DSA ASSIGNMENT-3(18CSC201J)

Contributors:

ARAMREDDY HARICHANDANA-RA191102601009

D SAI NIHARIKA-RA1911026010040

THEJASWIN.S-RA1911026010029

AKASH KOOTTUNGAL-RA1911026010010

Section: CSE-AIML K1

Faculty: Mr. Eliazer

Software Used: CodeBlocks ,Microsoft word

DESIGN OF ABSTRACT DATA TYPES (ADT) OF DATASTRUCTURES

Abstract data type:

An abstract data type refers to a set of data values and associated operations that are specified accurately, independent of any particular implementation. With an ADT, we know what a specific data type can do, but how it actually does is hidden. In broader terms, the ADT consists of a set of definitions that allow us to use the functions while hiding the implementation.

Abstract Data Type of a Queue:

The queue abstract data type is defined by the following structure and operations. A queue is structured, as described above, as an ordered collection of items which are added at one end, called the “rear,” and removed from the other end, called the “front.” Queues maintain a FIFO ordering property. The queue operations are given below.

- **Queue() :** creates a new queue that is empty. It needs no parameters and returns an empty queue.
- **enqueue(item) :** adds a new item to the rear of the queue. It needs the item and returns nothing.
- **dequeue() :** removes the front item from the queue. It needs no parameters and returns the item. The queue is modified.
- **isEmpty() :** tests to see whether the queue is empty. It needs no parameters and returns a boolean value.

- **size() :** returns the number of items in the queue. It needs no parameters and returns an integer.

Abstract Data Type of a Stack:

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO. The stack operations are given below.

- **Stack()** : creates a new stack that is empty. It needs no parameters and returns an empty stack.
- **push(item) :** adds a new item to the top of the stack. It needs the item and returns nothing.
- **pop()** : removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- **peek()** : returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- **isEmpty():** tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- **size()** : returns the number of items on the stack. It needs no parameters and returns an integer.

Abstract Data Type of a Linked List:

Linked List is an *Abstract Data Type (ADT)* that holds a collection of Nodes, the nodes can be accessed in a sequential way. Linked List doesn't provide a random access to a Node.

Usually, those Nodes are connected to the next node and/or with the previous one, this gives the linked effect. When the Nodes are connected with only the next pointer the list is called **Singly Linked List** and when it's connected by the next and previous the list is called **Doubly Linked List.**

Abstract Data Type of a Binary tree:

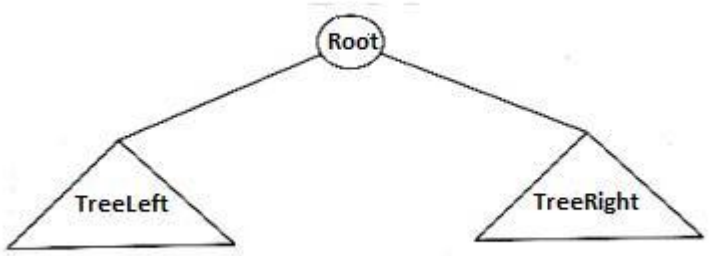
Tree: Recursive data structure (similar to list)

¤ Each node may have zero or more successors (children)

¤ Each node has exactly one predecessor (parent) except the root, which has none

¤ All nodes are reachable from root Binary tree: tree in

A binary tree is defined as a tree in which no node can have more than two children. The highest degree of any node is two. This indicates that the degree of a binary tree is either zero or one or two.



In the above fig., the binary tree consists of a root and two sub trees TreeLeft & TreeRight. All nodes to the left of the binary tree are denoted as left subtrees and all nodes to the right of a binary tree are referred to as right subtrees.

Description of how we implemented our program:

In our project, we have performed **enqueue** and **dequeue** operations on the queue and displayed each of the elements from the queue, and then again dequeued the elements from the queue and pushed them into the stack. Then we popped them from the stack and enqueued again into the queue. In the stack we have performed **push** and **pop** operations.Now,the elements which are present now in the queue are in the reverse order. Now, we will dequeue elements from the queue and insert them into the unordered binary tree. We will perform **Preorder traversal** and print the elements and **Post order traversal** and print them. Now we will perform **inorder** traversal and insert the data in linked list through the inorder traversal. We will print the elements in the linkedlist and we will perform the sorting operation using the quick sort.

Here, we have hidden the algorithms which will be used for implementing these operations-enqueue, dequeue, push, pop and not specified how the elements are reversed using the stack in the main

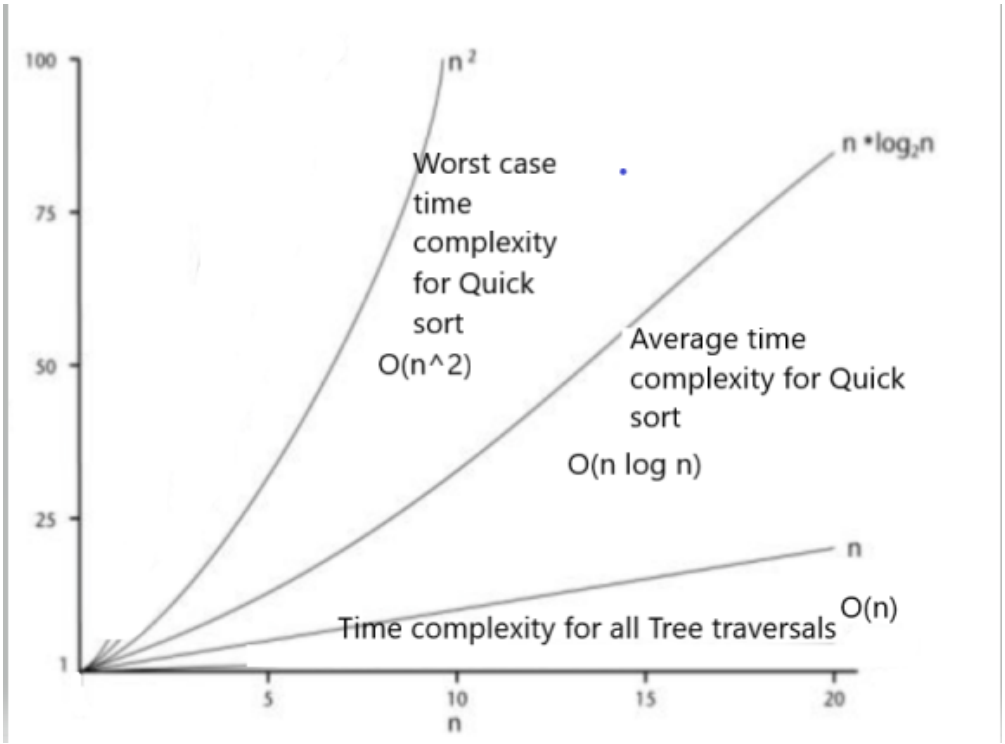
function and how these datatypes store the data. Because, **we** are concerned only with what the **data** is representing and not with how it will eventually be constructed and for the purpose of protecting the data from the direct access by the users.

We have used C++ language for code implementation

Time complexities for various operations in our program:

- The time complexity for both enqueue and dequeue operations in the queue is O(1).
- The time complexity for push and pop operations in the stack is O(1).
- The time complexity to insert an elements into an unordered binary tree is O(h) where h is the height of BST.
- The time complexity for preorder traversal is O(n)
- The time complexity for postorder traversal is O(n)
- The time complexity for inorder traversal is O(n).
- The time complexity for inserting an elements into the linked list from the queue is O(1)
- The time complexity for sorting the elements using quick sort algorithm in the linked list is

  The worst case time complexity of this algorithm is O(n^2) and the average case complexity is O(nlogn).



Description how we can use our program:

- We can use our program for enqueuing and dequeuing the elements into the queue ADT.
- Through our program, we can reverse the elements from the queue.
- We can perform preorder, postorder and inorder traversals through our program.
- And we can use our program to sort the elements in the linked list by using the quick sort algorithm.