课外练习

——By Nooraovo

Chapter1 算法分析

Tips：

区别N^2和N*2

循环中是所有的数都做了吗?

i ^ 0.5 < N → i < N ^ 2,故循环的是O(N^2)

判断题

1. 将长度分别为m,n的两个单链表合并为一个单链表的时间复杂度为O(m+n)

F

把一个接在另一个后面即可，O(1)

2. $(\log N)^2$是*O*(N)的。

T

> O的定义：小于等于

1-6 The Fibonacci number sequence $\{F_N\}$ is defined as: $F_0 = 0$, $F_1 = 1$, $F_N = F_{N-1} + F_{N-2}$, N=2, 3, ... The time complexity of the function which calculates F_N iteratively is $\Theta(F_N)$. (1分)

3. ○T ◉F

注意是**iteratively循环**计算，所以复杂度为O(N)

4. For the following piece of code，the lowest upper bound of the time complexity is O(N^3).

```
if ( A > B ){
  for ( i=0; i<N*2; i++ )
    for ( j=N*N; j>i; j-- )
      C += A;
}
else {
  for ( i=0; i<N*N/100; i++ )
    for ( j=N; j>i; j-- )
```

```
    for ( k=0; k<N*3; k++)
      C += B;
}
```

T

> if中上半部分复杂度明显为N^3，不要把N*2看成N平方，else复杂度为N * N * N(**当i到N的时候，j就不进去了**，因此最外层循环按N算，答案为T)

选择题

1. 与数据元素本身的形式、内容、相对位置、个数无关的是数据的（ ）。

A.存储结构
B.存储实现
C.逻辑结构
D.运算实现

2. 斐波那契数列F_N的定义为：$F_0=0$, $F_1=1$, $F_N=F_{N-1}+F_{N-2}$, N=2, 3, …。用**递归函数**计算FN的时间复杂度是：

A.O(logN)
B.O(N)
C.O(N!)
D.O(F_N)

> 递归：T(N) = T(N −1) + T(N −2) + 2，和前面的循环区别

3. Suppose A is an array of length N with some random numbers. What is the time complexity of the following program in the worst case?

```
void function( int A[], int N ) {
    int i, j = 0, cnt = 0;
    for (i = 0; i < N; ++i) {
        for (; j < N && A[j] <= A[i]; ++j);
        cnt += j − i;
    }
}
```

A.O(N2)

B.O(NlogN)

C.O(N)

D.O(N1.5)

> 不要看到两个循环就是N^2，j在循环中**没有赋初值**，所以++j最多执行n次

> For the following function (where n>0)
>
> ```
> void func ( int n )
> {   int i, j;
> for (i=n; i>0; i/=3)
>   for (j=0; j<i; j+=2)
>       printf("%d ", i+j);
> }
> ```
>
> the most accurate time complexity bound is:
>
> A. *O*(log*n*)
>
> B. *O*(*n*)
>
> C. *O*(*n*log*n*)
>
> D. $O(\sqrt{n})$

```
$n + n/3 + n/3^2+... = n (1 + 1/3 + 1/3^2)$
```

$$n + n/3 + n/3^2 + \ldots = n(1 + 1/3 + 1/3^2)$$

> 等比数列求和

6. The Fibonacci number sequence $\{F_N\}$ is defined as: $F_0 = 0$, $F_1 = 1$, $F_N = F_{N-1} + F_{N-2}$, N=2, 3, … The **space complexity** of the function which calculates F_N recursively is:

> 每个数算两次，空间复杂度为O(N)

note：Fibonacci复杂度总结

	循环	递归
时间复杂度	O(N)	$O(F_N)$
空间复杂度	O(1)【几个变量间相互迭代】	O(N)

Chaper3–5 链表、堆栈、队列

判断题

1. 顺序栈中元素值的大小是有序的。

F

> 顺序栈指内存是连续的，并非元素有序，还有一种是链式栈

2. 对顺序栈进行进栈、出栈操作不涉及元素的前、后移动问题。

T

> 只要移动top指针即可

R1-9 "Circular Queue" is defined to be a queue implemented by a circularly linked list or a circular array.

○ T ◉ F

> 循环队列是一个抽象的概念，不局限于实现方式。

4. 对于顺序存储的长度为N的线性表，访问结点和增加结点的时间复杂度分别对应为O(1)和O(N)。

T

> “增加结点”不一定是最后一个

选择题

1. 中缀表达式转后缀 https://blog.csdn.net/weixin_44162361/article/details/116170947
   中缀表达式 转 前缀表达式https://blog.csdn.net/zm_miner/article/details/115329977

2. 令P代表入栈，O代表出栈。当利用堆栈求解后缀表达式1 2 3 + * 4 一时，堆栈操作序列是：

   A.PPPOOPOO

   B.PPOOPPOOPPOO

   C.PPPOOPOOPPOO

   D.PPPOOPOOPPOOPO

> PPP:1 2 3 入栈
>
> OO：2 3 出栈参与 + 运算

> P：2+3=5 结果入栈
>
> OO：5 1出栈参与 * 运算
>
> PP：5*1=5 和 4 入栈
>
> OO：5 4 出栈参与 – 运算
>
> P：5+4=9 结果入栈
>
> O：9 出栈

3. 判断一个循环队列QU（最多元素为MaxSize）为空的条件是（）。

A.QU.front == QU.rear
B.QU.front != QU.rear
C.QU.front == (QU.rear + 1) % MaxSize
D.QU.front != (QU.rear + 1) % MaxSize

> 注意区别A（判断为空）和C（判断为满）
>
> 一般初始化Front=Rear=0

4. To merge two singly linked ascending lists, both with N nodes, into one singly linked ascending list, the minimum possible number of comparisons is:

   A.1

   B.N

   C.2N

   D.NlogN

> 虽然把一个接到另一个后面，即一张表的尾元素小于另一张表的首元素，最小比较一次。
>
> 但是要**找到尾元素一定要遍历链表一次**，复杂度为O(N)

5. Represent a queue by a singly linked list. Given the current status of the linked list as 1–>2–>3 where x–>y means y is linked after x. Now if 4 is enqueued and then a dequeue is done, the resulting status must be:

   A.1–>2–>3

   B.2–>3–>4

   C.4–>1–>2

   D.the solution is not unique

> 用链表表示队列，出队时要记住(新的）最后一个元素，所以**前出后入**

6. 在具有N个结点的单链表中，实现下列哪个操作，其算法的时间复杂度是O(N)?
   A.删除开始结点
   B.在地址为p的结点之后插入一个结点
   C.遍历链表和求链表的第i个结点
   D.删除地址为p的结点的后继结点

> BD中“地址为p”所以可以直接找到结点位置，复杂度为O(1)

7. If the most commonly used operations are to insert a new element after the last element, and to delete the first element in a linear list, then which of the following data structures is the most efficient?
   A.singly linked list
   B.singly linked circular list with a tail pointer
   C.singly linked circular list with a head pointer
   D.doubly linked list

> 循环链表的尾指针可以很快地找到头指针（注意链表不同于数组，一定是满的，尾指针下一个就是头指针），头指针却不能很快地找到尾指针

8. 广义表是一种（）数据结构。
   A.非递归的
   B.递归的
   C.树型
   D.图状

> 广义表multilist的定义：LS = (a1,a2,…,an)，an 表示广义表存储的数据，广义表中每个 ai 既可以代表单个元素，也可以代表另一个广义表。递归的数据结构

9. Suppose that enqueue is allowed to happen at both ends of a queue, but dequeue can only be done at one end. If elements are enqueued in the order {a, b, c, d, e}, the impossible dequeue sequence is:

A.b a c d e

B.d b a c e

C.e c b a d

D.d b c a e

> 虽然不确定什么时候Dequeue，但因为输入有顺序，所以可以从a出发，向左/右有序地依次拿掉，D中a的左右两边都和a不连续，明显错了。

Chapter3 树

Tips：

二叉树是否为complete

parent 和ancestor区别

是BST还是普通二叉树

判断题

1. 二叉树是度为 2 的树。

F

> 度可能为0/1/2

2. 若A和B都是一棵二叉树的叶子结点，则存在这样的二叉树，其前序遍历序列为...A...B...，而中序遍历序列为...B...A...。

F

> 先序中序后序是对根结点而言的，叶子结点的顺序保持不变。

3. 二叉搜索树的查找和折半查找的时间复杂度相同。

F

> 二叉排序树不一定是平衡树，它是只要求了左右子树与根结点存在大小关系，但是对左右子树之间没有层次差异的约束，因此通过二叉排序树进行查找不一定能够满足logn的，例如一棵只有多层左子树的二叉排序树。

4. 二叉搜索树的最小元素一定位于树根的左子树。

F

> 还可能是根节点

5. Given a binary search tree with 20 integer keys which include 10, 11, and 12, if 10 and 12 are on the same level, then 11 must be their common ancestor.

T

> 注意是**common ancestor**，不一定是parent

选择题

1.设树T的度为4，其中度为1、2、3、4的结点个数分别为4、2、1、1。则T中有多少个叶子结点?

A.4
B.6
C.8
D.10

> 结点个数和度数满足下面的公式:
>
> $$\Sigma Degree+1=n$$

2. 如果二叉树的前序遍历结果是12345，后序遍历结果是32541，那么该二叉树的中序遍历结果是什么?

A.23145
B.23154
C.24135
D.无法确定

> 能确定2和4一定是1的子节点，但无法确定3和5，没说是BST，先序和后序不能确定树。

3. .某二叉树的先序序列和后序序列正好相反，则下列说法错误的是（）。

A.二叉树不存在
B.若二叉树不为空，则二叉树的深度等于结点数
C.若二叉树不为空，则任一结点不能同时拥有左孩子和 右孩子
D.若二叉树不为空，则任一结点的度均为1

> BCD其实含义相同

4. 一棵树可转换成为与其对应的二叉树，则下面叙述正确的是（）。

A.树的先根遍历序列与其对应的二叉树的先序 遍历相同
B.树的后根遍历序列与其对应的二叉树的后序遍历相同
C.树的先根遍历序列与其对应的二叉树的中序遍历相同
D.以上都不对

> 先根遍历序列——对每个子树先遍历根，但仍然是从左至右的顺序
>
> 而T的**后序**遍历=BT的**中序**遍历

5. 若二叉搜索树是有*N*个结点的完全二叉树，则不正确的说法是：

A.所有结点的平均查找效率是O(logN)
B.最小值一定在叶结点上
C.最大值一定在叶结点上
D.中位值结点在根结点或根的左子树上

> 完全二叉树中位置在最右端的不一定是叶节点

6. 前序遍历的时间空间复杂度

R2-9 The time complexity of recursively traversing a binary tree with N nodes and height H in preorder is (3分)
_.

- (●) A. $O(N)$
- ( ) B. $O(H)$
- ( ) C. $O(N \log H)$
- ( ) D. $O(H \log N)$

R2-11 The space complexity of recursively traversing a binary tree with N nodes and height H in preorder (3分)
is _.

- ( ) A. $O(\log N)$
- (●) B. $O(H)$
- ( ) C. $O(N)$
- ( ) D. $O(H+N)$

2-23 In-order traversal of a binary tree can be done iteratively. Given the stack operation sequence as the following:

```
push(1), push(2), push(3), pop(), push(4), pop(), pop(), push(5), pop(), pop(), push(6), pop()
```

Which one of the following statements is TRUE? (3分)

- ( ) A. 6 is the root
- (●) B. 3 and 5 are siblings
- ( ) C. 2 is the parent of 4
- ( ) D. None of the above

> 方法：模拟栈，写出出栈序列–>中序遍历序列
>
> 而入栈顺序提示节点信息，父节点先入栈，没有左节点时pop

8. 在下述结论中，正确的是：
   ① 只有2个结点的树的度为1；

   ② 二叉树的度为2；

   ③ 二叉树的左右子树可任意交换；

   ④ 在最大堆（大顶堆）中，从根到任意其它结点的路径上的键值一定是按非递增有序排列的。

   A.①④

   B.②④

   C.①②③

   D.②③④

> 区别树的度和图的度，二叉树的度小于等于2（二叉树的定义要求二叉树中任意结点的度数小于等于2）
>
> 堆就不允许交换

Chapter 5 Heap

判断题

R1-3 The best "worst-case time complexity" for any algorithm that sorts by comparisons only must be $O(NlogN)$. (2分)

◉ T ○ F

> 最好的方法是堆排序

R1-6 For binary heaps with N elements, the *BuildHeap* function (which adjust an array of elements into a heap in linear time) does at most $N - log(N+1)$ comparisons between elements. (2分)

◉ T ○ F

评测结果 答案错误（0 分）

F

> 建堆比较次数为1~N/2节点的高度之和，为2N–2logN
>
> 分析：

（1）建堆过程：
每次最多对比关键字两次：左右孩子比较，根和大的那个比较
如果只有左孩子，则只需要一次关键字的对比
如果树高为h，当前结点在第i层，则一共需要下坠h-i层，对比关键字不超过2(h-i)
而h=$\lfloor log_2 n \rfloor$+1
第i层最多有2^{i-1}个结点，只有第1~h-1层才可能出现元素下坠
因此第一层有2^{1-1}=1个结点，下坠共需要1×2(h-1)次关键字对比；第二层共有2^{2-1}=2个结点，下坠共需要2×2(h-2)次关键字对比；第h-1层共有2^{h-2}，下坠共需要2^{h-2}×2次关键字对比，带入h，最终结果≤4n

故S=2（h−1+2(h−2)+2^2(h−3)+.....+2^{h-2})=2^{h+1}−2h=2N−2logN

对比：

R1-8 For binary heaps with N elements, the *BuildHeap* function (which adjust an array of elements into a heap in linear time) does at most $2N-2$ comparisons between elements. (2分)

○ T ◉ F

1-8 答案错误 ⓘ (0 分) 创建提问

这道题是对的，注意是“at most”，当N=2时为2N−2

R1-4 Given a max-heap with unique keys, there are at least `d-1` keys larger than any key in level `d` (the root level is `1`). (2分)

◉ T ○ F

读懂题意，至少有d−1个比d层所有元素都大的节点值

填空题

The function is to turn an array A[] of N elements into a max-heap. s

```
#define leftchild(i) ( 2*(i)+1 )
void BuildMaxHeap( ElementType A[], int N )
{ int i, j, child;
 ElementType Tmp;
 for ( i = (N-1)/2; i >= 0; i-- ) {
    j = i;
    for ( Tmp = A[j]; leftchild(j) < N; j = child ) {
    child = leftchild(j);
    if ___①___ (2分))
    child ++;
    if ___②___ (2分)) A[j] = A[child];
    else break;
}
    ___③___(2分);
}
}
```

注意①处除了判断child和child+1外，还要先判断child是否为数组最后一个元素

②要用Tmp和A[child]比

答案：

① child+1<N && A[child]<A[child+1]

②Tmp<A[child]

③A[j] = Tmp

Chapter 8 Disjoint set

判断题

1. In Union/Find algorithm, if Unions are done by size, the depth of any node must be no more than N/2, but not O(logN).

F

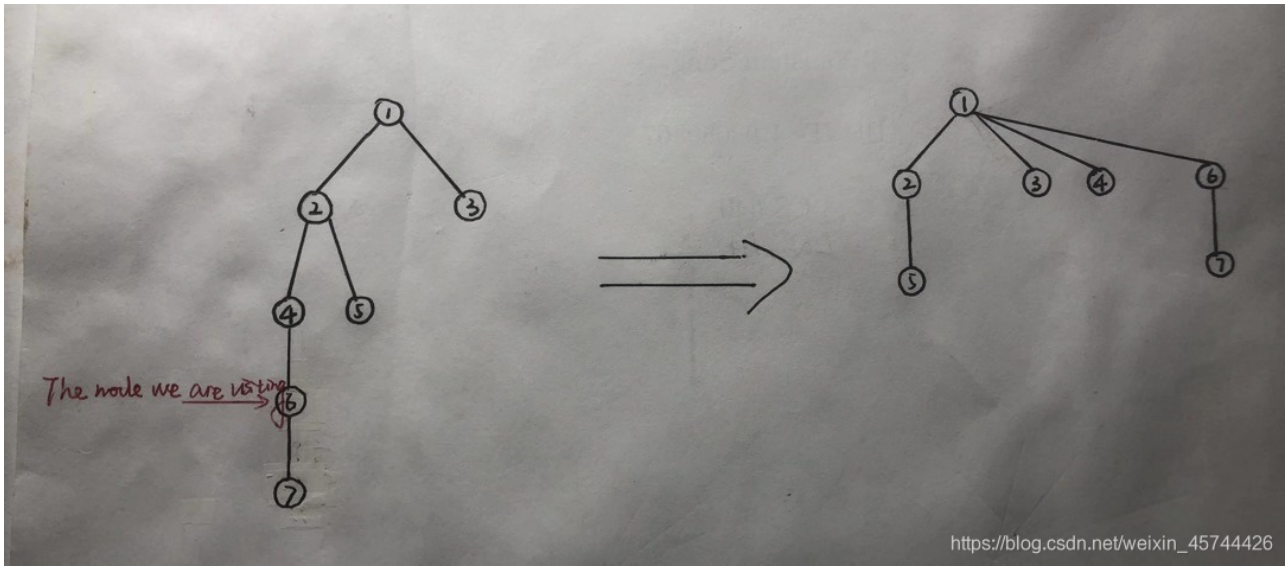
> 每做一次归并，都会使得小的集合深度加1，但是总的深度还是看大的集合。只有深度相同的归并才能使得总的深度加1，2,2归并，深度变为3；3,3归并深度变为4。因此**深度最大为**$log_2 N + 1$

选择题

R2-1 The array representation of the disjoint sets is given by S[] with S[i] being initialized to be -1 for all i. Please list the resulting array elements after invoking: union(1,2), union(3,4), union(3,5), union(1,7), union(3,6), union(8,9), union(1,8), union(1,3), and union(9,10). Assume that union-by-size and find-with-path-compression, and the elements are numbered from 1 to 10. (3分)

- A. {-3, 1, -4, 3, 3, 3, 1, -2, 8, –1}
- B. {-9, 1, 1, 3, 3, 3, 1, 1, 1, 1}
- C. {-9, 1, 1, 3, 3, 3, 1, 1, 8, –1}
- D. {-10, 1, 1, 3, 3, 3, 1, 1, 1, 1}

> 注意 path compression是在Find的过程中路径上的点依次连到最大的root上去
>
> 如下图



2. Given S = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 } and 8 equivalence relations: 5~6，7~8，9~10，2~6，3~8，6~8，1~8，1~5. After invoking successively these relations with union-by-size（if the two sizes are equal, the smaller element will be the root）and path compression, which one of the following statements is false? (3分)

   A.there are 4 equivalence classes

   B.element 1 is a child of element 5

   C.all elements are either a root or a child of a root

   D.element 2 and element 8 are siblings

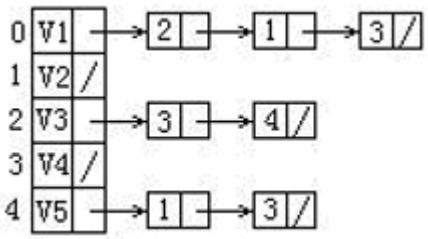
> 注意单个的元素（4,11）

Chapter 9 Graph

Tips:

有向图or无向图

用邻接表表示的时候区分下标和节点编号 eg:



connected≠complete

判断题

1. 如果无向图G必须进行两次广度优先搜索才能访问其所有顶点，则G一定有2个连通分量。
T

> 因为不论是bfs还是dfs我们在遍历的时候都进行了标记也就是当一个节点被标记了的时候这个节点就不会重复访问。因此两次bfs才访问完所有的节点不是因为有回路而是因为这个图有两个连通分量。
>
> 另外DFS可判断是否有loop

2. If e is the only shortest edge in the weighted graph G, then e must be in the minimum spanning tree of G
   T
3. If the BFS sequence of a graph is 1 2 3 4 ..., and if there is an edge between vertices 1 and 4, then there must be an edge between the vertices 1 and 3.
   T
4. In a directed graph G with at least two vertices, if DFS from any vertex can visit every other vertices, then the topological order must NOT exist.
   T

> 任一个点能联通其他所有点，则一定存在环

5. Suppose that a graph is represented by an adjacency matrix. If there exist non-zero entries in the matrix, yet all the entries below the diagonal are zeros, then this graph must be a directed graph
   T

> 一定是非对称矩阵，有向图
>
> 矩阵中的元素可以叫elements, items,或者entries

6. If a graph has a topological sequence, then its adjacency matrix must be triangular
   F

> 注意区分有向图和无向图，无向图是对称的，不一定是三角矩阵

7. If the adjacency matrix is triangular, then the corresponding directed graph must have a unique topological sequence.
   F
8. In a DAG, if for any pair of distinct vertices Vi and Vj, there is a path either from Vi to Vj or from Vj to Vi, then the DAG must have a unique topological sequence
   T

> 每两点间相对位置确定，则DAG唯一

9. Prim's algorithm is to grow the minimum spanning tree by adding one edge, and thus an associated vertex, to the tree in each stage.
   T

11. Kruskal's algorithm is to maintain a forest and to merge two trees into one at each stage.

T

> 注意上面两个算法的描述
>
> 另外Prim算法适合稠密图，Kruskal算法适合稀疏图

12. 在AOE-网工程中，减少任一关键活动上的权值后，整个工期也就会相应的减小。
    F

> 关键路径有多条时不一定。

13. 在关键路径上的活动都是关键活动，而关键活动也必在关键路径上。
    T

14. 若图G为连通图且不存在拓扑排序序列，则图G必有环。
    T

15. 在有n个顶点的有向图中，若要使任意两点间可以互相到达，则至少需要 n 条弧。
    T

> 注意是有向图，形成一个圆圈
>
> 如果是无向图，则为最小生成树，n-1

16. 用一维数组G[]存储有4个顶点的无向图如下：G[] = { 0, 1, 0, 1, 1, 0, 0, 0, 1, 0 }则顶点2和顶点0之间是有边的。
    T

> 发现边的集合中有10个元素，说明是邻接矩阵的一半（无向图），同时对角线上元素必为0，故是下三角矩阵

0			
1	0		
1	1	0	
0	0	1	0

选择题

1. 图的广度优先遍历类似于二叉树的：

A.先序遍历
B.中序遍历
C.后序遍历
D.层次遍历

> DFS类似先序遍历

2. 具有N（N>0）个顶点的无向图至多有多少个连通分量?

A.0
B.1
C.N−1
D.N

> 没有边时每个节点都是一个连通分量

3. .一个有*N*个顶点的强连通图至少有多少条边?

A.N−1
B.N
C.N+1
D.N(N−1)

> 定义：图中任意一对顶点，既有从vi到vj的**路径（不一定是边）**，也有从vj到vi的路径，则称该有向图是强连通图。
>
> 最少的情况是所有点围成一个圈。

4. 若无向图G =（V，E）中含10个顶点，要保证图G在任何情况下都是连通的，则需要的边数最少是：

A.45
B.37
C.36
D.9

> 任何情况下连通是指，边任意变动，都能保证G可以连通。
>
> 解法：先让n−1个点构成完全子图，然后把第n个顶点和这个子图相连，总共需要(n−1)(n−2)/2+1。

5. 12.具有 100 个顶点和 12 条边的无向图至多有多少个连通分量?

A.87
B.88
C.94
D.95

> 12条边，最多有可以有6个顶点，其余94个顶点各成一个连通分量。

6. 下列说法中正确的是

A.若一个无向图的最小生成树唯一，则该无向图中没有权值相同的边

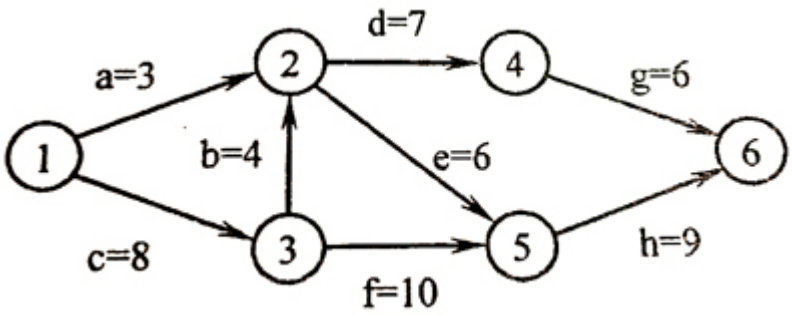
B.若一个无向完全图有 N 个顶点，且各边权值均相同，则该图有 N! 种最小生成树

C.若一个无向连通图没有权值相同的边，则该无向图的最小生成树唯一

D.一个无向图的最小生成树是该图的极大连通子图

> A&C 如果一个无向图的极小连通子图恰好是构成这个图的所有边，那么因为边没有任何选择，所以无向图中的边无所谓权值是否一样；
>
> B.拿N等于3来举例，当各边权值相同，有3种最小生成树，显然不符合N!这个规律；
>
> D.最小生成树是包含所有顶点的**极小**连通子图。

7. 下图所示的 AOE 网表示一项包含 8 个活动的工程。活动 d 的最早开始时间和最迟开始时间分别是：



A.3 和 7

B.12 和 12

C.12 和 14

D.15 和 15

> 边活动的最早开始时间：起点的最早开始时间
>
> 边活动的最迟开始时间：终点的最晚开始时间−边的时间
>
> 参考：https://blog.csdn.net/been123456789jimmy/article/details/106515471

8. 在拓扑排序算法中用堆栈和用队列产生的结果会不同吗?

A.是的肯定不同

B.肯定是相同的

C.有可能会不同

D.以上全不对

> 与拓扑排序是否唯一有关

9. 下列关于无向连通图特性的叙述中，正确的是

I. 所有顶点的度之和为偶数 II.边数大于等于顶点个数减1 III.至少有一个顶点的度为1 。

A.只有I

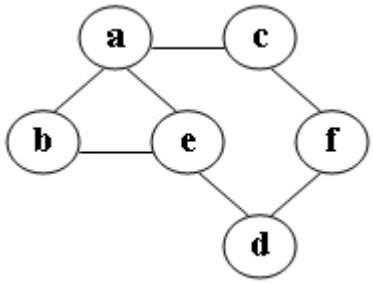
B.只有II

C.I和II

D.I和III

> II.可取等 eg：V=3，E=2
>
> III.的反例是三角形

10. 在图中自c点开始进行广度优先遍历算法可能得到的结果为：



A.c,a,b,e,f,d

B.c,a,f,d,e,b

C.c,f,a,d,e,b

D.c,f,a,b,d,e

> 这道题注意层序遍历的顺序，因为是用队列来实现的，所以若先遍历f再遍历a，下一层要先遍历f相连的节点d，再遍历a相连的节点be

Chapter 6 Sorting

判断题

1. 仅基于比较的算法能得到的最好的“最坏时间复杂度”是O(NlogN)。

T

> 注意不一定是简单排序，
>
> 仅基于比较的算法能得到的最好的“最好时间复杂度”是O(N)，仅基于比较的算法能得到的最好的“最坏时间复杂度”是O(NlogN)。

2. 内排序要求数据一定要以顺序方式存储。

F

> 内排序是完全在内存中存放待排序元素的排序，存储形式不一定是顺序的。

3. 在初始数据表已经有序时，快速排序算法的时间复杂度为O(nlogn)

F

> 快速排序的时间复杂度和是否有序无关。

4. To sort N distinct records by bubble sort, the number of record swaps must reach its maximum when the original sequence is almost in sorted order.

F

> 冒泡排序的record swaps取决于**逆序对的数量**，对于一个已经排序过的序列，逆序对的数量可以是**0或者最大**，逆序对的数量可能最大也可能最小。

选择题

1. 有组记录的排序码为{ 46，79，56，38，40，84 }，则利用堆排序的方法建立的初始堆为：

A.79，46，56，38，40，80
B.84，79，56，46，40，38
C.84，56，79，40，46，38
D.84，79，56，38，40，46

> 注意不是用线性的方法建堆，本身数组就形成了一个堆，只能从最后一个非叶子元素开始调整

2. 在快速排序的一趟划分过程中，当遇到与基准数相等的元素时，如果左右指针都会停止移动，那么当所有元素都相等时，算法的时间复杂度是多少?

A.O(logN)
B.O(N)
C.O(NlogN)
D.O(N2)

>

> 指针停止就会不断交换左右指针的元素，这样虽然多余的交换次数变多，但让子序列的元素相对平均，所以一共有logN次划分，每次时间复杂度是O(N)。

3. 在快速排序的一趟划分过程中，当遇到与基准数相等的元素时，如果左右指针都不停止移动，那么当所有元素都相等时，算法的时间复杂度是多少?

A.O(logN)
B.O(N)
C.O(NlogN)
D.$O(N^2)$

> 指针不停止，导致所有元素都被放到一个划分中去，一共N个元素，所以一共有N次划分，第i次递归时移动N−i次，故总共为$O(N^2)$
>
> 同样如果只停止一个指针，也是$O(N^2)$

4. 对大部分元素已有序的数组进行排序时，直接插入排序比简单选择排序效率更高，其原因是：

   (I). 直接插入排序过程中元素之间的比较次数更少

   (II). 直接插入排序过程中所需要的辅助空间更少

   (III). 直接插入排序过程中元素的移动次数更少

   A.仅 I

   B.仅 III

   C.仅 I、II

   D.I、II 和 III

> 考虑最好的情况，全部有序
>
> (I).最好情况下，比较1次；
>
> (II). 直接插入法只需要一个记录的辅助空间r[0]，所以空间复杂度为O(1)；
>
> (III).最好情况下，不移动；
>
> 课本P247：简单选择排序：
>
> (I).无论什么情况下，比较次数都为 n(n−1)/2；
>
> (II). 只有在两个记录交换时需要一个辅助空间，所以空间复杂度为O(1)；
>
> (III).最好情况，不移动；
>
> 所以 (I). 对；(II). 一样，错；(III).一样，错；

填空题

1. 下列代码的功能是将一列元素{ r[1] … r[n] }按其键值 key 的非递减顺序排序。普通选择排序是每次仅将一个待排序列的最小元放到正确的位置上，而这个另类的选择排序是每次从待排序列中同时找到最小元和最大元，把它们放到最终的正确位置上。

```
void sort( list r[], int n )
{
    int i, j, mini, maxi;

    for (i=1; i<n-i+1; i++) {
        mini = maxi = i;
        for( j=i+1;  ①       (3分); ++j ){
            if( ②        (3分) ) mini = j;
            else if(r[j]->key > r[maxi]->key) maxi = j;
        }
        if( ③          (3分) ) swap(&r[mini], &r[i]);
        if( maxi != n-i+1 ){
            if( ④            (3分) ) swap(&r[mini], &r[n-i+1]);
            else swap(&r[maxi], &r[n-i+1]);
        }
    }
}
```

需要注意的是第④空（i == maxi），因为先进行最小值的交换，此时若a[i]为最大值，则i的位置改变，即a[maxi]中不是最大值了，所以要将r[mini]（被换为a[i])和a[n-i+1]交换

2. 本函数的功能是从有N个元素的线性表A中查找第K小的元素。函数的初始调用为Qselect(A, K, 0, N-1)。请完成下列填空。

```
ElementType Qselect( ElementType A[], int K, int Left, int Right )
{
    ElementType Pivot = A[Left];
    int L = Left, R = Right+1;

    while (1) {
        while ( A[++L] < Pivot ) ;
                while (   ①   )(3分);
        if ( L < R ) Swap( &A[L], &A[R] );
        else break;
    }
    Swap( &A[Left], &A[R] );
    if ( K < (L-Left) )
        return Qselect(A, K, Left, R-1);
    else if ( K > (L-Left) )
                return Qselect(A,   ②   , R+1, Right);(3分);
    else
        return Pivot;
}
```

> 注意第二空，一方面需要用R+1~Right表示右区间，和上面对称【hint】；另一方面注意在右区间需要减去（左区间元素个数）不用+1！带几个数试一试！
>
> 答案：

> ①A[--R] > Pivot
>
> ② K-L+Left

Chapter 7 Hashing

判断题

1. 在散列中，函数“插入”和“查找”具有同样的时间复杂度

T

> 插入和查找具有同样的时间复杂度O(1)，默认是理想的散列。

选择题

1. The average search time of searching a hash table with N elements is:

A.O(1)

B.O(logN)

C.O(N)

D.cannot be determined

> 不确定是否发生冲突

2. 一个哈希函数被认为是“好的”，如果它满足条件（）。

   A.哈希地址分布均匀
   B.所有哈希地址在表长范围内
   C.保证不产生冲突
   D.满足(B)和(C)

> 其实分布均匀则可以尽量减少冲突，，不一定要不产生冲突

3. 哈希表的平均查找长度是（）的函数。

A.哈希表的长度
B.哈希表的装填因子
C.哈希函数
D.表中元素的多少

在一般情况下，处理冲突方法相同的哈希表，其平均查找长度依赖于哈希表的**装填因子**。

哈希表的装填因子定义为

$$\alpha = \frac{n}{m}$$

n为表中填入的记录数，m为哈希表的长度
采用链地址法时，α也是每个链表的平均长度。

设平均查找长度为S_n，则

$$S_n \approx -\frac{1}{\alpha}\ln(1-\alpha)$$

因此，哈希表的平均查找长度是α的函数，而不是n的函数。不管n多大，我们总可以选择一个合适的装填因子以便将平均查找长度限定在一个范围内。

如果哈希表的长度与元素的个数成比例，则n = o(m)，那么

$$\alpha = \frac{n}{m} = \frac{o(m)}{m} = o(1)$$

因此，查找都是常量时间。

4. 给定散列表大小为11，散列函数为H(Key)=Key%11。按照线性探测冲突解决策略连续插入散列值相同的4个元素。问：此时该散列表的平均不成功查找次数是多少?

A.1

B.4/11

C.21/11

D.不确定

> 先根据哈希函数进行分类，余数为[0,10]，假设四个元素都是0，找到空元素时停止，则分别查，5,4,3,2,1,1,1,1,1,1,1次，求和/N可得

5. 现有长度为 11 且初始为空的散列表 HT，散列函数是 H(key)=key%7，采用线性探查（线性探测再散列）法解决冲突。将关键字序列 87,40,30,6,11,22,98,20 依次插入到 HT 后，HT 查找失败的平均查找长度是：

A.4
B.5.25
C.6
D.6.29

> 注意总长度为11，但因为%7，在计算查找失败的平均查找长度的时候，是不需要把下标6以后的元素计算进去的，因为散列函数根本映射不到位置7

7. 设数字 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 在大小为10的散列表中根据散列函数 h(X)=X%10得到的下标对应为 {1, 3, 4, 9, 5, 0, 2}。那么继续用散列函数 “h(X)=X%表长”实施再散列并用线性探测法解决冲突后，它们的下标变为：

   A.11, 3, 13, 19, 4, 0, 9
   B.1, 3, 4, 9, 5, 0, 2
   C.1, 12, 9, 13, 20, 19, 11
   D.1, 12, 17, 0, 13, 8, 14

> 注意再散列把表长增加两倍之后，还需要**取质数**，所以是最终表长是 23。