出】

1

【样例说明】

测试数据的文件名为in.txt

【评分标准】

该题目有10个测试用例，每通过一个测试用例，得10分。

参考答案：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
```

```
5. const int MAXN=1000005;
6. int x[MAXN];
7. int n,m;
8. bool Judge(int mid)      //如果 bigcnt<=m/2 就满足条件，返回 true
9. { int bigcnt=0;      //x+n 作为尾端指针减去 mid 所在的位置，得到的是大于
   mid 的个数
10.   for(int i=0;i<n;i++)
11.     bigcnt+= x+n-upper_bound(x+i+1,x+n,x[i]+mid);
12.   return bigcnt<=m/2;
13.}
14.int BinSearch()       //二分查找
15.{ int low=0,high=x[n-1]-x[0];
16.   while(high-low>1)     //直到 low 和 high 相差 1 时，停止二分
17.  { int mid=(low+high)/2;
18.       if (Judge(mid))
19.          high=mid;
20.       else
21.        low=mid;
22.  }
23.   return high;
24.}
25.int main()
26.{
27.     freopen("in.txt","r",stdin);
28.     scanf("%d",&n);
29.  for(int i=0;i<n;i++)
30.        scanf("%d",&x[i]);
31.  m=n*(n-1)/2;
32.  sort(x,x+n);
33.  int ans=BinSearch();
34.     printf("%d\n",ans);
35.   return 0;
36.}
```

解析：为了确定中位数，我们可以使用二分判断的方式。假设我们检查的值为mid，就可以去判断差值超出mid的数对有多少个。因此先对输入数据进行排序，这样每个数与其他数的差值大小就更方便计算。使用upper_bound可以快速找出与当前数差值超出mid的数边界位置。最后如果超出mid的数对大于m/2，就将mid定为下界，否则定为上界。

3.2. 前m大的数

【问题描述】

给定一个包含N（N≤3000）个正整数的序列，每个数不超过5000，对它们两两相加得到N×(N-1)/2个和，求出其中前M大的数（M≤1000）并按从大到小的顺序排列。

【输入形式】

输入可能包含多组数据，其中每组数据包括两行，第一行两个数N和M，第二行N个数，表示该序列。

【输出形式】

对于输入的每组数据，输出M个数，表示结果。输出应当按照从大到小的顺序排序。

【样例输入】

4 4

1 2 3 4

【样例输出】

7 6 5 5

【样例说明】

测试数据的文件名为in.txt

【评分标准】

该题目有5个测试用例，每通过一个测试用例，得20分。

参考答案：

```
#include<iostream>
#include<unordered_map>
using namespace std;
#define MAXN 3005       //数据数量最大值
#define MAXV 10001      //数据差值可能的最大值
int a[MAXN];
int main()
{
 int n,m;
freopen("in.txt","r",stdin);
```

```
while(scanf("%d%d",&n,&m)!=EOF)
{
 unordered_map<int,int>hmap;
 for(int i=0;i<n;i++)
    cin>>a[i];
 for(int i=0;i<n;i++)
    for(int j=i+1;j<n;j++)
      hmap[a[i]+a[j]]++;
 bool first=true;
 int i=MAXV;
 while(i>0&&m>0)
 {
  if(hmap[i]!=0)
  { if(first)
     { cout<<i;
       first=false;
     }
     else cout<<' '<<i;
     hmap[i]--;
     m--;
  }
  else i--;
 }
 cout<<endl;
}
}

```

解析：因为序列长度只有3000，所以可以用n^2次将所有数对的和进行相加。又因为数的大小是不超过5000的，相加结果不会超过10000。那么就从10000从大到小找计算结果是否出现，出现过多少次就输出多少次，直到输出够M次。