一， 选择题

1-5 CBBDD

1. Given the input order of a stack is 6, 5, 4, 3, 2, 1, ( )is not the valid output order?

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

2. If the MaxSize of a **Circular Queue** is n and there is always a space not used, front points to the previous of the front element in the queue, and rear points to the rear element in the queue. ( ) means that the Queue is Empty.

A. (rear+1) MOD n == front B. rear == front

C. rear+1 == front D. (rear-l) MOD n == front

3. In the following sorting methods, ( ) is not stable.

A. Insertion sort B. Heap C. Bubble D. Merge sort

4. In the following sorting methods, the method whose KCN(Keys Compare Number) is irrelative with the initial order of sequence is ( ).

A. Insertion sort B. Bubble sort C. Heap sort D. Selection sort

5. In the following sorting methods, ( ) need extra Θ(n) space.

A. Shell sort B. Heap sort C. Selection sort D. merge sort

1. 解题思路：根据栈先进后出的原则，一个一个答案核实，即可选出正确答案。
2. 解题思路：画图。
3. 解题思路：记住或者现场自己排一遍。
4. 解题思路：答案为 D。B 选项冒泡排序一般都会优化，一旦发现有序，则停止比较。选择排序，每次都要从剩下的元素中选一个最值出来，所以无关。
5. 解题思路：归并排序需要额外的 O(n)存储空间，其他三个排序不需要。

6-10 CACCA

6. In the following sequence, ( ) is a heap?

A. 75, 65, 30, 15, 25, 45, 20, 10 B. 75, 65, 45, 10, 30, 25, 20, 15

C. 75, 45, 65, 30, 15, 25, 20, 10 D. 75, 45, 65, 10, 25, 30, 20, 15

7. The data Structures can be divided into ( ) according to their **Physical form**

A. Array-based structures and Linked structures B. Dynamic structures, Static structures

C. Liner structures, Non-liner structures D. Simple structures, Complex structures

8. In the following data-structures, ( ) is liner structure.

A. DAG B. BST C. linked based Stack D. Heap

9. If the height of a Complete Binary Tree is n, then the number of node is at most ( ).

A. 2^n B. n C. 2^n-1 D. 2^{n-1}-1

10. When sorting the sequence {15，9，7，8，20，-1，4}, the middle result after one pass is: {9，15，7，8，20，-1，4}; Then the sort method used is ( ).

A. Insertion Sort B. Heap sort C. Quick sort D. Bubble Sort

6. 解题思路：根据堆的定义，挨个画出来，即可得到正确答案。
7. 解题思路：上课讲过，根据物理存储形式，数据结构分为基于数组的结构和基于链表的结构。
8. 解题思路：DAG 是有向无环图，BST 是二叉搜索树，Heap 是堆（完全二叉树），都不是线性结构，只有基于栈的链表才是线性结构。
9. 解题思路：在一个完全二叉树中，如果高度为 n，那么节点最多为 2^n-1，记不住这个结论就现场画一个完全二叉树，数一数即可获得正确答案。
10. 解题思路：看排序过后的中间结果，发现 7（包含 7）后面的顺序没变，只有前面两个数组的顺序变了，即可知道这是插入排序。

11-15 **CBDDB**

11. A collision resolution technique that places all records directly into the hash table is called ( ).
    A. Open hashing B. Separate chaining C. Closed hashing D. Probe function
12. A 2-3 tree is a specific variant of a ( ).
    A. Splay tree B. B-tree C. BST D. Trie
13. Pick the growth rate that corresponds to the most efficient algorithm when n = 4. ( )
    A. 5n B. 20 log n C. $2n^2$ D. 2^n
14. All operations on a stack can be implemented in constant time except ( ).
    A. Push
    B. Pop
    C. The implementor's choice of push or pop (they cannot both be implemented in constant time).
    D. None of the above.
15. Recursion is generally implemented using ( ).
    A. A sorted list B. A stack C. A queue D. none of the above

11．解题思路：如果不是直接将记录存放在哈希表里面，而是用到链表，那就是 open hashing。

12. 解题思路：2-3 树是一种特殊的 B 树。Splay tree 是伸展树

13. 解题思路：将 4 带入进去算，选最小的。Log 默认底数为 2.

14. 解题思路：Push 和 Pop 都可以在常量时间内完成。

15. 解题思路：递归是用栈实现的。

二，　名词解释题

1. Queue：队列是一种受限制的线性表，队列元素只能从队尾插入（成为入队操作），从队首删除（成为出队操作）

2. Heap：是一种完全二叉树，堆中存储的数据是局部有序的。父节点都比子节点大的是大顶堆，父节点比子节点都小的是小顶堆。
3. Double linked list：从一个表节点出发，存储两个指针，一个指针指向后继节点，一个指针指向前驱结点。
4. B+ tree：B+树是 B 树的一种特殊形式，它只在叶节点存储记录，内部节点存储关键码值，关键码值只占据位置，用于引导检索。根节点的孩子数为 2-m，m 为 B+树的阶数。

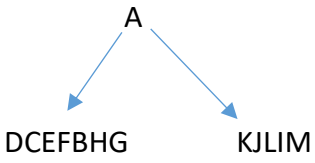
三，　应用题

1. **中序遍历:DCEFBHGAKJLIM；后序：DFECHGBKLJMIA，画出二叉树并写出前序遍历。**

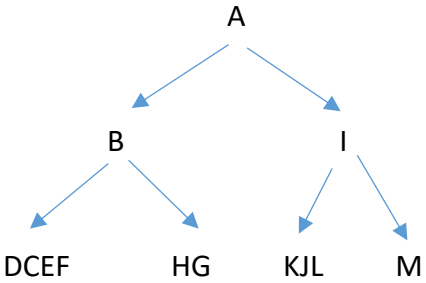
1. Suppose you have a binary tree whose data fields are single characters. When the nodes are output in in-order, the output is DCEFBHGAKJLIM, and when they are output in post-order, the output is DFECHGBKLJMIA. Draw the binary tree showing the data in each node, and show the result when the nodes are output in pre-order.

解答过程：

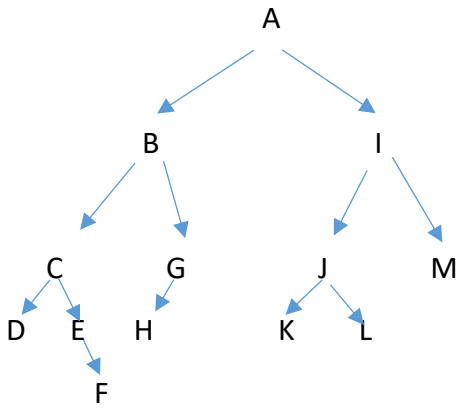
1) 根据后序最后一个字母是 A，说明 A 是根节点，然后根据中序将该树分为根节点的左右子树 DCEFBHG 和 KJLIM,



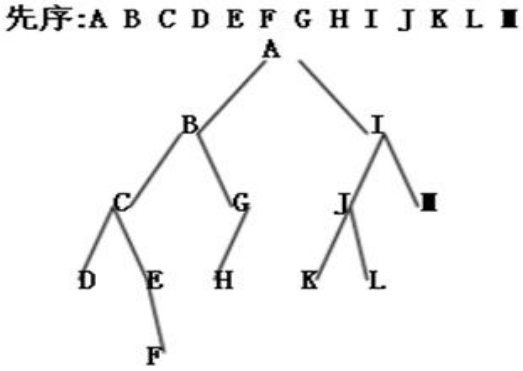
2) 根据后序 DFECHGB | KLJMIA，则 A 的左子树的根是 B，右子树的根是 I，那么



3) 根据后序 DFEC | HGB | KLJ | MIA，可知 B 的左子树的根是 C，右子树的根是 G，I 的左子树的根是 J，右子树的根是 M。并且同理可知，该二叉树的结构如下：



4) 所以，根据二叉树的结构，该二叉树的前序遍历是 ABCDEFGHIJKLM



2. 二叉查找树：23,49,28,10,30,5,16

画出二叉查找树

2. Starting from an empty binary tree, sequentially insert the following elements one by one according to the insertion algorithm of binary search tree: 23, 49, 28, 10, 30, 5, 16.
(a) Draw the binary search tree after inserting all the above elements.
(b) Beginning at the root, search for a record with value 7 in the above BST. Before the search is over, which nodes will be compared with value 7?

解答过程：

(a):

1) 插入 23：

23

2) 插入 49，49 比 23 大，作为 23 节点的右孩子插入该树：

23

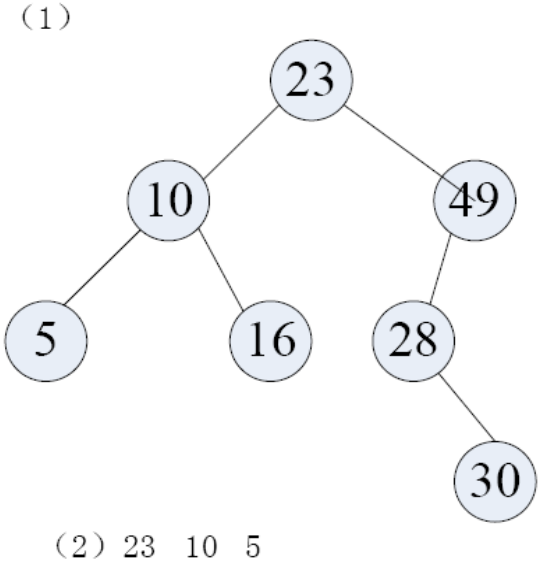
49

3) 插入 28，28 比 23 大，往 23 节点的右子树中走，28 比 49 小，所以作为 49 节点的左孩子插入该树；

23

49

4) 同理，插入 10,30,5,16，最后得到的搜索二叉树如下图所示：



(b):

查找 7：7 比 23 小，遍历 23 节点左子树，7 比 10 小，遍历 10 节点左子树，7 比 5 大，遍历 5 节点右子树，然而 5 节点又子树为空，未找到 7，遍历结束，所以该过程访问过的节点有 23,10,5.

3. 查找：Hash 表，H(key)=key MOD 13,用 open hashing 处理冲突

3. Build a hash table of 19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79, using hash function H(key) = key MOD 13. The collision resolution policy adopts open hashing, namely the collision results is stored in a certain slot's linked list. The size of hash table n = 13. Please show the process.

解答过程：

根据 hash 函数 H(key)=key MOD 13 可知：

19%13=6，放入下标为 6 的哈希表中；

14%13=1，放入下标为 1 的哈希表中；

23%13=10，放入下标为 10 的哈希表中；

1%13=1（冲突），链接到下标为 1 的哈希表的 slot's linked list 中；

68%13=3，放入下标为 3 的哈希表中；

20%13=7，放入下标为 7 的哈希表中；

84%13=6（冲突），链接到下标为 6 的哈希表的 slot's linked list 中；

27%13=1（冲突）；链接到下标为 1 的哈希表的 slot's linked list 中；

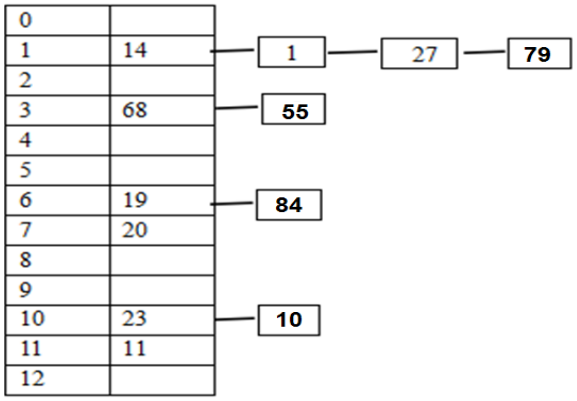
55%13=3（冲突）；链接到下标为 3 的哈希表的 slot's linked list 中

11%13=11；放入下标为 11 的哈希表中；

10%13=10（冲突），链接到下标为 10 的哈希表的 slot’s linked list 中

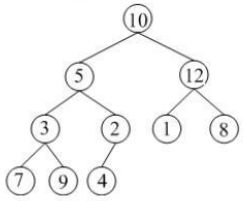
79%13=1（冲突），链接到下标为 1 的哈希表的 slot’s linked list 中.

最后的哈希表结构如下：



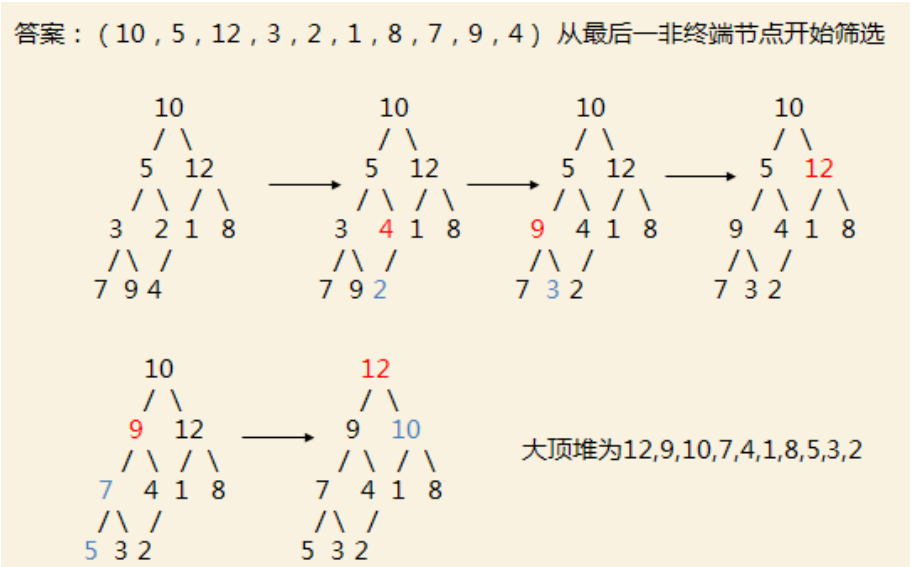
4. 堆

4. Is the following binary a max-heap? If not, please change it to a max-heap. Be sure to show the steps of building max-heap.



解答过程：

The binary is not a max-heap.



从最后一个非叶结点开始筛选：

1) 2 比 4 小，则 2 与 4 交换位置（画出交换位置过后的二叉树）；

2) 3,7,9 相比，9 最大，3 和 9 交换（画出交换位置过后的二叉树）；

3) 12,1,8 相比，12 最大，不交换（画出此刻的二叉树）；

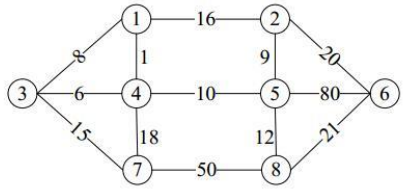
4) 5,9,4 相比，9 最大，9 和 5 交换，shifdown，5,7,3 相比，7 最大，5 与 7 交换（画出交换位置过后的二叉树）；

5) 10,9,12 相比，12 最大，12 和 10 交换，10,1,8 相比，10 最大，不交换。

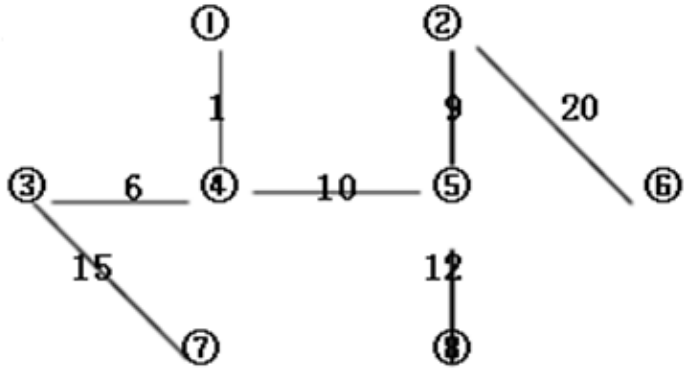
最后的大顶推为：12,9,10,7,4,1,8,5,3,2（可以再画出二叉树形式的大顶堆）

5. 图

5. List the order in which the edges of the following graph are visited when running Prim's MST algorithm starting at Vertex 1. Show the MST.



解答过程：



1) MST={1}，选取与 MST 连接并不能构成环的权值最小的边(1,4),此时 MST={1,4}；

2) MST={1,4}，选取与 MST 连接并不能构成环的权值最小的边(3,4),此时 MST={1,4,3};

3) MST={1,4,3}，选取与 MST 连接并不能构成环的权值最小的边(4,5),此时 MST={1,4,3,5};

4) MST={1,4,3,5}，选取与 MST 连接并不能构成环的权值最小的边(2,5),此时 MST={1,4,3,5,2};

5) MST={1,4,3,5,2}，选取与 MST 连接并不能构成环的权值最小的边(5,8),此时 MST={1,4,3,5,2,8};

6) MST={1,4,3,5,2,8}，选取与 MST 连接并不能构成环的权值最小的边(3,7),此时 MST={1,4,3,5,2,8,7};

7) MST={1,4,3,5,2,8,7}，选取与 MST 连接并不能构成环的权值最小的边(2,6),此时 MST={1,4,3,5,2,8,7,6};

8) 最小生成树构造完毕，加入 MST 中节点的顺序为 1, 4, 3, 5, 2, 8, 7, 6

四、编程题

1. Suppose the input data are stored in a singly linked list with head node. The following is the **selection** sorting algorithm on linked list. The definition of linked node is like following:

```
typedef struct node{
    ElemType data; // 数据域
    struct node *link; // 指针域
    }node;
node *SelectSort(node *La)
{   node *p, *q, *r, *s;
    p = La;
    while ( p -> link != null ) {
        q = p -> link;   r = p;
        while (    (1)    ) {
            if (q -> link -> data < r -> link -> data )   r = q;
            q = q -> link;
            }
        if (    (2)    ) {
            s = r -> link;   r -> link = s -> link;
            s -> link =    (3)    ;
                (4)    ;
            }
          (5)    ;
        }
    return(La);
}
```

1．(1)q->link!=NULL　(2)r!=p　(3)p->link　(4)p->link=s　(5)p=p->link

画一个例子：5,1,7,2,4

2.酌情给分

2. Given two sorted linked list L1 and L2, write a function to compute L3 = L1 ∪ L2. What is the running time of your algorithm?

```
//非递归合并两个有序的单链表
ListNode* mergeList2(ListNode *pHead1, ListNode *pHead2)
{
    //参数校验
    if(pHead1==NULL && pHead2 == NULL)
    {
        return NULL;
    }
    else if(pHead1 == NULL)
    {
        return pHead2;
    }else if(pHead2 == NULL)
    {
        return pHead1;
    }
    //合并后的头指针
    ListNode *pMergeHead;
    pMergeHead->next=NULL;
    ListNode *p1 = pMergeHead;//p1指向合并链表的最后一个节点
    while(pHead1!=NULL && pHead2!=NULL)
    {
        if(pHead1->data < pHead2->data)
        {
            p1->next = pHead1;
            pHead1=pHead1->next;
        }else{
            p1->next = pHead2;
            pHead2 = pHead2->next;
        }
        p1 = p1->next;
    }
    //把剩余的节点插入到合并的链表中
    if(pHead1!=NULL)
    {
        p1->next = pHead1;
    }
    //如果第二个链表没有插入完
    if(pHead2!=NULL)
    {
        p1->next = pHead2;
    }
    return pMergeHead;
}
```

```
struct ListNode{
    int data;//数据域
    ListNode *next;//指向下一个节点的指针域
};
//递归合并两个有序的单链表
ListNode* mergeList1(ListNode *pHead1, ListNode *pHead2)
{
    //参数校验
    if(pHead1==NULL && pHead2 == NULL)
    {
        return NULL;
    }
    else if(pHead1 == NULL)
    {
        return pHead2;
    }else if(pHead2 == NULL)
    {
        return pHead1;
    }
    ListNode *pMergeHead = NULL;
    //链表1的节点数据域数据较小
    if(pHead1->data < pHead2->data)
    {
        pMergeHead = pHead1;
        pMergeHead->next = mergeList1(pHead1->next , pHead2);
    }else{
        pMergeHead = pHead2;
        pMergeHead->next = mergeList1(pHead1, pHead2->next);
    }
    return pMergeHead;
}
```

参数校验中也可以写成：

```
if(pHead1==NULL || pHead2==NULL)
    return pHead1==NULL? pHead2:pHead1;
```

最后再循环一次，去重。