University of Sargodha

BS 3rd Term Examination 2016

Subject: Computer Science Paper: Data Structure & Algorithms (CMP-3113)

Time Allowed: 2:30 Hours **Maximum Marks: 80**

Note: **Objective part is compulsory. Attempt any three questions from subjective part.**

Objective Part (Compulsory)

Q 1. Attempt all questions each required in 2-3 lines having equal marks? (2*16)

i. What is data-structure?
ii. What is an algorithm?
iii. What are asymptotic notations?
iv. What is divide and conquer approach.
v. What is the post fix notation of (a + b) * (c + d)?
vi. Define an Abstract Data Type (ADT)
vii. What is the difference between a PUSH and a POP?
viii. Why do we use queues?
ix. List out the advantages of using a linked list.
x. What are doubly linked lists?
xi. What is recursion?
xii. What is a binary tree?
xiii. What is a graph?
xiv. What is hashing?
xv. Which sorting algorithm is best if the list is already sorted? Why?
xvi. What is max heap?

Subjective Part (3*16)

Q 2. Write a programmer / algorithm to do the following.

i. Find **sum** of element of array by using **recursion**.
ii. Program / algorithm to POP a value in a stack.

Q 3. What are circular queue? Write down routines / algorithms for inserting and deleting elements from a circular queue implemented using arrays.

Q 4. What is a Binary Search Tree (BST)? Make a BST for the following sequence of numbers.

45,32,90,34,68,72,15,24,30,66,11,50,10

Traverse the tree in **preorder**, **inorder** and **postorder**.

Q 5. Write a programmer / algorithm to do the following operations

i. **Delete** a node **at the end of single link** list
ii. **insert** the **first node in the circular link** list

Q 6. Write an algorithm ofMerge sort method. Also apply merge sort on the following data.

10, 4, 5, 3, 100, 30, 85, 15, 70

Objective

Q 1: Short Questions Answers.

i. What is Data-Structure?

Data structure is a specialized format for organizing and storing data, so that we can access and modify that data easily later. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

ii. What is an algorithm?

An algorithm is defined as a step-by-step procedure or method for solving a problem by a computer in a finite number of steps. Steps of an algorithm definition may include branching or repetition depending upon what problem the algorithm is being developed for.

iii. What are the asymptotic notations?

Asymptotic analysis of an algorithm refers to defining the mathematical foundation /framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as g(n2). This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Usually, the time required by an algorithm falls under three types –

Best Case – Minimum time required for program execution.

Average Case – Average time required for program execution.

Worst Case – Maximum time required for program execution.

iv. What is divide and conquer approach?

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently.
When we keep on dividing the sub-problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible.
Those atomic smallest possible sub-problem (fractions) are solved.
The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

v. What is the postfix notation of (a+b)*(c+d)?

i. (a+b)*(c+d)
ii. (ab+)*(cd+)
iii. ab + cd + *

vi. Define ADT (Abstract data Type)?

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction. Three ADTs namely List ADT, Stack ADT, Queue ADT.

(is question mn us na ADT k about puxha hy , just upper sa kuch definition jo underline hy wo ikh lyn to kafi hoga, but wo in types mn bhi puch skta hy is ly unki details bhi bta di hy)

List ADT

A list contains elements of same type arranged in sequential order and following operations can be performed on the list.
get() – Return an element from the list at any given position.
insert() – Insert an element at any position of the list.
remove() – Remove the first occurrence of any element from a non-empty list.
removeAt() – Remove the element at a specified location from a non-empty list.
replace() – Replace an element at any position by another element.
size() – Return the number of elements in the list.
isEmpty() – Return true if the list is empty, otherwise return false.
isFull() – Return true if the list is full, otherwise return false.

Stack ADT

A Stack contains elements of same type arranged in sequential order. All operations takes place at a single end that is top of the stack and following operations can be performed:
push() – Insert an element at one end of the stack called top.
pop() – Remove and return the element at the top of the stack, if it is not empty.
peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
size() – Return the number of elements in the stack.
isEmpty() – Return true if the stack is empty, otherwise return false.
isFull() – Return true if the stack is full, otherwise return false.

Queue ADT

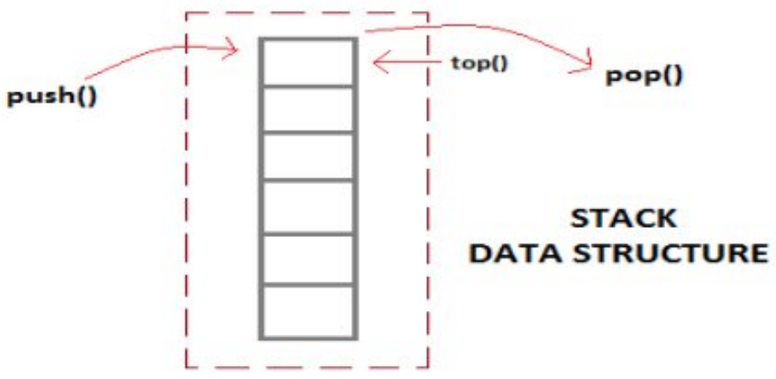
A Queue contains elements of same type arranged in sequential order. Operations takes place at both ends, insertion is done at end and deletion is done at front. Following operations can be performed:
enqueue() – Insert an element at the end of the queue.
dequeue() – Remove and return the first element of queue, if the queue is not empty.
peek() – Return the element of the queue without removing it, if the queue is not empty.
size() – Return the number of elements in the queue.
isEmpty() – Return true if the queue is empty, otherwise return false.
isFull() – Return true if the queue is full, otherwise return false.

vii. What is the difference between PUSH and POP?

Stack is an ordered list of similar data type.
Stack is a **LIFO(Last in First out)** structure or we can say **FILO(First in Last out).**
push() function is used to insert new elements into the Stack and **pop()** function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called **Top**.
Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.

viii. Why do we use queues?

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the **top** of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.



ix. List out the advantages of using a linked list?

Advantages of Linked List

a) Dynamic Data Structure. Linked list is a dynamic data structure so it can grow and shrink at runtime by allocating and de-allocating memory.
b) Insertion and Deletion. Insertion and deletion of nodes are really easier.
c) No Memory Wastage.
d) Implementation.
e) Memory Usage.
f) Traversal.
g) Reverse Traversing.

x. What are doubly linked lists?

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

xi. What is recursion?

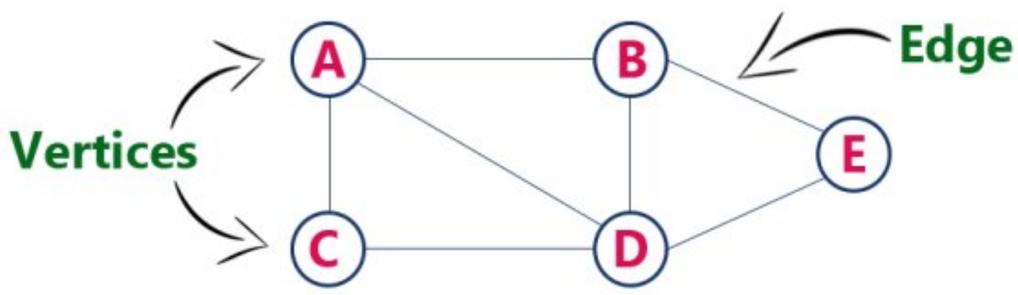
Some computer programming languages allow a module or function to call itself. This technique is known as recursion. In recursion, a function α either calls itself directly or calls a function β that in turn calls the original function α. The function α is called recursive function.

xii. What is a binary tree?

In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. A binary tree is a finite set of elements that are either empty or is partitioned into three disjoint subsets.

xiii. What is a graph?

A **graph** is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.



xiv. What is hashing?

In computing, a hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

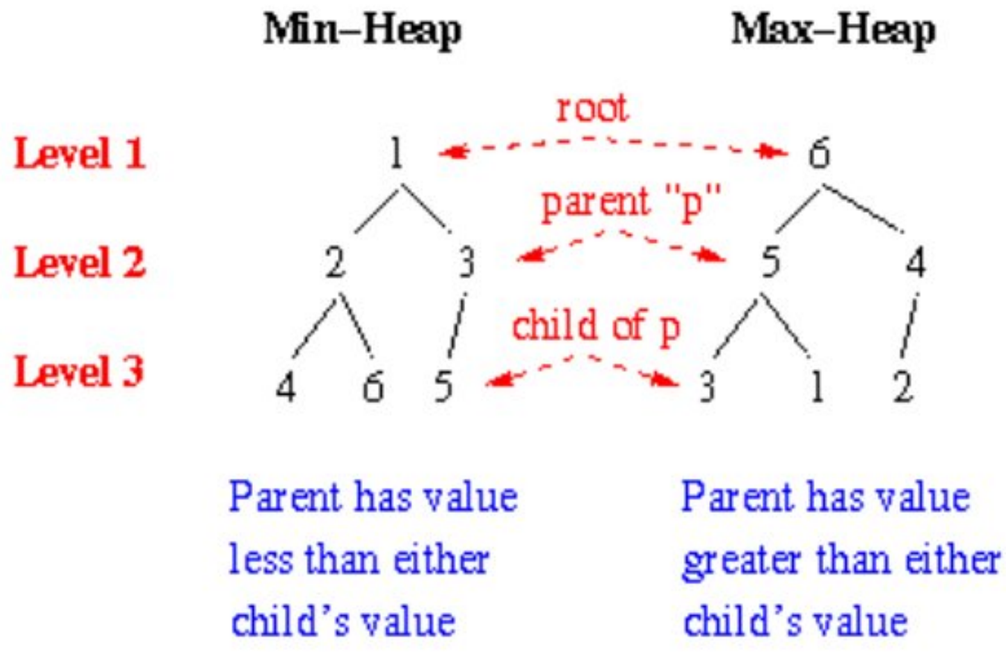
A technique used to uniquely identify a specific object from a group of similar objects. To convert a range of key values into a range of indexes of an array by using a **hash function.**

xv. Which sorting algorithm is best if list is already sorted? Why?

Insertion sort. Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

xvi. What is a MAX heap?

A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node. A min-heap is defined similarly.



Subjective

Q 2. Write a program/ algorithm to do the following.

a) Find sum of elements of array by using recursion.

```
class Test {
    static int arr[] = { 1, 2, 3, 4, 5 };

    // Return sum of elements in A[0..N-1]
    // using recursion.
    static int findSum(int A[], int N)
    {
        if (N <= 0)
            return 0;
        return (findSum(A, N - 1) + A[N - 1]);
    }

    // Driver method
    public static void main(String[] args)
    {
        System.out.println(findSum(arr,
arr.length));
    }
}
```

b) To POP a value in the stack.

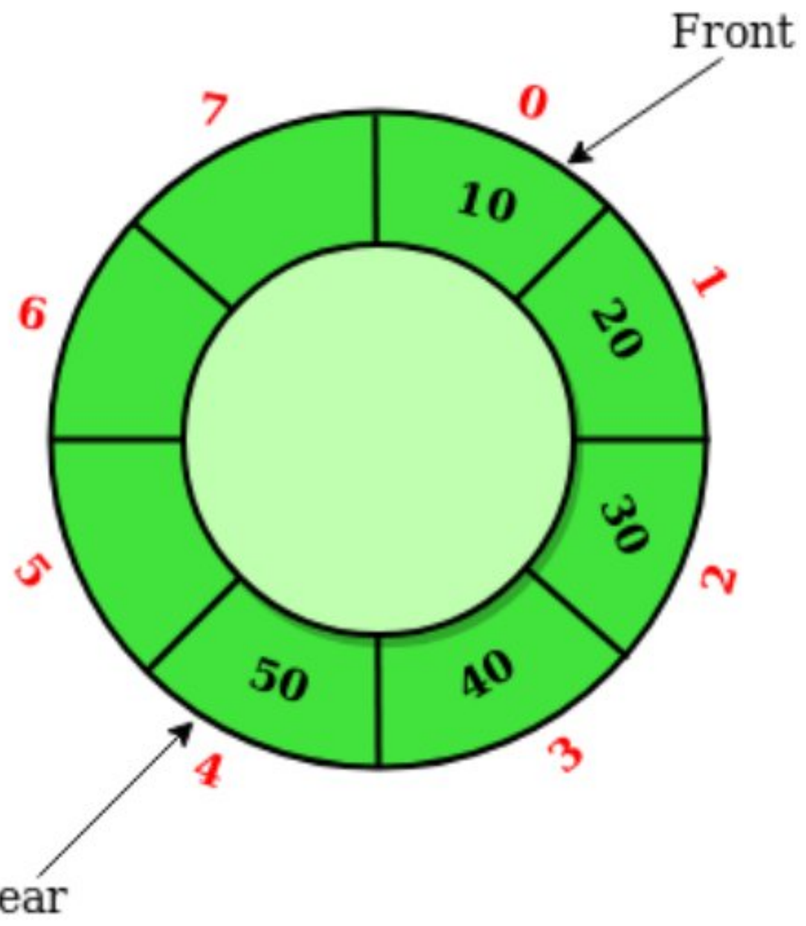
```
public class TestStringStack {
public static void main(String[] args) {
stack.push("GB");
stack.push("DE");
stack.push("FR");
stack.push("ES");
System.out.println(stack);
System.out.println("stack.peek(): " +
stack.peek());
System.out.println("stack.pop(): " +
stack.pop());
System.out.println(stack);
```

```
        System.out.println("stack.pop(): " +
        stack.pop());
        System.out.println(stack);
        System.out.println("stack.push(IE): ");
        stack.push("IE");
        System.out.println(stack);
         }
        }
```

x_____x

Q 3. What is a Circular Queue? Write down routines algorithm for inserting and deleting element from circular queue implemented using arrays.

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'**.



Operations on Circular Queue:

- **Front:** Get the front item from queue.
- **Rear:** Get the last item from queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

enQueue(value) - Inserting value into the Circular Queue

In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...

- **Step 1: Check whether** queue **is** FULL.
  ((rear == SIZE-1 && front == 0) || (front == rear+1))
- **Step 2: If it is** FULL**, then display** "Queue is FULL!!! Insertion is not possible!!!" **and terminate the function.**
- **Step 3: If it is** NOT FULL**, then check** rear == SIZE - 1 && front != 0 **if it is** TRUE**, then set** rear = -1**.**
- **Step 4: Increment** rear **value by one** (rear++)**, set** queue[rear] **=** value **and check** 'front == -1' **if it is** TRUE**, then set** front = 0**.**

deQueue() - Deleting a value from the Circular Queue

In a circular queue, deQueue() is a function used to delete an element from the circular queue. In a circular queue, the element is always deleted from **front**position. The deQueue() function doesn't take any value as parameter. We can use the following steps to delete an element from the circular queue...

- **Step 1:** Check whether **queue** is **EMPTY.** (**front == -1 && rear == -1**)
- **Step 2:** If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3:** If it is **NOT EMPTY**, then display **queue[front]** as deleted element and increment the **front** value by one (**front ++**). Then check whether **front == SIZE**, if it is **TRUE**, then set **front = 0**. Then check whether both **front - 1** and **rear** are equal (**front -1 == rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

x_____x

Q 4. What is Binary Search Tree (BST)? Make a BST for the following sequence of numbers.

45,32,90,34,68,72,15,24,30,66,11,50,10

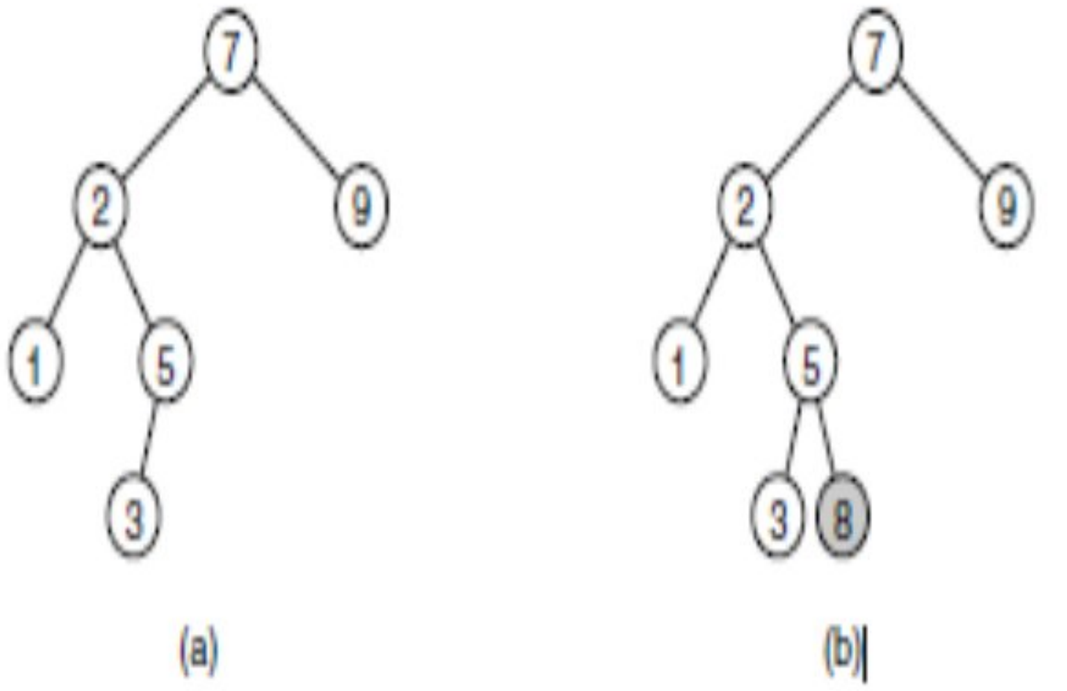
A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following conditions: -

All keys (if any) in the left sub tree of the root precede the key in the root.

- The key in the root precedes all keys (if any) in its right sub tree.
- The left and right sub trees of the root are again search trees.

The binary search tree, a simple data structure that can be viewed as extending the binary search algorithm to allow insertions and deletions. The running time for most operations is O(log N) on average. Unfortunately, the worst-case time is O(N) per operation.

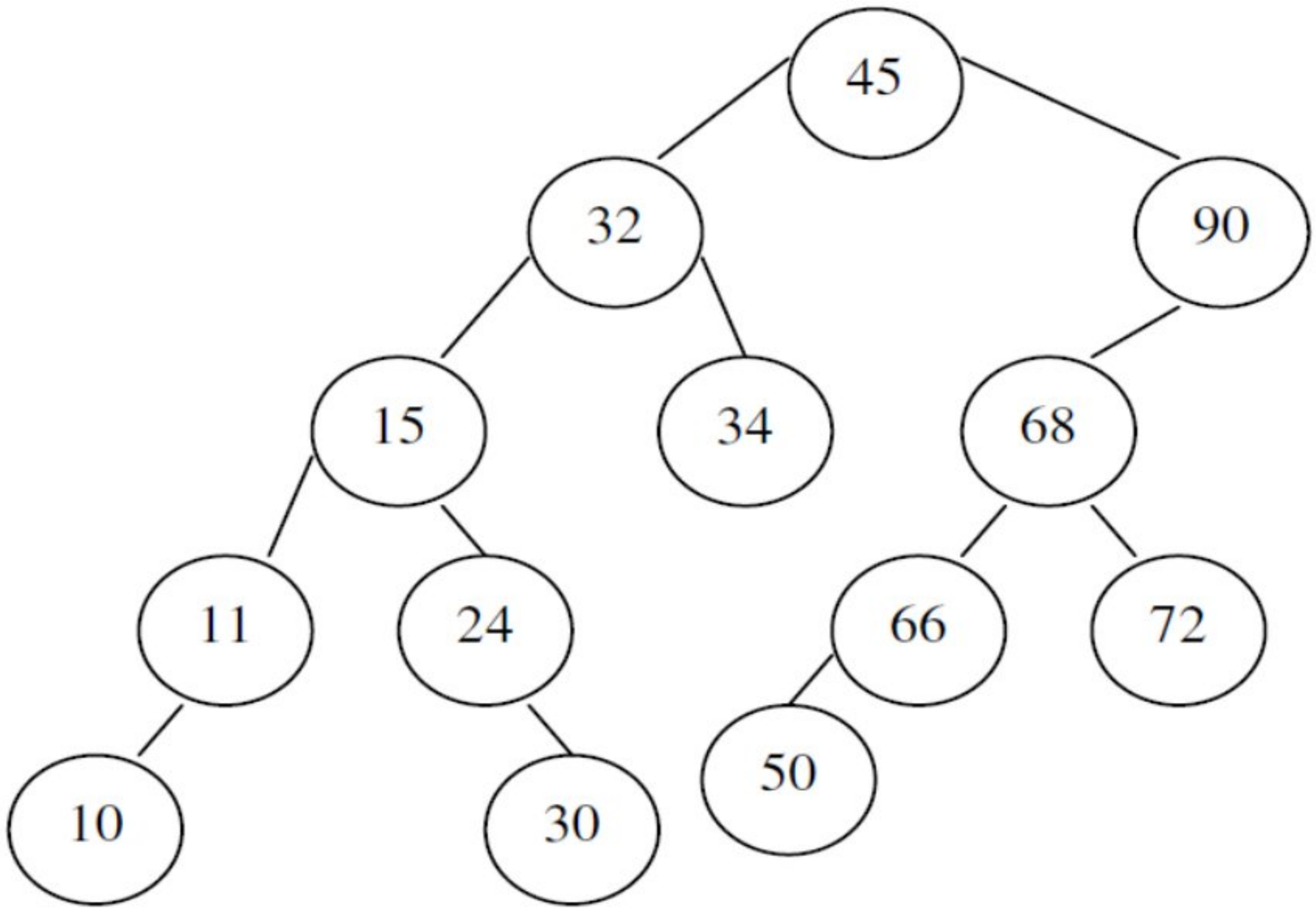
In the general case, we search for an item (or element) by using its key. For instance, a student transcript could be searched on the basis of a student ID number. In this case, the ID number is referred to as the item's key. The binary search tree satisfies the search order property; that is, for every node X in the tree, the values of all the keys in the left sub-tree are smaller than the key in X and the values of all the keys in the right sub-tree are larger than the key in X. The tree shown in fig



The property that makes a binary tree into a binary search tree is that for every node, X, in the tree, the values of all the keys in the left sub-tree are smaller than the key value in X, and the values of all the keys in the right sub-tree are larger than the key value in X.

The Binary Search Tree for given data

45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10



The traversals for the above tree is as follows:-

- **Pre-order traversal is-**

  45 , 32, 15, 11, 10, 24, 30, 34, 90, 68, 66, 50, 72

- **In-order traversal is:-**

  10, 11, 15, 24, 30, 32, 34, 45, 50, 66, 68, 72, 90

- **Post-order traversal:-**

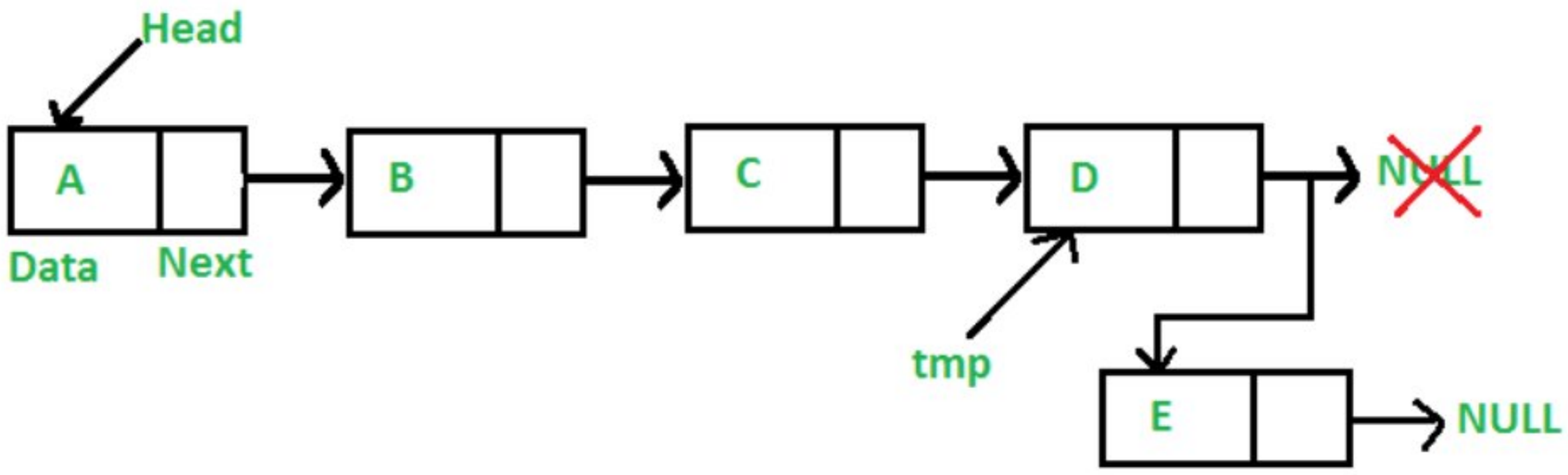
  10, 11, 30, 24, 15, 34, 32, 50, 66, 72, 68, 90, 45

Q 5. Write a Program/Algorithm to do the following operations

i. **Insert a new node at the end of Single link list**
ii. **Delete the first node in Single link list.**

Ans: i) **Add a node at the end: (6 steps process)**

The new node is always added after the last node of the given Linked List.




```
public void append(int new_data)
{
/* 1. Allocate the Node &
2. Put in the data
3. Set next as null */
Node new_node = new Node(new_data);

/* 4. If the Linked List is empty, then make the
new node as head */
if (head == null)
{
head = new Node(new_data);
return;
}

/* 4. This new node is going to be the last node, so
make next of it as null */
new_node.next = null;

/* 5. Else traverse till the last node */
```

```
Node last = head;
while (last.next != null)
last = last.next;

/* 6. Change the next of last node */
last.next = new_node;
return;
}
```

Is question mn jo Pucha hy us na k Start wala element delet krna ho to usk lye code/algorithm Red color mn hy, jo light green mn hn wo comments hn apko smjhany k lye k idr kia kam ho rha, and jo red sa nechy hn wo agr head ky elawa ksi jaga sa delete krna ho Single linked list mn to usk lye code ha.

```
/* Given a reference (pointer to pointer) to the head of a list
     and a position, deletes the node at the given position */

void deleteNode(int position)
  {
     // If linked list is empty
     if (head == null)
        return;

     // Store head node
     Node temp = head;

     // If head needs to be removed
     if (position == 0)
     {
        head = temp.next;   // Change head
        return;
     }

// Find previous node of the node to be deleted

for (int i=0; temp!=null && i<position-1; i++)
   temp = temp.next;

// If position is more than number of ndoes

if (temp == null || temp.next == null)
   return;

// Node temp->next is the node to be deleted
```

```
        // Store pointer to the next of node to be deleted
        Node next = temp.next.next;

        temp.next = next;
    }
```

x_____x

Q 6. Write an algorithm of merge sort method? Also apply merge sort on the following data

10,4,5,3,100,30,85,15,70

- **Algorithm**

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

Step 1 – if it is only one element in the list it is already sorted, return.
Step 2 – divide the list recursively into two halves until it can no more be divided.
Step 3 – merge the smaller lists into new list in sorted order.

- **Algorithm steps:**

MergeSort(arr[], l, r) // l for Left and r for right half

If (r > l)

1. Find the middle point to divide the array into two halves:
   middle m = (l+r)/2
2. Call mergeSort for first half:
   Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
   Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
   Call merge(arr, l, m, r)

→ Merge Sort