2015-2016 学年度第一学期软件学院 2014 级本科

《数据结构》课程考试试卷（A 卷）

专业、班级：_____ 姓名：_____学号：_____成绩：_____ 考试日期： 2016 年 1 月 24 日 考试时间： 150 分钟

说明：

- 本试卷中使用的数据结构如下： typedef
  struct bINTREENODE
  {
  int data;
  struct bINTREENODE *leftChild;
  struct bINTREENODE *rightChild;
  } BINTREENODE;

- 每题 10 分，请合理安排时间。如果有编程困难，可以使用伪代码或文字表达以得到部

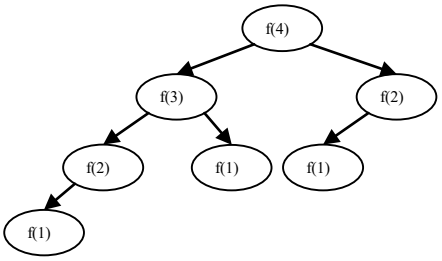
  分分数。

1. 一个递归函数具有如下形式

```
void func ( int x )
{
  if ( x>0)  {     func ( x -
1);     printf ( “%d,”, x*x);
func ( x -2);
  }  return;
}
```

请依次写出 func（1），func（2），func（3），和 func（4）的执行结果，画出递归调用树，

并分析 func (n)的计算时间复杂度，以 O 的形式给出，要求给出分析过程。



func(1): 1,

func(2):1,4,

func(3):1,4,9,1

func(4):1,4,9,1,16,1,

4 T(n)=T(n-1)+T(n-

2)+O(1)

T(0)=O(1)

F(n)=T(n)+O(1)，因此等价于 Fibonacci 数

T(n)=O(ψ^n), ψ=1.618

2，(5^200) % 7=？写出你的计算依据和推导过程。

根据费马小定理，5^6 % 7 = 1, 而 5^200 = (5^6)^33*5^2，因此 5^200 % 7 = 25 % 7 = 4

3，编写一个 C 语言函数 unsigned char isBST (BINTREENODE *root)，判断一棵以 root 为根

节点的二叉树是不是 BST，如果是返回 1，否则返回 0。 需要判断对所有节点，其左子树所有节点的值都小于该节点的值，其右子树所有节点的值都

大于该子树（假定所有值均为正数）。

或者使用中序遍历，判断是否升序排列（只需保存遍历时得到的最大值，与当前节点比较，并更新为当前节点；否则判为不是 BST）。

```
int is_BST (BINTREENODE *root, int m)
{
      if (root == NULL || m == -1)
     return (m);

  m = is_BST (root->left, m); if
     (root->data < m)    return (-
     1); m = root->data; m =
     is_BST (root->right, m);
     return (m);
}

unsigned char isBST (BINTREENODE *root)
{
      if (root != NULL && is_BST (root, 0) < 0)
      return (0);

      return (1);
}
```

4，构造一棵 n 个结点的 BST 的平均时间复杂度用 O 表示是多少？证明你的结论。

O(nlog n)

因为 BST 节点插入为树叶，在空树中依次插入节点得到 BST 的代价等同于所有节点深度之和。令 n 个结点的 BST 的深度之和为 T(n)，则

T(n) = T(m) + T(n-1-m) + n-1，其中 m 为左子树节点个数，n-1-m 为右子树节点个数，等同于根节点令左右子树节点加深一层 对于平均情况，m 可以等概率取 0 到 n-1 中的任何值，对 n 种不同情况求和，有 nT(n) = 2 $\sum_{m=0}^{n-1} T(m)$ + n(n-1)

带入 n-1 的情况

(n-1)T(n-1) = 2 $\sum_{m=0}^{n-2} T(m)$ + (n-1) (n-2) 两式相减

nT(n) - (n-1)T(n-1) = 2T(n-1) + 2(n-1)

nT(n) = (n+1)T(n-1) + 2(n-1)，两边同时除以 n (n+1)，近似地

T(n) /(n+1) = T(n-1)/n + 2/ n

T(n-1) /(n) = T(n-2)/(n-1) + 2/(n-1)

……

T(1) = 1

利用调和序列求和与欧拉常数

T(n)=(n+1) ∑1/m=O(n ln n) =O(n log n)

5，设计并编写一个 C 语言函数 unsigned char isIdentical (int a[], unsigned int n)，判断给定的长度为 n 的元素各不相同且已按升序排序的数组 a 中是否存在一个元素等于其索引值，即 a[i]=i，如果存在返回 1，否则返回 0。 要求算法的时间复杂度为 O(log n)。 分治算法，比较 a[mid] 和 mid ，如果 a[mid] = mid，返回 1;否则如果 a[mid]< mid，在右侧找；否则在左侧找

```
unsigned char is_identical (int a[], int first,  int last)
{
   int mid = (first + last)/2;

   if (first > last)
      return (0);

   if (a[mid] == mid)
return (1);

   if (a[mid] < mid)
     return (is_identical (a, mid+1, last));

   return (is_identical (a, first, mid-1));
}

unsigned char isIdentical (int a[], unsigned int n)
{
   return (is_identical (a, 0, n-1));
}
```

6，对数组{10，8，26，16，28，100，110，7，77，92，17，78，207，18，97}进行以 10 为基的基数排序。要求用图或表表示排序过程，并写出在每一趟中的回收结果。

个位

110 100 10		92				16 26	207 97 17 77 7	78 18 28 8	
0	1	2	3	4	5	6	7	8	9

回收结果:10->100->110->92->26->16->7->77->17->97->207->8->28->18->78 十位

8 207 7 100	18 17 16 110 10	28 26					78 77		97 92
0	1	2	3	4	5	6	7	8	9

回收结果:100->7->207->8->10->110->16->17->18->26->28->77->78->92->97

百位

97 92 78 77 28 26 18 17 16 10 8 7	110 100	207							
0	1	2	3	4	5	6	7	8	9

回收结果: 7->8->10->16->17->18->26->28->77->78->92->97->100->110->207

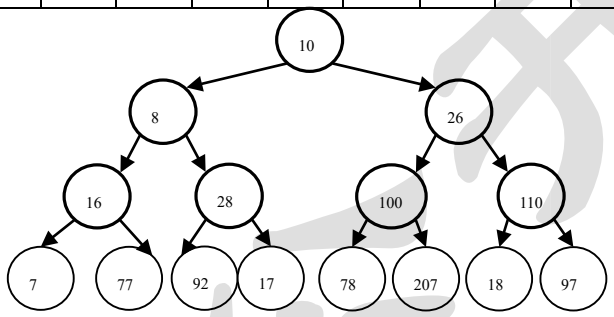
7，把数组{10，8，26，16，28，100，110，7，77，92，17，78，207，18，97}调整为一个最大堆，并给出排序的结果，要求用图或表表示建堆及排序的过程。

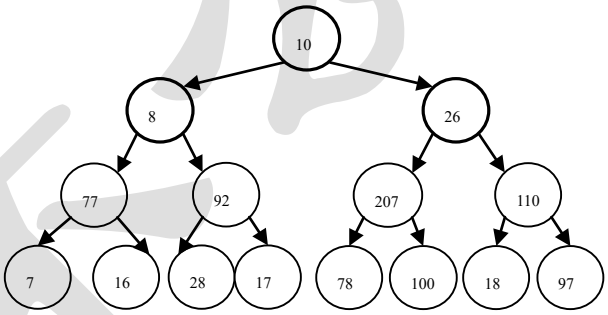
建堆过程 初始数组

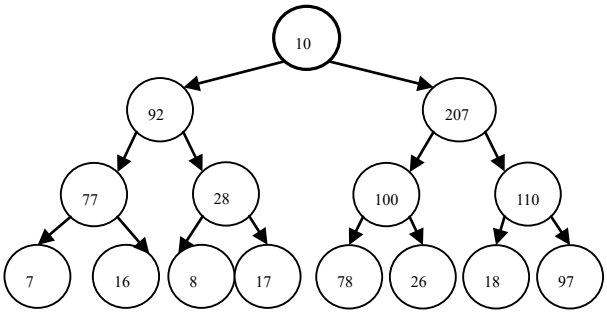
10	8	26	16	28	100	110	7	77	92	17	78	207	18	97

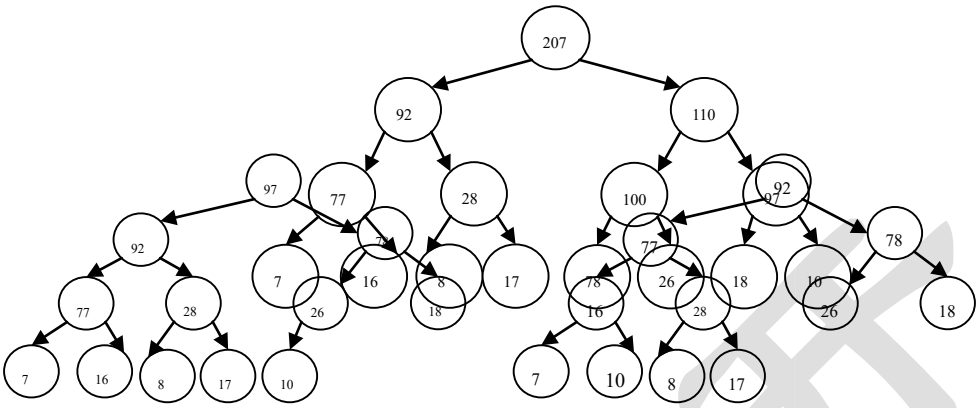
树表示（加粗表明需要调整的节点）

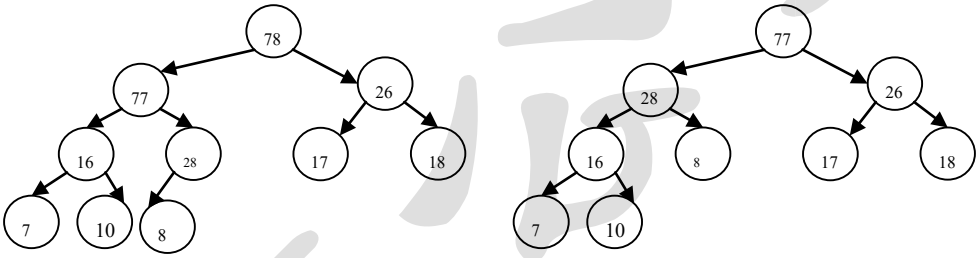
207	92	110	77	28	100	97	7	16	8	17	78	26	18	10



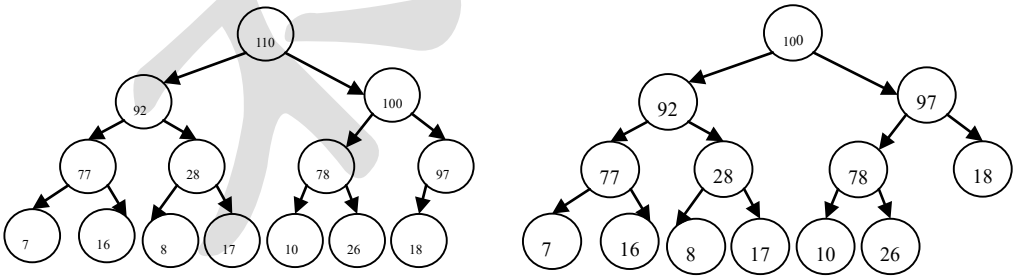




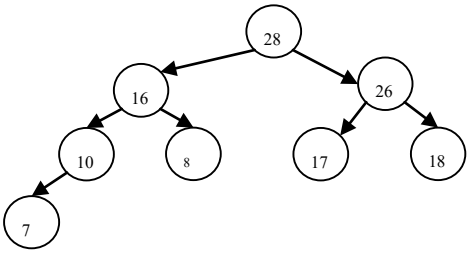


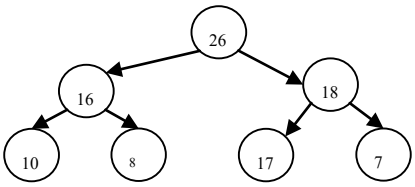


排序

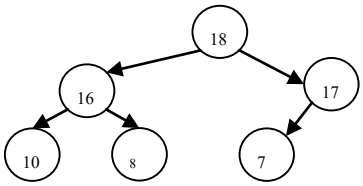


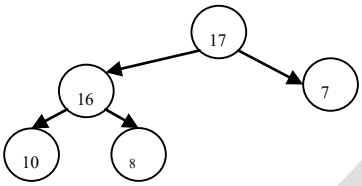
110	92	100	77	28	78	97	7	16	8	17	10	26	18	207
100	92	97	77	28	78	18	7	16	8	17	10	26	110	207
97	92	78	77	28	26	18	7	16	8	17	10	100	110	207
92	77	78	16	28	26	18	7	10	8	17	97	100	110	207
78	77	26	16	28	17	18	7	10	8	92	97	100	110	207
77	28	26	16	8	17	18	7	10	78	92	97	100	110	207



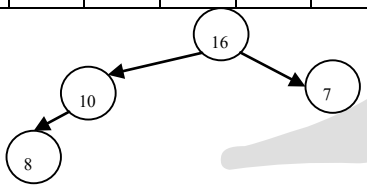


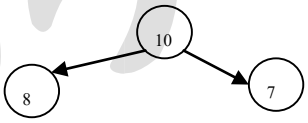
28	16	26	10	8	17	18	7	77	78	92	97	100	110	207
26	16	18	10	8	17	7	28	77	78	92	97	100	110	207



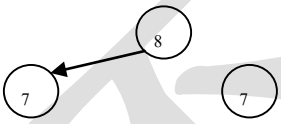


18	16	17	10	8	7	26	28	77	78	92	97	100	110	207
17	16	7	10	8	18	26	28	77	78	92	97	100	110	207



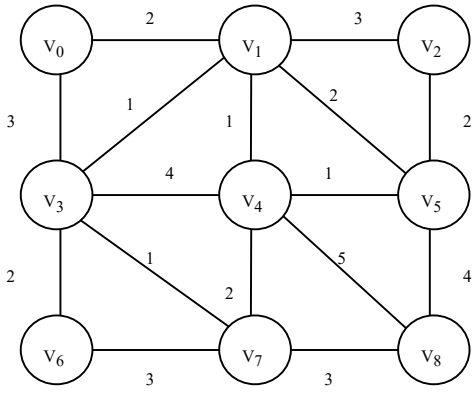


16	10	7	8	17	18	26	28	77	78	92	97	100	110	207
10	8	7	16	17	18	26	28	77	78	92	97	100	110	207



8	7	10	16	17	18	26	28	77	78	92	97	100	110	207
7	8	10	16	17	18	26	28	77	78	92	97	100	110	207

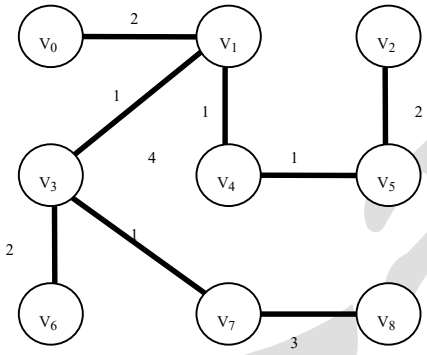
8，写出下图的邻接矩阵，给出用 Prim’s 算法得到其最小生成树的过程。



邻接矩阵

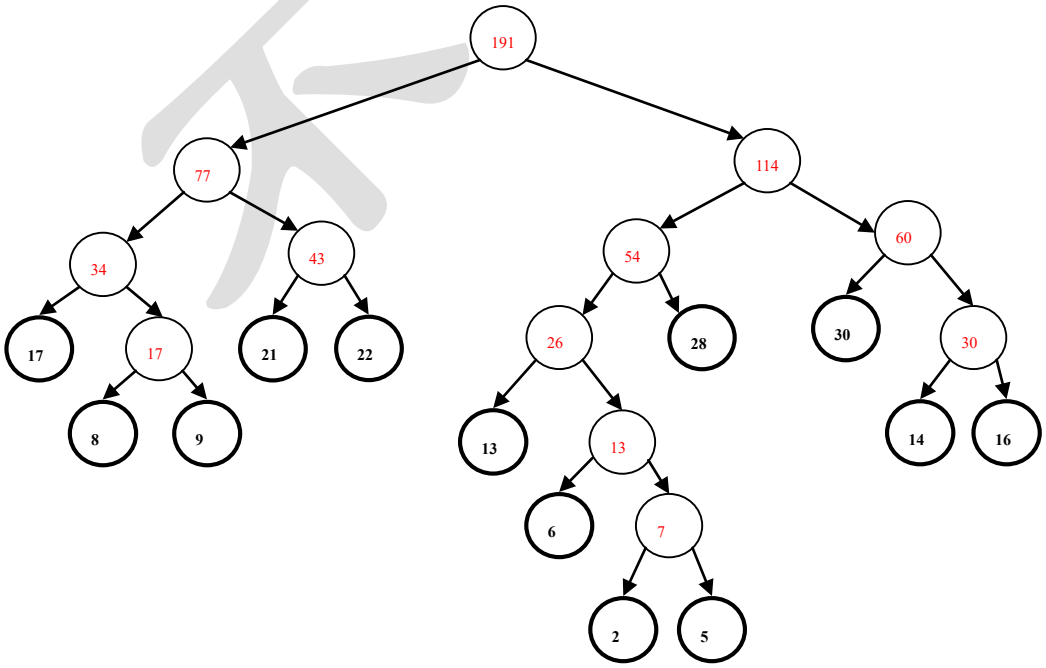
$$\begin{bmatrix} \infty & 2 & \infty & 3 & \infty & \infty & \infty & \infty & \infty \\ 2 & \infty & 3 & 1 & 1 & 2 & \infty & \infty & \infty \\ \infty & 3 & \infty & \infty & \infty & 2 & \infty & \infty & \infty \\ 3 & 1 & \infty & \infty & 4 & \infty & 2 & 1 & \infty \\ \infty & 1 & \infty & 4 & \infty & 1 & \infty & 2 & 5 \\ \infty & 2 & 2 & \infty & 1 & \infty & \infty & \infty & 4 \\ \infty & \infty & \infty & 2 & \infty & \infty & \infty & 3 & \infty \\ \infty & \infty & \infty & 1 & 2 & \infty & 3 & \infty & 3 \\ \infty & \infty & \infty & \infty & 5 & 4 & \infty & 3 & \infty \end{bmatrix}_{9\times9}$$

最小生成树



9，一组符号 S_i, i = 0..12,其出现的频率分别是 5，8，22，13，14，6，17，9，2，16, 28， 30 和 21。请设计出相应的 Huffman 编码。要求画出 Huffman 树，并给出编码。编码可能不唯一。

Huffman 树



Huffman 编码(左 0 右 1)

S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12

100111	0010	011	1000	1110	10010	000	0011	100110	1111	101	110	010

10，设计一个算法，并编写相应的 C 语言函数 int selectInTwoArrays (int a[], int b[], unsigned int n, unsigned int m, unsigned int k)，从长度分别为 n 和 m 的已按升序排序的数组 a 和 b 的并集中挑选出第 k 大的值并返回该值，1≤k≤m + n。 要求算法的时间复杂度为 O (log m + log n) ，需要证明你的结果。 分治算法

取两个数组的中位值，比较大小：若相等，则找到了并集的中位值；若不等，则知其中一个数组的前一半小于中位值，另一数组的后一半大于中位值；将 k 与(m+n)/2 比较，则可在缩小的空间中搜索。

注意数组的长度可能是奇数，因此在起始/结束位置上应作处理（从 a[0]到 a[m/2-1]共有 m/2 个元素，而从 a[m/2]到 a[m-1]共有(m+1)/2 个元素）。

```
int select_in_2arrays (int a[], int b[], int f1, int l1, int f2, int l2, int k)
{
      int m1 = (f1 + l1)/2, m=l1-f1+1;
      int m2 = (f2 + l2)/2, n=l2-f2+1;

      if (f1 > l1) return
            (b[k-1]);
      if (f2 > l2) return
      (a[k-1]); if (m == 1
      && n==1)
      {
         if (k == 1)
        return (a[0] < b[0] ? a[0] : b[0]);
return (a[0] > b[0] ? a[0] : b[0]);    }

      if (a[m1] == b[m2])
      {
            if (k == (m + n)/2)
           return (a[m1]);
            if (k < (m + n)/2)
           return (select_in_2arrays (a, b, f1, m1-1, f2, m2-1, k)); return
            (select_in_2arrays (a, b, m1, l1, m2, l2, k-(m+n)/2));
      }
      if (a[m1] < b[m2])
      {
            if (k == (m + n)/2)
           return (select_in_2arrays (a, b, m1, l1, f2, m2-1, k-m/2)); if (k <
            (m + n)/2)
           return (select_in_2arrays (a, b, f1, l1, f2, m2-1, k)); return
            (select_in_2arrays (a, b, m1, l1, f2, l2, k-m/2));
      }
      if (a[m1] > b[m2])
      {
            if (k == (m + n)/2)
```

```
            return (select_in_2arrays (a, b, f1, m1-1, m2, l2, k-n/2)); if (k <
             (m + n)/2)
            return (select_in_2arrays (a, b, f1, m1-1, f2, l2, k));
return (select_in_2arrays (a, b, f1, l1, m2, l2, k-n/2));    }
}
int selectInTwoArrays (int a[], int b[], unsigned int n, unsigned int m, unsigned int k)
{
      if (k < 1 || k > (m+n))
      return (-1); if (m < 0
      || n < 0)    return (-
      2);
      return (select_in_2arrays (a, b, 0, m-1, 0, n-1, k));
}
```

复杂度分析

最坏情况下 T(m,n)=T(m)+T(n/2)+ O(1), T(1,1)=O(1)（m 和 n 可以互换）；因此，经过 log n 次迭代得到 T(m,1)，这时的最坏情况为 T(m,n)=T(m /2)+ O(1)，经过 log m 次迭代得到 T(1,1)。总的时间复杂度为 O(log m + log n)。