上 海 交 通 大 学 试 卷（A卷）

（ 2015 至 2016 学年 第 一 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称 《数据结构（A类）》 成绩 _____

一、单项选择题（每格 1.5 分，共 24 分）

1. 在以下各种查找方法中，平均查找时间与结点个数n无关的查找方法是 _____。

A. 顺序查找　　B. 折半查找　　C. 哈希查找　　D. 分块查找

2. 已知邻接矩阵如右图所示，从结点0出发，不能按深度优先遍历的结点序列是 _____。

A. 0, 2, 4, 3, 1, 5, 6
B. 0, 1, 3, 5, 6, 4, 2
C. 0, 4, 2, 3, 1, 6, 5
D. 0, 1, 3, 4, 2, 5, 6

0	1	1	1	1	0	1
1	0	0	1	0	0	1
1	0	0	0	1	0	0
1	1	0	0	1	1	0
1	0	1	1	0	1	0
0	0	0	1	1	0	1
1	1	0	0	0	1	0

3. 对于一个头指针为head的带头结点的单链表，判定该表为空表的条件是 _____。

A. head==NULL　　B. head->next==NULL
C. head->next==head　　D. head!=NULL

4. 在二叉树结点的前序序列，中序序列和后序序列中，所有叶子结点的先后顺序 _____。

A. 完全相同　　B. 前序和中序相同，而与后序不同
C. 都不相同　　D. 中序和后序相同，而与前序不同

5. 下列选项给出的是从根分别到达两个叶节点路径上的权值序列，能属于同一棵哈夫曼树的是 _____。

A. 24, 10, 5 和 24, 10, 7　　B. 24, 10, 5 和 24, 12, 7
C. 24, 10, 10 和 24, 14, 11　　D. 24, 10, 5 和 24, 14, 6

6. 时间复杂度不受数据初始状态影响而恒为 $O(N\log_2 N)$ 的是 _____。

A. 堆排序　　B. 冒泡排序　　C. 希尔排序　　D. 快速排序

7. 快速排序在最坏情况下的时间复杂度为 _____。

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

我承诺，我将严格遵守考试纪律。

承诺人： _____

题号	一	二	三	四	五	六	总分
得分							
批阅人(流水阅) 卷教师签名处)							

8. 设有50000个待排序的记录关键字，如果需要用最快的方法选出其中最小的10个记录关键字，则用下列 _____ 方法可以达到此目的。

A. 快速排序　　B. 选择排序　　C. 归并排序　　D. 插入排序

9. 设某散列表的长度为100，散列函数H(k)=k mod P，则P通常情况下最好选择 _____。

A. 99　　B. 97　　C. 91　　D. 93

10. 设一组初始关键字记录关键字为(20, 15, 14, 18, 21, 36, 40, 10)，则以20为基准记录的一趟快速排序结束后的结果为 _____。

A. 10, 15, 14, 18, 20, 36, 40, 21　　B. 10, 15, 14, 18, 20, 40, 36, 21
C. 10, 15, 14, 20, 18, 40, 36, 21　　D. 15, 10, 14, 18, 20, 36, 40, 21

11. 下列关键字序列中 _____ 是最大堆。

A. 94, 23, 31, 12, 16, 13　　B. 16, 72, 31, 23, 94, 53
C. 16, 23, 53, 31, 94, 72　　D. 16, 53, 23, 94, 31, 72

12. 对有14个元素的有序表A[1..14]作二分查找，查找元素A[6]时的被比较元素依次为 _____。

A. A[1], A[2], A[3], A[4]　　B. A[1], A[14], A[7], A[4]
C. A[7], A[5], A[3], A[6]　　D. A[7], A[3], A[5], A[6]

13. 判断一个有向图中是否存在回路，下列选项中两种算法均可行的是 _____。

A. Dijkstra 算法和深度优先遍历算法　　B. 深度优先遍历算法和拓扑排序方法
C. 拓扑排序方法和 Kruskal 算法　　D. Kruskal 算法和 Floyd 算法

14. 将长度为N的单链表链接在长度为M 的单链表之后的算法的时间复杂度为 _____。

A. O(1)　　B. O(N)　　C. O(M)　　D. O(M+N)

15. 非递归后序遍历二叉树时，考察栈中的数据：若当前结点为x，且x没有右儿子，这时栈顶的结点为y，且它的TimesPop值为2，则栈顶结点y必是x的 _____。

A. 前序的后件　　B. 中序的后件　　C. 后序的前件　　D. 后序的后件

16. 下面关于 B 树和 B+树的叙述中，不正确的是 _____。

A. 都是平衡的多叉树　　B. 都能有效地支持顺序检索
C. 都可以用于文件的索引结构　　D. 都能有效地支持随机检索

二、程序填充题（每格 1.5 分，共 24 分）

1. 假设 root 是一棵给定的非空二叉查找树的根指针，对于下面给出的函数 del_leaf，当执行 del_leaf(root, key);时，就可以实现如下的操作：在非空二叉查找树中查找值为 key 的结点；若值为 key 的结点在树中存在，并且是一个叶子结点，则删除此叶子结点，同时函数返回 true；若值为 key 的结点不在树中，或者虽然在树中，但不是叶子结点，则不进行删除，同时函数返回 false。应注意到非空二叉查找树只包含一个结点情况，此时树中的唯一结点，既是根结点，也是叶子结点。

```
struct Node {
    int key;
    Node *left, *right;
};
bool del_leaf (Node *&t, int key) {
    Node *p, *pf;
    bool found = false;
    p = t;
    while (_____ && _____) {
        if (key == p->key) found = true;
        else {
            _____
            if (key < p->key) p = p->left;
            else p = p->right;
        }
    }
    if (found && p->left == NULL && p->right == NULL) {
        if (_____)
        if (pf->left == p) pf->left = NULL;
        else pf->right = NULL;
        else _____
        delete p;
    }
    else found = false;
    return found;
}
```

2. 前序遍历打印二叉树上的结点信息，请完整之。其中的 linkStack 为教材中实现的链接栈类。

```
Struct BNODE {
    int info;
    BNODE *left, *right;
}
void printPreOrder (BNODE *root) {
    BNODE * ptr = root;
    linkStack <BNODE*> stack;
    while (_____ || _____) {
        if (_____) {
            cout << ptr->info << "\t";
            stack.push (ptr);
            _____
```

```
        }
        else {
            ptr = stack.pop ();
            ____________________________
        }
    }
    cout << endl;
}
```

3. 下列算法为奇偶交换排序，思路为，第一趟对所有奇数的 i，将 a[i]和 a[i+1]进行比较，第二趟对所有偶数的 i，将 a[i]和 a[i+1]进行比较，每次比较时若 a[i]>a[i+1]，将二者交换，以后重复上述两趟过程，直至整个数组有序。完成以下代码：

```
void oesort (int a[], int length) {
    int i, t, n = length-1;
    bool flag;
    do {
        ____________________________
        for (i = 0; i < n; ____________________________) {
            if (a[i] > a[i+1]) {
                flag = true;
                ____________________________
                a[i+1] = a[i];
                a[i] = t;
            }
        }
        for (i = 1; i < n; ____________________________) {
            if (a[i] > a[i+1]) {
                flag = true;
                ____________________________
                a[i+1] = a[i];
                a[i] = t;
            }
        }
    } while (____________________________);
}
```

三、简答题（每题 8 分，共 24 分）

1. 在外排序过程中，经常会使用置换选择技术来进行预处理。请对下列数据采用置换选择：5, 2, 34, 10, 4, 23, 3, 54, 33, 1, 7, 12, 26, 11, 40, 18, 35, 15, 27。假设内存只能存放 3 个元素，请问：能生成多少个初始的已排序片段？每个已排序片段包含哪些数据？

2. 已知有向图有 6 个顶点，边的输入序列如下：
<1, 2>, <1, 3>, <3, 2>, <3, 0>, <4, 5>, <5, 3>, <0, 1>
求该图的邻接表，强连通分量的个数。

3. 已知一棵二叉树的前序遍历的结果是ABKCDFGHIJ，中序遍历的结果是KBCDAFHIGJ，试画出这棵二叉树。

四、分析题（20 分）

1. 下面是基于邻接矩阵的 Kruskal 函数，请分析其时间复杂度。（8 分）

```
struct edgeNode {
    int beg, end;
    TypeOfEdge w;
    bool operator < (const edgeNode &rp) const {
        return w < rp.w;
    }
};
template <class TypeOfVer, class TypeOfEdge>
void adjMatrixGraph <TypeOfVer, TypeOfEdge> :: function() const {
    int edgesAccepted = 0;
    int u, v, i, j;
    DisjointSet  ds (Vers);
    priorityQueue <edgeNode> pq;
    edgeNode e;
    for (i = 0; i < Vers; ++i) {
        for (j = 0; j < Vers; ++j) {
            if (edge[i][j] != noEdge) {
                e.beg = i;
                e.end = j;
                e.w = edge[i][j];
                pq.enQueue (e);
            }
        }
    }
    While (edgesAccepted < Vers-1) {
        e = pq.deQueue ();
        u = ds.Find (e.beg);
        v = ds.Find (e.end);
        if (u != v) {
            edgesAccepted++;
            ds.Union (u, v);
            cout << '(' << ver[e.beg] << ',' << ver[e.end] << ")\t";
        }
    }
}
```

2. 下面是采用优先级队列实现的 Dijkstra 算法：

```
template <typename TypeOfVer, typename TypeOfEdge)
void adjListGraph <TypeOfVer, TypeOfEdge>
   :: diskstra (TypeOfVer start, TypeOfEdge noEdge) const {
   TypeOfEdge *distance = new typeOfEdge [Vers];
   int *prev = new int [Vers];
   bool *known = new bool [Vers];
   int *hop = new int [Vers];
   int sNo, i;
   edgeNode *p;
   priorityQueue <queueNode> q;
   queueNode min, succ;
   for (i = 0; i < Vers; ++i) {                    // 初始化
      known[i] = false;
      distance[i] = noEdge;
      hop[i] = 0;
   }
   for (sNo = 0; sNo < Vers; ++sNo)                // 寻找源点的序号
      if (verList[sNo].ver == start) break;
   if (sNo == Vers) {
      cout << '起始结点不存在' << endl;
      return;
   }
   distance[sNo] = 0;
   prev[sNo] = sNo;
   min.dist = 0;
   min.node = sNo;
   q.enQueue (min);
   while (!q.isEmpty ()) {
      min = q.deQueue ();                          //_____________________________
      if (known[min.node]) continue;
      known[min.node] = true;
      //_____________________________ (for 语句功能)
      for (p = verList[min.node].head; p != NULL; p = p->next)
         if (!known[p->end] &&
            (distance[p->end] > min.dist+p->weight|| //_____________________________
             distance[p->end] == min.dist+p->weight &&
             hop[p->end] > hop[min.node]+1)) {
            succ.dist = distance[p->end] = min.dist+p->weight;
            prev[p->end] = min.node;
            hop[p->end] = hop[min.node]+1;
            succ.node = p->end;
            q.enQueue (succ);
         }
   }
   for (i=0; i<Vers; ++i) {                        // 输出路径
```

```
        cout << start << "to" << verList[i].ver << ':';
        printPath (sNo,i,prev);
        cout << "\tLength:" << distance[i] << endl;
    }
}
// 输出序号为 start 的结点到序号为 end 的结点的路径
template <typename TypeOfVer, typename TypeOfEdge)
void adjListGraph <TypeOfVer, TypeOfEdge>
    :: printPath (int start, int end, int prev[]) const {
    if (start == end) {
        cout << verList[start].ver;
        return;
    }
    printPath (start, prev[end], prev);
    cout << '-' << verList[end].ver;
}
```

请为 Dijkstra 算法下划线处添加适当的注释，并分析该算法的时间复杂度。

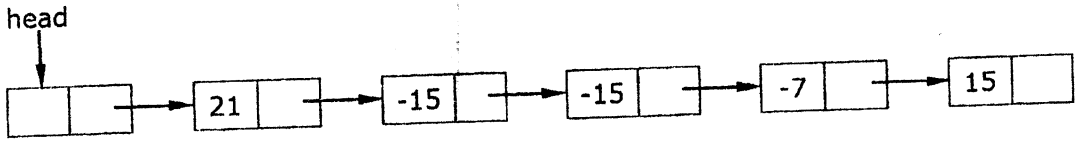
已知无负权值有向图 G(V, E)，其中 V(G)={a, b, c, d, e, f, g}，E(G)={<a, b, 2>, <a, d, 1>, <b, d, 7>, <b, e, 2>, <c, a, 4>, <c, f, 8>, <d, c, 2>, <d, e, 1>, <d, f, 3>, <d, g, 3>, <e, g, 4>, <g, f, 5>}。设顶点 a, b, c, d, e, f, g 的序号依次为 0~6，有向边<u, v, w>表示顶点 u 到顶点 v 有一条边，权值为 w，将图保存在邻接表类的对象 g 中，对 g 调用 dijkstra 函数 g.dijkstra('b', 255)，请输出运行结果。（12 分）

五、程序题（8 分）

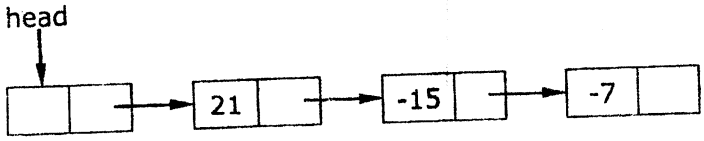
已知整数数组 int a[n]。其中，a[1], a[2], ..., a[n]已被整理为最小化堆。请设计一算法，在最短的时间内，找出 a[1], a[2], ..., a[n]数组元素中最大的元素的下标，并且简单证明其时间复杂度。

六、附加题（10 分）

用单链表保存 m 个整数，结点的结构为(data, link)，且|data|<n (n<m)。现要求设计一个时间复杂度尽可能高效的算法，对于链表中绝对值相等的结点，仅保留第一次出现的结点而删除其余绝对值相等的结点。例如若给定的单链表 head 如下：



删除结点后的 head 为：



要求：

(1) 给出算法的基本思想；

(2) 根据设计思想，采用 C++语言实现之；

(3) 说明所涉及算法的时间复杂度和空间复杂度。