

信息学院本科生 2010—2011 学年第二学期 数据结构期末考试试卷（A 卷）答案

专业：_____ 年级：_____ 学号：_____

姓名：_____ 成绩：_____

得分

一、单项选择题（每小题 2 分，共 30 分）

1. 设 n 是描述问题规模的非负整数，下面程序片段的时间复杂度是_____。

```
x = 2;
while ( x < n/2 )
    x = 2*x;
```

A. $O(\log_2 n)$ B. $O(n)$ C. $O(n \log_2 n)$ D. $O(n^2)$

2. 元素 a, b, c, d, e 依次进入初始为空的栈中，所有元素进栈且只进入一次，允许进栈、退栈操作交替进行。栈空时，在所有可能的出栈序列中，以元素 d 开头的序列个数是_____。

A. 3 B. 4 C. 5 D. 6

3. 若元素 a, b, c, d, e, f 依次进栈，允许进栈、退栈操作交替进行。但不允许连续三次进行退栈工作，则不可能得到的出栈序列是_____。

A. dcebf B. cbdaef C. abcdef D. afedcb

4. 已知循环队列存储在一维数组 $A[0..n-1]$ 中，且队列非空时 front 和 rear 分别指向队头元素和队尾元素。若初始时队列为空，且要求第 1 个进入队列的元素存储在 $A[0]$ 处，则初始时 front 和 rear 的值分别是_____。

A. 0, 0 B. 0, n-1 C. n-1, 0 D. n-1, n-1

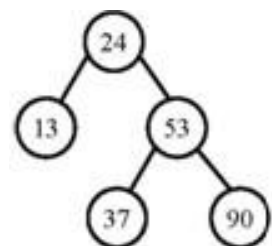
5. 若一棵完全二叉树有 768 个结点，则该二叉树中叶结点的个数是_____。

A. 257 B. 258 C. 384 D. 385

6. 在右图所示的平衡二叉树（AVL 树）中插入关键字 48 后得到一棵新平衡二叉树，在新平衡二叉树中，关键字 37 所在结点的左、右子结点中保存的关键字分别是_____。

A. 13, 48 B. 24, 48

C. 24, 53 D. 24, 90



7. 若一棵二叉树的前序遍历序列和后序遍历序列分别为 1, 2, 3, 4 和 4, 3, 2, 1, 则该二叉树的中序遍历序列不会是_____。
- A. 1, 2, 3, 4 B. 2, 3, 4, 1 **C. 3, 2, 4, 1** D. 4, 3, 2, 1
8. 对于下列关键字序列, 不可能构成某二叉排序树中一条查找路径的序列是_____。
- 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
9. 下列关于图的叙述中, 正确的是_____。
- I. 回路是简单路径
II. 存储稀疏图, 用邻接矩阵比邻接表更省空间
III. 若有向图中存在拓扑序列, 则该图不存在回路
- A. 仅 II B. 仅 I、II **C. 仅 III** D. 仅 I、III
10. 无向图 $G = (V, E)$ 中含 7 个顶点, 顶点间的边是随机设置的, 为保证图 G 在任何情况下都是连通的, 则需要的最少边数是_____。
- A. 6 B. 15 **C. 16** D. 21
11. 为提高散列 (Hash) 表的查找效率, 可以采取的正确措施是_____。
- I. 增大装填 (载) 因子
II. 设计冲突 (碰撞) 少的散列函数
III. 处理冲突 (碰撞) 时避免产生聚集 (堆积) 现象
- A. 仅 I B. 仅 II C. 仅 I、II **D. 仅 II、III**
12. 采用 Hash 技术, 下面操作中性能不佳的是_____。
- A. 搜索给定关键字。
B. 按关键字升序排列输出所有元素。
C. 删除给定关键字的元素。
D. 输出关键字升序排列位于第 k 位的元素。
13. 为实现快速排序算法, 待排序序列宜采用的存储方式是_____。
- A. 顺序存储** B. 散列存储 C. 链式存储 D. 索引存储
14. 已知序列 25, 13, 10, 12, 9 是最大堆 (大根堆), 在序列尾部插入新元素 18, 将其再调整为大根堆, 调整过程中元素之间进行的比较次数是_____。
- A. 1 **B. 2** C. 4 D. 5

15. 对一组数据 (2, 12, 16, 88, 5, 10) 进行排序, 若前三趟排序结果如下

第一趟: 2, 12, 16, 5, 10, 88

第二趟: 2, 12, 5, 10, 16, 88

第三趟: 2, 5, 10, 12, 16, 88

则采用的排序方法可能是_____。

A: 起泡排序

B: 希尔排序

C: 归并排序

D: 基数排序

得 分

二、(本题 10 分) 在任意一棵非空二叉排序树 (二叉搜索树, BST) T1 中, 删除某结点后又将其插入, 则所得二叉排序树 T2 与原二叉排序树 T1 相比, 会有几种情况? 试证明你的结论。

删除的结点为 A。

分两种情况:

1 不变化: 删除的是叶结点。

2 有变化: 删除的不是叶结点。再分以下两种情况。

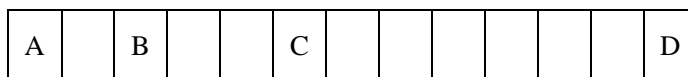
度为 1: 为其原孩子结点 B 的后代 (子孙) 结点。具体地, 若 B 是 A 的左孩子, 插入到 B 的右子树中; 若 B 是 A 的右孩子, 插入到 B 的左子树中。

度为 2: 若使用 A 的直接前驱 C 替代, 插入到 C 的右子树中; 若使用 A 的直接后继 D 替代, 插入到 D 的左子树中。

证略。

得 分

三、(本题 8 分) 用一维数组存放的一棵二叉树如下图所示:

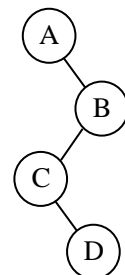


画出该二叉树, 并分别写出先序、中序及后序遍历该二叉树时访问结点的顺序。

先序: ABCD

中序: ACDB

后序: DCBA



得分

四、(本题 12 分) 有以下 10 个关键字：28，72，97，63，4，53，84，32，61，52，使用归并排序方法将所给关键字排成升序序列，给出排序过程。

初始：28，72，97，63，4，53，84，32，61，52
 i=1： 28，72，63，97，4，53，32，84，52，61
 i=2： 28，63，72，97，4，32，53，84，52，61
 i=3： 4，28，32，53，63，72，84，97，52，61
 i=4： 4，28，32，52，53，61，63，72，84，97

得分

五、(本题 10 分) 设一个哈希表的地址区间为 0-16, 哈希函数为 $H(K)=K \bmod 17$ 。采用线性探测法处理冲突，请将关键字序列 19，14，23，01，68，20，84，27，55，11，10，79，12 依次存储到哈希表中，画出结果，并计算平均查找长度。

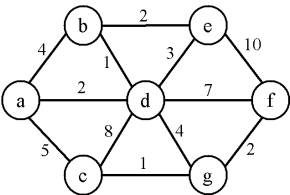
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
68	01	19	20	55		23				27	11	10	79	14	12	84
1	1	1	1	1		1				1	1	3	3	1	4	1

ASL=20/13

得分

六、(本题 15 分) 对右面的带权图，回答下列问题。

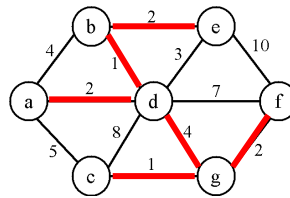
- 1) 给出每个顶点的度。
- 2) 画出图的邻接矩阵。
- 3) 使用 Prim 算法求图的最小生成树。



顶点	a	b	c	d	e	f	g
度	3	3	3	6	3	3	3

邻接矩阵

	4	5	2			
4			1	2		
5			8			1
2	1	8		3	7	4
	2		3		10	
			7	10		2
		1	4		2	



得分

七、(本题 15 分) 一个长度为 L ($L \geq 1$) 的升序序列 S , 处在第 $\lceil L/2 \rceil$ 个位置的数称为 S 的中位数。例如, 若序列 $S_1=(11, 13, 15, 17, 19)$, 则 S_1 的中位数是 15。两个序列的中位数是含它们所有元素的升序序列的中位数。

例如, 若 $S_2=(2, 4, 6, 8, 20)$, 则 S_1 和 S_2 的中位数是 11。现有两个等长升序序列 A 和 B , 试设计一个在时间和空间两方面都尽可能高效的算法, 找出两个序列 A 和 B 的中位数。

```
int M_Search ( int A[], int B[], int n )
```

```
{   int start1, end1, mid1, start2, end2, mid2;
```

```
    start1 = 0; end1 = n-1;
```

```
    start2 = 0; end2 = n-1;
```

```
    while ( start1 != end1 || start2 != end2 )
```

```
    {   mid1 = (start1 + end1) / 2;
```

```
        mid2 = (start2 + end2) / 2;
```

```
        if ( A[ mid1 ] == B[ mid2 ] ) return A[ mid1 ];
```

```
        if ( A[ mid1 ] < B[ mid2 ] )
```

```
        { // 分别考虑奇数和偶数, 保持两个子数组元素个数相等
```

```
            if ( ( start1 + end1 ) % 2 == 0 )
```

```
            { // 若元素为奇数个
```

```
                start1 = mid1; // 舍弃 A 中间点以前的部分且保留中间点
```

```
                end2 = mid2; // 舍弃 B 中间点以后的部分且保留中间点
```

```

    }
    else
    { // 若元素为偶数个
        start1 = mid1 + 1; // 舍弃 A 的前半部分
        end2 = mid2; // 舍弃 B 的后半部分
    }
}
else
{   if ( ( start1 + end1 ) % 2 == 0 )
    { // 若元素为奇数个
        end1 = mid1; // 舍弃 A 中间点以后的部分且保留中间点
        start2 = mid2; // 舍弃 B 中间点以前的部分且保留中间点
    }
    else
    { // 若元素为偶数个
        end1 = mid1; // 舍弃 A 的后半部分
        start2 = mid2+1; // 舍弃 B 的前半部分
    }
}
}
return A[start1] < B[start2] ? A[start1] : B[start2];
}

```