第六章作业：树

2、一棵有 124 个叶子结点的完全二叉树，最多有（A）个结点。
错选答案：249
正确答案：248

5、一棵非空的二叉树的先序遍历序列与后序遍历序列正好相反，则该二叉树一定满足（ ）。
错选答案：所有的结点均无左孩子
正确答案：只有一个叶子结点

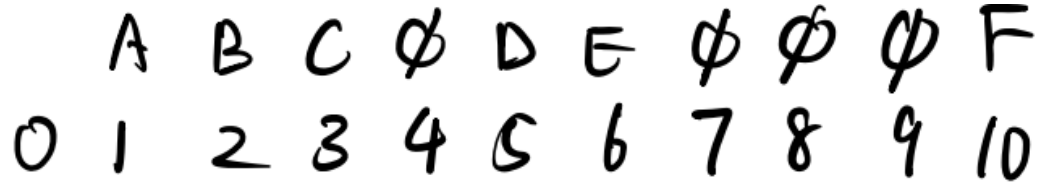
6、线索化二叉树中，某结点 p 没有孩子的充要条件是（ ）。
错选答案：p->lchild==NULL && p->ltag==1
正确答案：p->ltag==1 && p->rtag==1

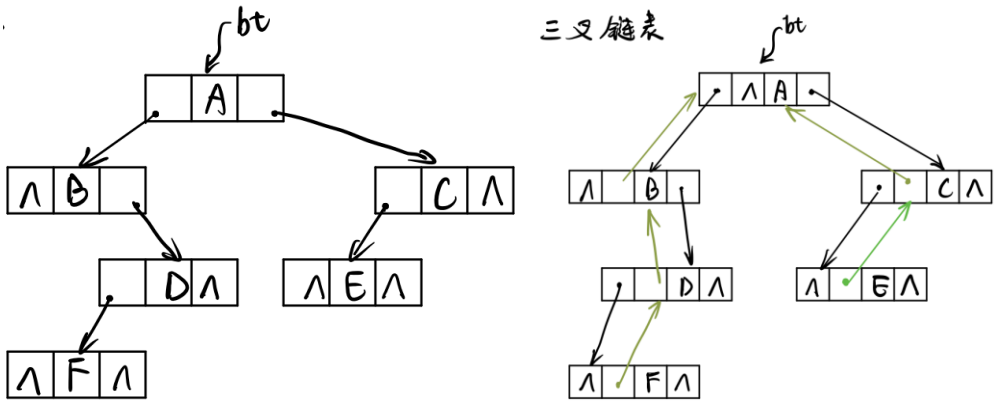
9、设森林 T 中有 4 棵树，第一、二、三、四棵树的结点个数分别是 n1、n2、n3、n4，那么当把森林 T 转换成一棵二叉树后，且根结点的右子树上有（ ）个结点。
错选答案：n1-1
n1+n2+n3
正确答案：n2+n3+n4

12、对 n 个权值均不相同的字符构成哈夫曼树，关于该树的叙述中，错误的是（ ）。
错选答案：树中一定没有度为 1 的结点
正确答案：该树一定是一棵完全二叉树

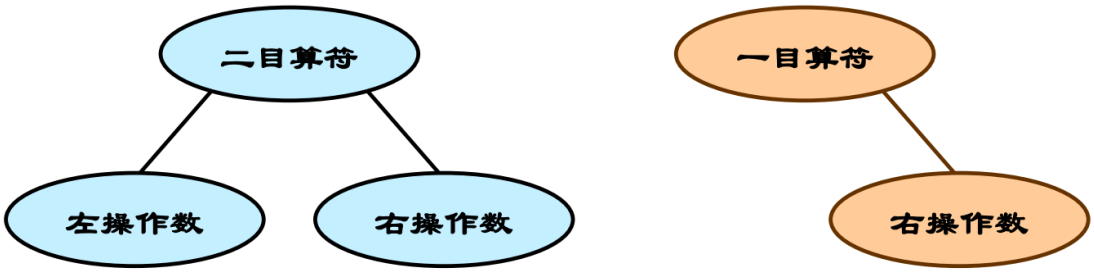
13、深度为 k 的完全二叉树至少有 **2^(k-1)**个结点，至多有 **(2^k)-1** 个结点。
14、设一棵完全二叉树的顺序存储结构中存储数据元素为 ABCDEF，则该二叉树的先序、中序、后序遍历序列分别为 **ABDECF、DBEAFC、DEBFCA。**
17、高度为 h 的严格（正则）二叉树至少有 **2h-1** 个结点，至多有**(2^h)-1** 个结点。
18、若以{6，14，53，15，12}作为叶子结点的权值构造哈夫曼树，则该树的根结点的权值为 **100**。

19、已知一棵二叉树的先序扩展序列为{A、B、#、D、F、#、#、#、C、E、#、#、#}，其中#表示虚结点。请分别画出它的顺序、二叉链表、三叉链表存储结构示意图。
问题：很多同学顺序存储结构只画出了一棵树。

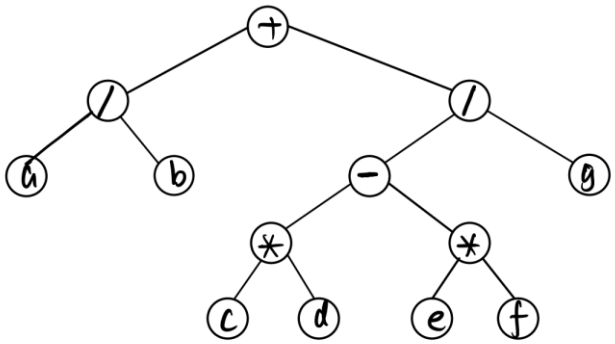




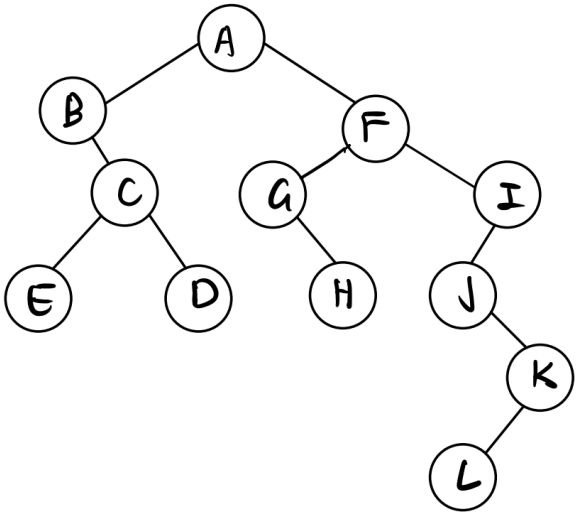
20、请画出与中缀表达式 a/b+(c*d-e*f)/g 所对应的一棵表达式树。
叶节点存放操作数，分支结点存放操作符



前缀表示（波兰式）：先序遍历(OP, S1, S2)
中缀表示：中序遍历(S1, OP, S2)
后缀表示（逆波兰式）：后序遍历(S1, S2, OP)



21、已知一森林 F 包含 3 棵树（分别称为 T1、T2 和 T3），其中 T1 按层次表达的树枝序列为{(#,A), (A,B), (A,C), (A,D), (C,E)}，符号#表示虚结点。类似地，T2 和 T3 所对应的树枝序列依次为{(#,F), (F,G), (F,H)}和{(#,I), (I,J), (I,K), (K,L)}。试画出该森林经转换后所对应的二叉树，并指出在孩子兄弟链表中的叶子结点所对应的森林中结点的特点。



在孩子兄弟链表中的叶子结点所对应的森林中结点的特点是：无左孩子

（森林中叶子结点对应在孩子兄弟链表中结点的特点：它们都是叶子结点且没有右兄弟）

22、已知二叉树用二叉链表存储，写一个算法将二叉树中的叶子结点按从右至左的顺序建立一个单链表。

问题：部分同学没有按先左后右的顺序遍历后没有逆序

算法思路：先序遍历二叉树，将叶子结点存入栈中，然后从栈中弹入单链表中；

或者按照 根→右子树→左子树 的顺序先序遍历，。

24、已知一棵二叉树 T 采用二叉链表结构存储，给定 p 和 q 两个结点，试编写一个算法，求与它们最近的共同祖先结点的地址。

算法思路：遍历二叉树找到 p 和 q 的所有祖先，然后比较找出最近的共同祖先。

若 root 是 p,q 的最近公共祖先，则只可能为以下情况之一：

1、p 和 q 在 root 的子树中，且分列 root 的异侧（即分别在左、右子树中）；

2、p=root，且 q 在 root 的左或右子树中；

3、q=root，且 p 在 root 的左或右子树中；

考虑通过递归对二叉树进行遍历，当遇到节点 p 或 q 时返回。从底至顶回溯，当节点 p,q 在节点 root 的异侧时，节点 root 即为最近公共祖先，则向上返回 root。

```
1.  TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
2.      if(root == nullptr || root == p || root == q) return root;
3.      TreeNode *left = lowestCommonAncestor(root->left, p, q);
4.      TreeNode *right = lowestCommonAncestor(root->right, p, q);
5.      if(left == nullptr) return right;
6.      if(right == nullptr) return left;
7.      return root;
8.  }
```

25、假设二叉树用二叉链表表示，试编写一算法，判别给定的二叉树是否为完全二叉树。

问题：部分同学的判定条件为倒数第二层结点有左结点无右节点，对于 ABCD#E#, AB#C###

两种情况容易出现误判。

算法思路：层序遍历读取二叉树，检测到某个结点为空时，后续结点应全为空。

```
1.  bool  Judge(BiTree T) {
2.      BiTree p;
3.      q = createQueue();
4.      EnQueue(q, T);//根结点入队
5.      while(!isEmpty(q))
6.      {
7.          p = DeQueue(q);//根结点出队
8.          if(p != nullptr)
9.          {
10.             EnQueue(q, p->lchild);//左孩子入队
11.             EnQueue(q, p->rchild);//右孩子入队
12.         }
13.         else
14.         {
15.             while(!isEmpty(q))
16.             {
17.                 p = DeQueue(q);
18.                 if(p != nullptr) return false;
19.             }
20.         }
21.     }
22.     return true;
23. }
```

第七章作业：图

2、设某强连通图中有 n 个顶点，则该强连通图中至少有（ ）条边。

错选答案：D.n(n-1)

正确答案：A.n

5、设有向图 G=(V,E)，顶点集 V={v0,v1,v2,v3}，边集 E={<v0,v1>,<v0,v2>,<v0,v3>,<v1,v3>}，若从顶点 v0 开始对图进行深度优先遍历，则可能得到的不同遍历序列个数是（）。

错选答案：B.4

正确答案：A.5

7、下列关于最小生成树的说法中正确的是（ ）。

错选答案：B.权值最小的边一定会出现在所有的最小生成树中

正确答案：D.最小生成树的代价唯一

17、有向图 G=(V,E)，其中 V(G)={0,1,2,3,4,5} ，用<a,b,d> 三元组表示弧<a,b> 及弧上的权 d。E(G) 为 {<0,5,100>,<0,2,10><1,2,5><0,4,30><4,5,60><3,5,10><2,3,50><4,3,20>}，则从

源点 0 到顶点 3 的最短路径长度是 **50**，经过的中间顶点是 **4**。

18、某图采用邻接表存储，其中 graph 为顶点表，边结点指针为 node_pointer。如下为对其进行拓扑排序的 C 程序，请在程序空白处填上适当语句。

```
1.  void topsort(hdnodes graph [], int n)
2.  {
3.      int i, j, k, top;     //top 表示同样入度为 0 的下一访问顶点
4.      node_pointer ptr ;
5.      top=-1;
6.      for (i=0; i<n; i++)  //count 表示该顶点入度
7.          if (!graph[i].count) { graph[i].count=top; top=i; }  //入度为 0 的顶点的 count 赋值为下一入度为 0 的顶点，最后一个 count 为 0，第一个位置给 top
8.      for (i=0; i<n; i++)
9.      //top==-1,没有入度为 0 的顶点，不构成 DAG 图；top!=-1,访问该入度为 0 的顶点
10.         if(top == -1) { fprintf(stderr, "\ngraph has a cycle \n"); exit(1); }
11.         else {j=top; top=graph[top].count; printf( "v%d, " ,j) ;  //打印入度为 0 的顶点；入度为 0 的顶点的 count 值表示下一个入度为 0 的顶点，赋给 top
12.             for (ptr=graph[j].link; ptr; ptr=ptr->link)  //遍历边结点
13.                 {k=ptr->vertex; graph[k].count--;  //边结点的 count 仍表示入度，--
14.                 if(!graph[k].count) { graph[k].count=top; top=k; }   //若有 count--后为 0，表示删去当前顶点后该边界节点入度为 0，把 top 赋给该边结点的 count，k 赋给 top，下一个访问的入度为 0 的结点为该边结点，该边结点的 count 串联下一个入度为 0 的结点
15.                 }
16.             }
17. }
```

20、已知一个有向图的顶点集{a, b, c, d, e, f, g, h, i}，按行描述的邻接矩阵如下：
[0，1，1，0，0，0，0，0，0；
0，0，0，1，1，0，0，0，0；
0，1，0，0，0，0，1，0，0；
0，0，0，0，0，0，0，0，0；
0，0，0，0，0，0，0，1，1；
0，0，1，0，1，0，0，0，0；
0，0，0，0，0，0，0，0，1；
0，0，0，0，0，0，0，0，0；
0，0，0，0，0，1，0，0，0]。
（1）画出其邻接表存储结构；（2）写出图的所有强连通分量；（3）写出顶点 a 到顶点 i 的全部简单路径。
错误情况：2、3 小题概念没弄清，漏写/错写
(1)
a->b->c
b->d->e
c->b->g

d
e->h->i
f->c->e
g->i
h->
i->f
(2)
强连通图： 有向图中任意两个顶点之间都强连通
强连通分量：有向图 G 的极大强连通子图
a, d, h, c-d-e-f-g-i
(3)
简单路径：序列中顶点不重复出现的路径
a-c-g-i
a-b-e-i
a-c-b-e-i

22、用邻接表存储图 G，设计一个算法，找出图中从顶点 u 到顶点 v 的长度为 n 的所有简单路径。
递归问题：终止条件，递推工作，返回值，是否回溯

```
int visited[MAX_VERTEX_NUM] = {0};

void DFS_All_SimplePath(Graph G, int u, int v, int n, int count, Stack *S) {
   //在连通图 G 上输入顶点 u 至顶点 v 的长度为 n 的全部简单路径，count 为当前栈中路径的长度
    int top_elem;
    Push(S, u);
    visited[u] = 1;     //给 u 打上访问标记
    if (u == v)
    {
        if (count == n) {
            StackTraverse(S);//打印出自栈底到栈顶的所有顶点序列
        }
        return;
    }
    //FirstAdjVex(G, u)为求邻接表 G 的顶点 u 行中顶点 u 的下一结点
    //NextAdjVex(G, u, w)为求邻接表 G 的顶点 u 行中顶点 w 的下一结点
    for (int w = FirstAdjVex(G, u); w != NULL; w = NextAdjVex(G, u, w)) {
        if (!visited[w]) {
            DFS_All_SimplePath(G, w, v, n, count + 1, S);     //下一层递归中 count+1
            Pop(S, &top_elem);     //令退出 DFS 遍历的顶点出栈
            visited[top_elem] = 0;     //将出栈顶点的访问标志恢复成 0
        }
    }
}
```

23、设有向 G 图有 n 个点(用 1,2,…,n 表示)，e 条边，写一算法根据其邻接表生成其逆邻接表，要求算法复杂度为 O(n+e)。

```
1.  void ReverseAdjList(ALGraph G, ALGraph RevG) {
2.      RevG.vexnum = G.vexnum;  //顶点
3.      RevG.arcnum = G.arcnum;  //边
4.      for (int i = 0; i < G.vexnum; i++) {  //表头结点
5.          RevG.vertices[i].data = G.vertices[i].data;
6.          RevG.vertices[i].firstarc = NULL;
7.      }
8.      //初始化逆转矩阵的表头结点数组等
9.      for (int i = 0; i < G.vexnum; i++) {
10.         ArcNode *temp;
11.         temp = G.vertices[i].firstarc;
12.         while (temp) {
13.             int j = temp->adjvex;      //当前边弧尾指向的顶点为 j
14.             ArcNode *pnew = (ArcNode *) malloc(sizeof(ArcNode));
15.             pnew->adjvex = i;
16.             pnew->nextarc = RevG.vertices[j].firstarc;
17.             RevG.vertices[j].firstarc = pnew;  //i 插入到 j 为顶点表的链域
18.             temp = temp->nextarc;      //下一条边
19.         }
20.     }
21. }
```

第八章作业：查找表

6、设有一组记录的关键字为{19，14，23，1，68，20，84，27，55，11，10，79}，用链地址构造散列表，散列函数为 H(key) = key mod 13，则散列地址为 1 的链中有（A）记录。

A.4

B.3

C.2

D.1

7、假定有 k 个关键字互为同义词，若用线性探测法把这 k 个关键字存入散列表中，至少要进行（A）次探测。

A.k*(k+1)/2

B.k+1

C.k

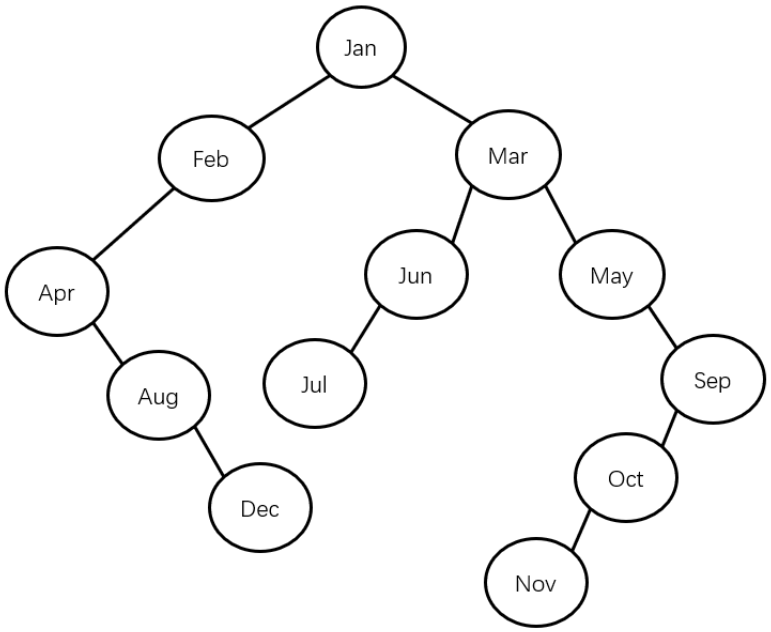
D.k-1

11、树深为 4 的 AVL 树， 最少具有（A）个结点。
A.7
B.20
C.12
D.5
平衡二叉树：所有结点的平衡因子只可能是-1，0，1

12、给定一棵 BST 和一个关键字，则从 BST 中找出值与给定关键字最接近的元素，最好采用（A）。
A.直接依关键字在 BST 中查找，记录最接近结点并更新最小差值
B.在 BST 上按层序遍历，记录最接近结点并更新最小差值
C.在 BST 上按中序遍历，记录最接近结点并更新最小差值
D.在 BST 上按先序遍历，记录最接近结点并更新最小差值

13、顺序查找 n 个元素的顺序表，若查找成功，则比较关键字的次数最多为 **n** 次；当使用监视哨时，若查找失败，则比较关键字的次数为 **n+1**。
14、在有序表 A[1..15]中，按折半查找方法进行查找，则查找长度为 4 的元素个数是 **8**，查找 A[10]时，所比较的元素下标依次为 **8，12，10**。
（折半查找的判定树：一定是平衡二叉树，且只有最下面一层是不满的）
16、如果按关键字递增的顺序将 n 个关键字插入到一棵二叉排序树中，则在这颗二叉排序树上进行查找时，平均查找长度为**(n+1)/2**；若按同样的次序插入到一棵 AVL 树中，则在该 AVL 树上进行查找时，平均查找长度为 **log(n)**。

18、**已知有一个长度为 12 的顺序查找表：{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec}，试按表中元素的顺序依次插入到一棵初始为空的二叉排序树中。画出插入完成后的二叉排序树，并求查找成功情况下的平均查找长度。**



ASL = (1+2*2+3*3+3*4+2*5+6)/12 = 3.5

19、对于索引顺序查找表，若索引表和各块内均用顺序查找，则有 900 个元素的线性表分成多少块最好？请给出你的理由。

最好分成 30 块。

假设分成 k 块，那么每块平均为 n/k 个元素，

k 块中平均查找长度为(k+1)/2,

每块中平均查找长度为(n/k+1)/2,

总的平均查找长度为(k+n/k+2)/2,

k = sqrt(n) = 30 时，总的平均查找长度取到最小值。

21、编写一个算法，利用折半查找算法在一个顺序存储的有序表中插入一个元素 x，并保持表的有序性。

```
1.  void insertK(int nums[],int n,int k){  //nums 顺序表，n 顺序表大小，k 插入元素
2.      int mid;// 记录中间下标
3.      int low=0,high=n-1;
4.      int flag=0;// 标志，用来记录是否有等于 k 的值
5.      int pos;// 定义的变量，为插入的位置
6.      while(low<=high&&flag==0){// 循环当 low>high 时跳出循环
7.          mid=(low+high)/2;
8.          if(nums[mid]==k){
9.              flag=1;//如果发现有关键字等于 k,则将标志 flag 置为 1，退出循环然后插入 k
10.         }else if(nums[mid]>k){
11.             high=mid-1;
12.         }else if(nums[mid]<k){
13.             low=mid+1;
14.         }
15.     }
16.     /* 确定插入位置 */
17.     if(flag==1){// 如果 flag 为 1 则发现序列中有关键字等于 k，则使插入位置等于 mid
18.         pos=mid;
19.     }else{// 如果 flag 为 0 则表示序列中没有与关键字 k 相等的值，则插入位置为 low
20.         pos=low;
21.     }
22.     /* 插入关键字 k */
23.     for(int i=n-1;i>=pos;i--){
24.         nums[i+1]=nums[i];
25.     }
26.     nums[pos]=k;
27. }
```

22、已知一棵二叉排序树上所有关键字中的最小值为-max，最大值为 max，又知道 x 在取值范围内，请编写算法，给出最接近 x 的左右两个数 a<x<b。

```
1.  void fun(BiSTree T, int x, int *a, int *b) {
2.      if (T)
3.          if (x < T->data.key)//当 x 小于根结点时，修改 b，在左子树上继续查找
```

```
        {
            *b = T->data.key;
            fun(T->lchild, x, a, b);
        } else if (x > T->data.key)//当 x 大于根结点时，修改 a，在右子树上继续查找
        {
            *a = T->data.key;
            fun(T->rchild, x, a, b);
        } else//当 x 等于根结点时，a 是其左子树的最右下结点，b 是其右子树的最左下结点
        {
            if (T->lchild)
            {
                p = T->lchild;
                while (p->rchild)p = p->rchild;
                *a = p->data.key;
            }
            if (T->rchild) {
                p = T->rchild;
                while (p->lchild)p = p->lchild;
                *b = p->data.key;
            }
        }
}
```

第九章作业：排序

3、下列选项中，不可能是快速排序第 2 趟排序结果的是（A）。
A.{3，2，5，4，7，6，9}
B.{2，3，5，4，6，7，9}
C.{2，7，5，6，4，3，9}
D.{4，2，3，5，7，6，9}
快排：一趟完成后，会有一个中间值，左边比它小，右边比它大。

4、用希尔排序方法对一个数据序列进行排序时，若第 1 趟排序的结果为{9，1，4，13，7，8，20，23，15}，则该趟排序采用的增量（间隔）可能是（A）。
A.3
B.2
C.4
D.5
希尔排序：按间隔 D 分成若干子序列，然后利用插入排序完成子序列内排序。排序后以 D 为间隔的子序列呈现有序性。

5、已知关键字序列{5，8，12，19，28，20，15，22}是小根堆（最小堆），插入关键字 3，调整后得到的小根堆是（A）。

A.{3，5，12，8，28，20，15，22，19}
B.{3，5，12，19，20，15，22，8，28}
C.{3，12，5，8，28，20，15，22，19}
D.{3，8，12，5，20，15，22，28，19}
小顶堆： $r_k < r_{2k}$ **且** $r_k < r_{2k+1}$**，大顶堆：** $r_k > r_{2k}$ **且** $r_k < r_{2k+1}$

6、已知序列{25，13，10，12，9}是大根堆，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比较次数是（A）。
A.2
B.1
C.5
D.4
堆的操作：
插入/建初堆：自底向上比较
删除/调整：自顶向下比较

7、设有 1000 个无序的元素，希望用最快的速度挑选出其中前 10 个最大的元素，则最好选用（A）排序算法。
A.堆　**建大根堆 O(n);每次取出最大元素，并调整堆 O(logn);时间复杂度为 O(n + 10*logn)**
B.冒泡　**O(10n)**
C.快速　**快排序需要全部排好才可以挑出最小的 10 个。最好为每次递归可划分中间 O(nlogn),最坏为顺序或逆序 O(n^2)**
D.基数　**时间复杂度为 O(dn)，最小 O(3n)**

11、对一组初始关键字序列{40、50、95、20、15、70、60、45、10}进行冒泡排序，则第一趟需要进行相邻记录的比较的次数为 **8**，在整个排序过程中最多需要进行 **8** 趟排序才可以完成。
13、设有一组初始关键字序列为（24，35，12，27，18，26），则第 3 趟直接插入排序结束后的结果是 **12, 24, 27, 35, 18, 26**；第 3 趟简单选择排序结束后的结果是 **12, 18, 24, 27, 35, 26**。
14、设初始记录关键字序列为（k1,k2,…,kn），则用自顶向下筛选法思想建堆必须从第 **n/2** 个元素开始进行筛选，平均时间复杂度为 **O(n)**。

17、平均时间复杂度 O(nlogn)的排序算法有哪些？对其中不稳定的排序算法，请给出反例说明。
平均时间复杂度为 O(nlogn)的排序算法：快速排序、堆排序、归并排序
不稳定：快速排序、堆排序
快速排序的反例：49 65 38 97 76 38* 13 52 -> 13 38* 38 49 52 65 76 97
堆排序的反例：21 20 20* 12 11 8 7 -> 7 8 11 12 20* 20 21

20、试以单链表为存储结构，写一个插入排序算法。

```
1.  LinkList* Insert_Sort(LinkList* list)  //list 表示头结点
2.  {
3.      //单链表带头结点
```

```
    LinkList* cur,*pre,*p;
    cur = list->next->next;     //指向第二个结点
    list->next->next = NULL;
    while(cur){
        p = cur->next;      //保存当前结点的下一个结点的指针
        pre = list;
        //找到合适的位置
        while(pre->next && pre->next->data < cur->data){
            pre = pre->next;
        }
        //进行插入操作
        cur->next = pre->next;
        pre->next = cur;
        cur = p;
    }
    return list;
}
```