**CS235101 Data Structure Midterm Exam**

1. **(10%) Performance Analysis**
2. **(3%)** Please define three asymptotic notations, Θ, Ο and Ω.
3. **(2%)** Prove that if

.

1. **(5%)** Please determine the big-Ο complexity of following functions:
2. **(15%) Expression Evaluation**
3. **(6%)**Please evaluate the value of the following postfix and prefix expression
4. Postfix :

3 6 – 2 5 \* +

1. Prefix :

* + – \* 2 3 5 / \* 4 3 6 5

1. **(9%)**Given the infix expression

A – ( C **+** D )\* E – F \* C

Please show the sequence of stacking operations of infix to postfix

(Please refer the following example of converting infix to postfix)

|  |  |  |  |
| --- | --- | --- | --- |
| Infix Expr : A + B | | | |
|  | Next Token | Stack | Output String |
| 1 | A | EMPTY | EMPTY |
| 2 | + | EMPTY | A |
| 3 | B | + | A |
| 4 | END OF STRING | + | AB |
| 5 |  | EMPTY | AB+ |

1. **(15%) Circular Lists**
2. **(10%) Linked Stack and Queue:**

Given the follow class definition of linked stack and queue, please write down the implementation of two member functions:   
void LinkedStack::pop(void) and void LinkedQueque::push(int data).

|  |  |
| --- | --- |
| class LinkedStack{  LinkedStack(void);  void pop(void);  int capacity;  ChainNode \*top;  } | class LinkedQueque{  LinkedQueque(void);    void push(int data);    ChainNode \*front;  ChainNode \*rear;  } |

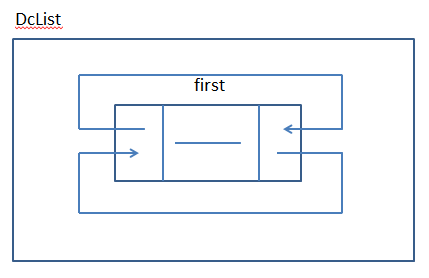
|  |
| --- |
| class ChainNode{  ChainNode( int data, ChainNode\* link);  int data;  ChainNode\* link;  } |

1. **(5%) Double Circular List**

Given the class definition of DcList and DclistNode, please write down the implementation of void DcList::insert(DclistNode \*p, DclistNode \*x).

(Note that initially DcList ->first points to a dummy node as shown in the following figure).

|  |  |
| --- | --- |
| class DcList {  DcList (void);  //insert node p to the right of node x  void insert(DclistNode \*p, DclistNode \*x);    DclistNode \*first;  } | |
|  |  |
|  |  |
| class DclistNode{  friend class DcList;  DclistNode (int, DclistNode\*, DclistNode\*);    int data;  DclistNode \*left;  DclistNode \*right;  } | |



1. **(20%) Tree**

Given the above two trees, please answer the following questions:

1. **(5%)** Please convert T1 to a binary tree using left child-right sibling representation.
2. **(5%)** Degree of T1? Depth of T1? Ancestors of node J? Siblings of node B?
3. **(5%)** Suppose we use array representation for T2, then for any node with index i, 1, please define the following terms using i (considering cases of root or having no children):

Parent(i) = ? leftChild(i) = ? and rightChild(i) = ?

1. **(5%)** Please give the output of inorder, preorder, postorder and level order traversal of T2.

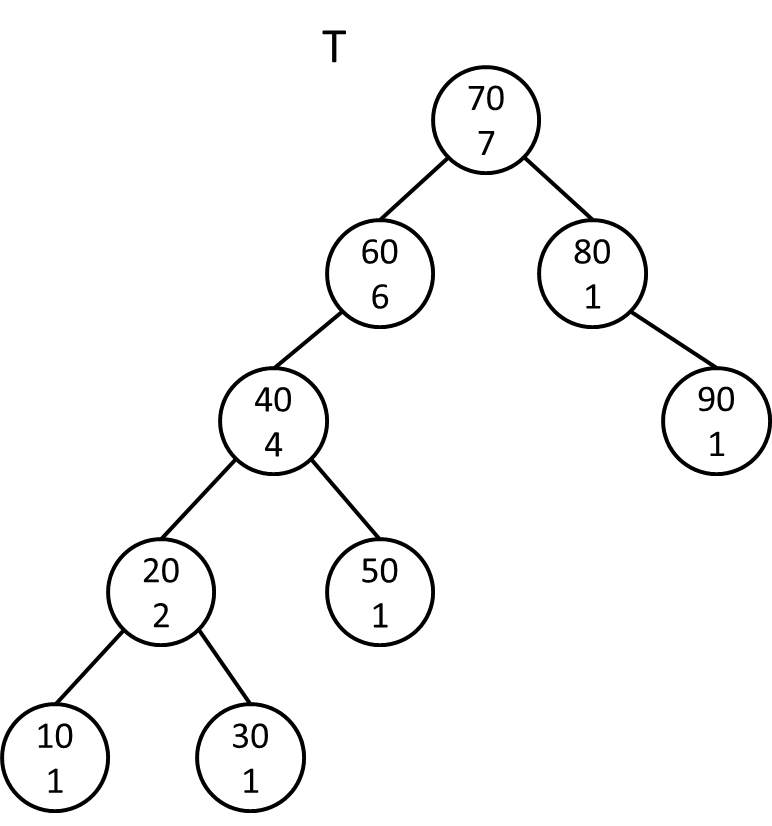
1. **(20%) Priority Queue (Heap)**
2. **(5%)** Compare the run-time performance (in big-O) of max heaps with that of unsorted and sorted linear lists as a representation of priority queue based on the following operations:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Top() | Push() | Pop() |
| Unsorted list |  |  |  |
| Sorted list |  |  |  |
| Heap |  |  |  |

1. **(5%)** Please reconstruct the max heap from a given output string of inorder traversal of the heap.

16, 62, 5, 95, 1, 88, 2, 99, 3, 14, 78, 10

1. **(10%)** Please illustrate EVERY steps of performing a delete operation on the above max heap.
2. **(20%) Binary Search Tree (BST)**



Given a binary search tree T with each node contains an additional data field leftSize which is one plus the number of nodes in the left subtree, please answer the following questions:

1. **(15%)** Please write down the codes of “Insert” and “Delete”, and “Max” operations of BST (Make sure you maintain the correct number of leftSize!)
2. **(5%)** Please illustrate EVERY steps of deleting a node 40 of the tree T.

|  |
| --- |
| class Node {  Node (Key k, Element e) { key = k; element = e; leftSize = 1;}  Node\* leftChild;  Node\* rightChild;  Key key;  Element element;  int leftSize;  } |
| class BST{  BST (void) { root = NULL; }  // Insert the node into BST  void insert (Node& n);  // Delete the node from BST  void delete (Node& n);  // Find the node with largest key in the left subtree  Node\* max (Node\* n);    Node\* root;  } |