

Practice-sheet

List, Stack, Binary Search Tree

1. Write an iterative as well as recursive function in C for reversing a singly linked list.
2. Given two singly linked lists storing n elements in ascending order. Transform these lists into one list where elements are stored in ascending order. The time complexity of the algorithm has to be $O(n)$. You may use only $O(1)$ extra space in your algorithm.
3. You are given the head of a singly linked list. You need to determine if this list loops (a node whose next pointer points to some node appearing earlier in the list) somewhere or not. Design the most efficient algorithm for this problem. Time complexity of the algorithm has to be $O(k)$ where k denotes the length of the list if there is no loop, otherwise k denotes the length of the longest loopless prefix of the list.
4. Circular linked list is a singly linked list where the next field of the last node points to (stores the address of) the first node of the list. What are the advantages/disadvantages of a circular linked list over a singly linked list and a doubly linked list ?

5. **Operations on Binary Search Tree**

Given a binary search tree T specified by the address (pointer) of its root node, design an algorithm

- (a) to compute its height.
- (b) to enumerate all the values stored in the tree in the decreasing order.
- (c) to transform this tree into another binary tree which is the **mirror image** of the original tree T . Note that the original tree loses its identity after this operation.

6. **Structure of Binary Tree**



- What can be the maximum number of nodes in a binary tree of height h ?
- Depth of a node v in a binary tree T is the number of edges on the path from root to node v . What can be the maximum number of nodes at depth d in a binary tree ?
- What can be minimum height of a binary tree having n nodes ?

7. Merging 2 Binary Search Trees

You are given 2 binary search trees T_1 and T_2 storing distinct elements. You need to create another BST that stores all the elements of $T_1 \cup T_2$. Your algorithm needs to run in $O(n)$ time where n is the sum of the number of elements in T_1 and T_2 . You may use $O(n)$ extra space.


8. Generating permutations using stack[Optional]

A stack can be used to generate some (not necessarily all) permutations of first n numbers. For example, for $n = 3$, we can generate $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 2, 1)$ using stacks but we can not generate permutation $(3, 1, 2)$ using stack.

- Design and implement an algorithm which prints all permutations of first n positive integers which can be generated by a stack. 
- Can you give a closed form expression for the number of permutations generated using a stack ? 

9. Two stacks

You need to implement 2 stacks in an algorithm/program. It is given that at every stage of time there will be total at most n elements in both the stacks. How will you implement the 2 stacks using a single array ?

10. **Think fresh ...** Company ElGoog has come to IITK for recruiting bright students. There are n students with roll number from 1 to n appearing for interview. Each student has information: $(name, roll\ no., section, \dots)$ associated with him/her. The interview board calls students in any arbitrary order. The information (record) of each student is destroyed just after the interview. Due to the reason best known to the board, while interviewing a student, with roll number, say i , they want to know the information about the student with maximum roll number less than i who has not been interviewed yet. We call such student as **pred**(i) (a shorthand for predecessor). Likewise we define **succ**(i) (a shorthand for successor). So design a data structure storing information of n students so that each of the following operations can be performed in worst case $O(1)$ time. (No assumption on the sequence of operations is permitted). 

- **Access**(i): return information about a student with roll number i .
- **Delete**(i): delete the record of a student with roll number i .
- **Pred**(i): return the roll no. of the predecessor of a student with roll number i .
- **Succ**(i): return the successor of an existing student with roll number i .

Hint: Think of an elegant way to use arrays for implementing a doubly linked list.