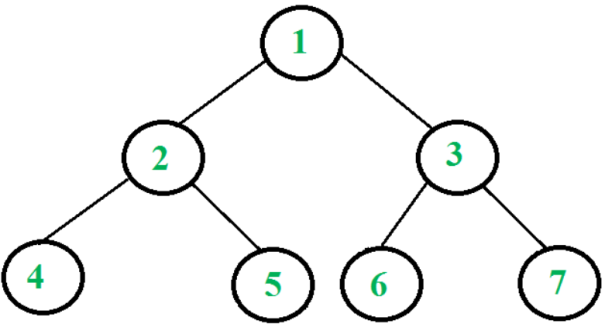
Mid-term Examination of Data Structure and Algorithm Analysis 2021

(Duration: 90 minutes Mark: 100)

Student Number:Name: **Score:**

1. **Please fill the blanks: ()**
2. Given an unsorted array with *n* records, the time complexity for reading the *i*-th record is ( Ɵ(1) ) (**3 points**)
3. Given an unsorted array with *n* records, the average time complexity for finding the record with a given key *K* by the sequential search is ( Ɵ(n) ) (**3 points**)
4. Given a complete binary tree with height *h* (*h*>0), there are ( *2i* ) nodes at the *i*-th level with *0 ≤i≤h-1*, and there are at least ( 1 ) nodes， at most ( *2h* ) nodes at the bottom level (i.e., the *h*-th level). (**Each blank is 3 points**)
5. Given a binary tree in Fig. 1, the inorder traversal sequence is (4,2,5,1,6,3,7), the postorder traversal sequence is ( 4,5,2,6,7,3,1 ), and the preorder traversal sequence is ( 1,2,4,5, 3,6,7 ). (**Each blank is 3 points**)



**Fig. 1**

1. The preorder traversal of a binary tree is A,B,C,D,E, and the inorder traversal is B,C,A,E,D, please draw the binary tree. (**9 points**)

***Requirement***: first briefly describe the basic idea about how you figure out the structure of the tree, and then draw the tree.

***Answer***: The preorder traversal first visits a node, then its left sub-tree, followed by its right subtree. On the other hand, the inorder traversal visits its left sub-tree, the node itself, followed by its right subtree. (**1 point**)

From the preorder traversal sequence A,B, C, D, E, we first know that node A is the root of the tree. From the inorder traversal sequence B,C, A, E, D, it can be seen that both nodes B and C are in the left subtree of node A, while both nodes E and D are in the right subtree of node A. (**2 points**)

What the structure of the left subtree rooted at A with two nodes B and C? Notice that both the preorder and inorder traversals in the left subtree are B, C. Then, B is the root of the left subtree, and node C is in the right subtree of the subtree rooted at B. (**2 points**)

Similarly, we can figure out the structure of the right subtree rooted at A with nodes D and E. Notice that the preorder in the right subtree is D, E, while its inorder traversal is E,D. Then, node D is the root of the right subtree, and node E is in the left subtree of the subtree rooted at D. (**2 points**)

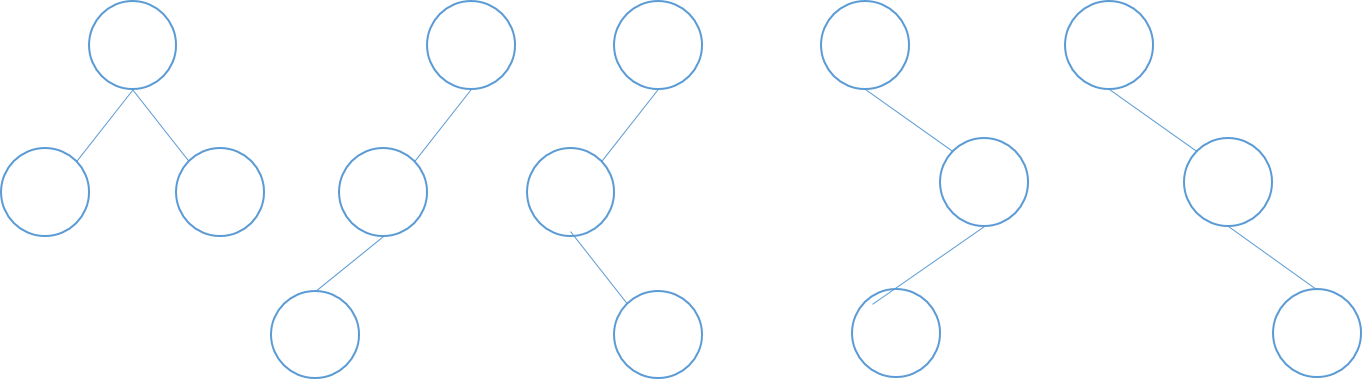
The tree structure then is: (**2 points**)

1. Draw all binary trees with 3 nodes. (**10 points**)

***Requirement***: first briefly describe the principle of enumerating all trees with 3 nodes, and then draw the tree.

***Answer***: Each binary tree has a root node **(1 point)**. We consider three cases: （1）Each of the two subtrees of root has exactly 1 node; （2）The left subtree has 2 nodes, while the right subtree has 0 node; (3) The left subtree has 0 nodes, while the right subtree has 2 node. **(2 points)**

For case (1), the structure is shown in the following first figure **(2 points)**. For case (2), we consider the sub-problem of enumerating all trees with 2 nodes with the similar discussion with 3 nodes. There are two different tree structures. See the following second and third figures **(3 points)**. The tree structures of case (3) are shown in the fourth and fifth figures. **(2 points)**



1. Draw the binary search tree (BST) by inserting 2, 1, 4, 5, 9, 3, 6, 7 (**5 points**). After the insertions, draw the results for deleting 4 and 9, respectively (**8 points**).

Answer: The tree after all insertions is as follows.

Fig. The tree after all insertion

The process of deleting 4 is as follows. Node 4 has two subtrees. First find the smallest node in its right subtree, i.e., node 5, and swap 5 and 4, see the following figure **(2 points)**. Then, remove node 4 in the swapped tree, by letting its parent node 5 point its right child 9. The tree after all deleting node 4 is as follows **(2 points).**

Fig. The tree after swapping 5 and 4

Fig. The tree after deleting 4

The process of deleting 9 is as follows. Node 9 has only one subtree. Remove node 9 by letting its parent node 5 point to its left child 6. The tree after all deleting node 9 is as follows **(4 points).**

Fig. The tree after deleting 9

1. The factorial function is defined as F(n) = n\*F(n-1) for n>0 and F(0)=1. Consider the following recursive algorithm for calculating F(n).

int Fact ( int n ) {

int m;  
 if (n= =0) return 1;  
 else{

m=n\*Fact(n-1);  
 return(m);

}  
 }

Please convert this recursive algorithm to an iterative algorithm and analyze the time complexity of your algorithm. What is the maximum integer *n* your algorithm can calculate for *Fact(n)* without overflow, assuming that an *int* variable consists of 32 bits ? (**18 points**).

***Requirement***: first briefly describe the basic idea behind your algorithm, and then write down your algorithm.

***Answer***: For calculating F(n), we calculate F(0), F(1), F(2),…,F(n-1), F(n) one by one **(4 points)**. The algorithm is as follows.

***int FactIterative( int n){***

***int m=1;* (1 points). *for(int i=1; i<=n; ++i)* (2 points).**

***m \*= i;* (2 points)**

***return m;* (1 points)**

***}***

The time complexity of the algorithm is Ɵ(n). **(2 points).**

The maximum value stored an int variable is 231-1= 2,147,483,647 **(3 points).**. On the other hand, the maximum number n such that F(n) is no greater than 231-1 is ***n=12***, where F(12)= 479,001,600. **(3 points).**

1. Given two lists ，，please write a function to merge A and B into list C by the following rules,

**  ；**

** **

Lists A , B, and C are singly linked lists, and each list has a head node. The number of lists in A and B are not given in advance. (**25 points**)

***Requirement***: first briefly describe the basic idea behind your algorithm, and then write down your algorithm.

***Answer***: We scan the two lists from their beginnings to their ends iteratively. Within each iteration, we scan one node in list A and one node in list B, followed by moving them to list C in an interwoven way, if both lists A and B have at least one node **(4 points)**. However, if one of the two lists is exhausted, we simply link the rest nodes of the other list to the tail of list C. **(4 points).**

Node<int>\* interwovenMerge(Node<int> \*headA, Node<int> \*headB){ **(1 point)**

Node<int>\* curA= headA->next; // point to the first node in list A **(1 point)**

Node<int>\* curB= headB->next; // point to the first node in list B **(1 point)**

Node<int> \*headC=headA; // the head node of list C **(1 point)**

headC->next = NULL; // an initially empty list

delete headB; // does not need it any more **(1 point)**

Node<int> \*tail = headC; // the tail of list C **(1 point)**

while( curA != NULL || curB != NULL ){ // one list has nodes **(2 points)**

if( curA != NULL && curB != NULL ){ // both are not empty **(1 point)**

**// sequence: link the first node in list A, then curA moves forward,**

**// finally, link the first node in list B and curB moves forward**

tail->next = curA; // move the first node from list A to list C **(1 point)**

tail = tail->next; // point to the newly added node

curA= curA->next; // move forward **(1 point)**

tail->next = curB; // move the first node from list B to list C **(1 point)**

tail = tail->next; // point to the newly added node

curB= curB->next; // move forward

}else if( curA != NULL){ //list A is not empty, and list B is empty **(1 point)**

tail->next = curA; // link the rest nodes in list A **(1 point)**

break; // the work is done **(1 point)**

}else{ //list A is empty, and list B maybe or not empty **(1 point)**

tail->next = curB; // link the rest nodes in list A **(1 point)**

break; // the work is done

}

}

}