University of Sargodha

B.S 3rd Term Examination 2017

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=32)

i. What is Abstract Data Type
ii. Define O-notation?
iii. What is linear data structure?
iv. When is a binary search best applied?
v. List out the advantages of using a linked list.
vi. Why do we use stacks?
vii. Define PUSH and POP operation.
viii. What is FIFO?
ix. What is the postfix of (a + b / c * (d * e)?
x. What is merge sort?
xi. Define a complete binary tree.
xii. Define leaves node in tree.
xiii. What is a spanning tree?
xiv. What is a graph?
xv. What is max heap?
xvi. What is hashing?

Subjective Part (3*16 = 48)

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

i. Find **max** element of array by using **recursion**.
ii. Program / algorithm to PUSH 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, 21, 78, 65, 87, 132, 90, 96, 41, 74, 92

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

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

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

Q 6. Write an algorithm of Quick sort method. Describe the behavior of Quick sort when input is already sorted.

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

Objective

Q 1: Sort Answers

i. What is 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.

ii. Define O-notation?

Definition: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. **Formal Definition:** $f(n) = O(g(n))$ means there are positive constants c and k, such that $0 \leq f(n) \leq cg(n)$ for all $n \geq k$.

iii. What is linear data structure?

A data structure is classified into two categories: Linear and Non-Linear data structures. A data structure is said to be linear if the elements form a sequence, for example Array, Linked list, queue etc. Elements in a nonlinear data structure do not form a sequence, for example Tree, Hash tree, Binary tree, etc.

iv. When is a binary search best applied?

Binary search is a type of algorithm. It is best applied to search in a list in which the elements are already in order or sorted.

The binary search algorithm starts searching the list from the middle. If the middle value is not the correct one, then it will go on to search the top or the bottom half in a similar manner, i.e. it will then divide the top or the bottom part into halves and start searching from its middle. It will continue to do this until the searched for value is found.

v. List out the advantages of using a linked lists?

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.

vi. Why do we use stacks?

In computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and. pop, which removes the most recently added element that was not yet removed.

i. Stack is an ordered list of similar data type.
ii. Stack is a **LIFO(**Last in First out) structure or we can say **FILO**(First in Last out).
iii. 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.
iv. 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.

Analysis of Stack Operations

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure.

- **Push Operation** : O(1)
- **Pop Operation** : O(1)
- **Top Operation** : O(1)
- **Search Operation** : O(n)

vii. Define PUSH and POP operation?

- 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. What is FIFO?

FIFO is an acronym for first in, first out, a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. ... A priority queue is neither FIFO or LIFO but may adopt similar behavior temporarily or by default

ix. What is the postfix of (a+b/c*(d*e)?

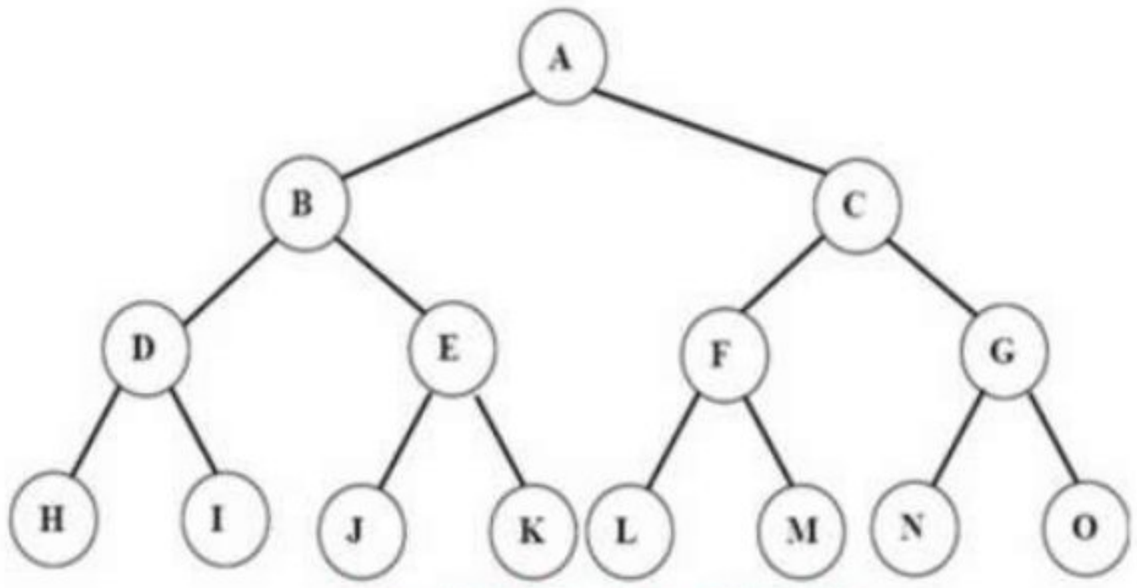
The Postfix of
(a+b/c*(d*e) is
a b c / d e * * +

x. What is merge sort?

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves.

xi. Define a complete binary Tree?

A complete binary tree is a binary tree in which every level, is completely filled, and all nodes are as far left as possible.
A binary tree of depth d is called complete binary tree if all of whose leaves are at level d.
Every Level of Tree A is completely filled is called complete binary tree.



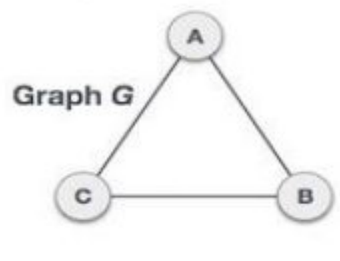
A complete Binary Tree of depth 3

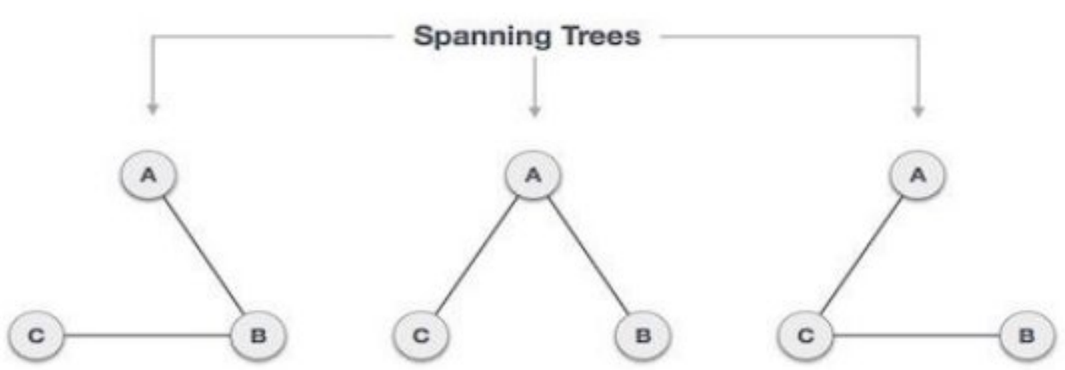
xii. Define leaves node in Tree?

Some binary tree implementations store data only at the leaf nodes, using the internal nodes to provide structure to the tree. By definition, a leaf node does not need to store pointers to its (empty) children. In simple we can say, Node which does not have any child node is called leaf node.

xiii. What is a spanning tree?

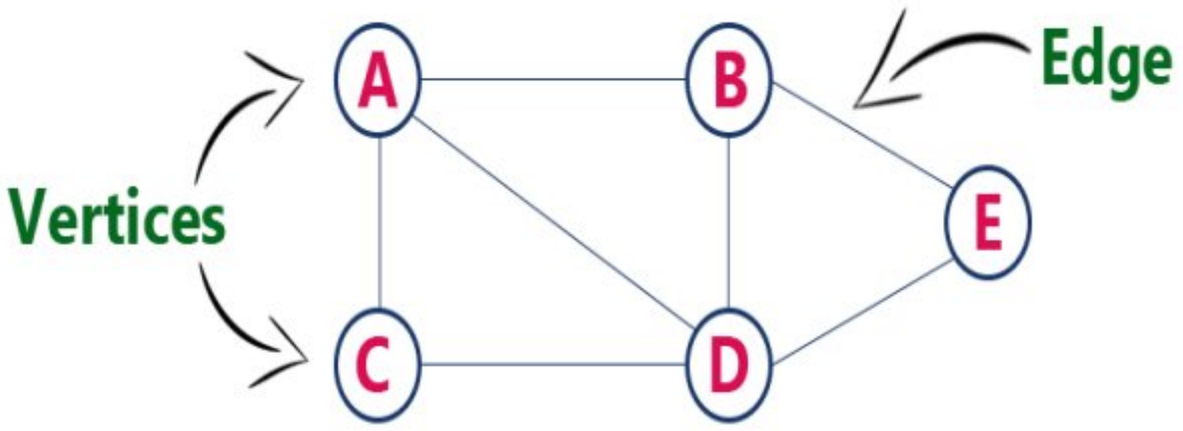
A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..
By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.





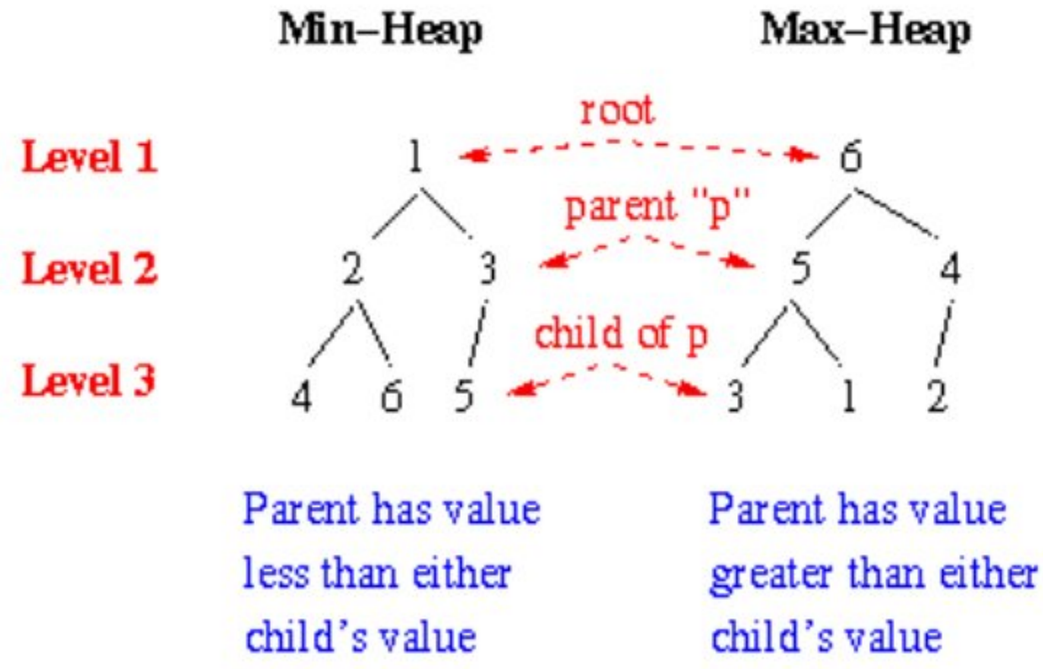
xiv. What is a graph?

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.



xv. 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.



xvi. 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.**

Subjective

Q 2. Write a Program/Algorithm to do the following

i. **Find Max element of array by using Recursion**
ii. **Program/ Algorithm to PUSH a value in the Stack.**

ANS: i)

```
// function to return maximum element using recursion
   public static int findMaxRec(int A[], int n)
   {
     // if size = 0 means whole array
     // has been traversed
     if(n == 1)
       return A[0];

       return Math.max(A[n-1], findMaxRec(A, n-1));
   }
```

ii)

```
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);
 }
}
```

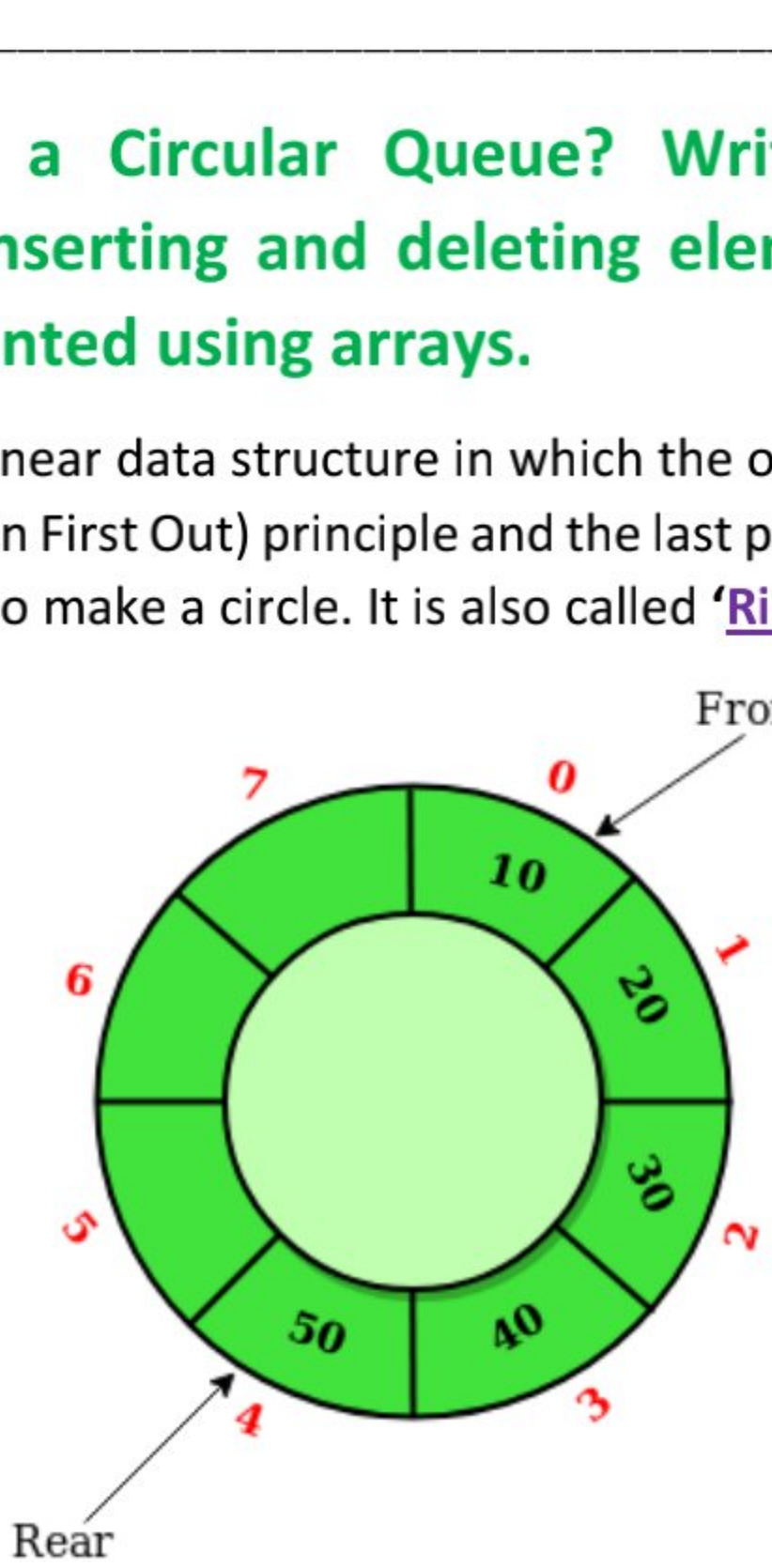
- **The output is:**

```
[ES, FR, DE, GB]
stack.peek(): ES
stack.pop(): ES
            [FR, DE, GB]
stack.pop(): FR
            [DE, GB]
stack.push("IE"):
            [IE, DE, GB]
```

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.

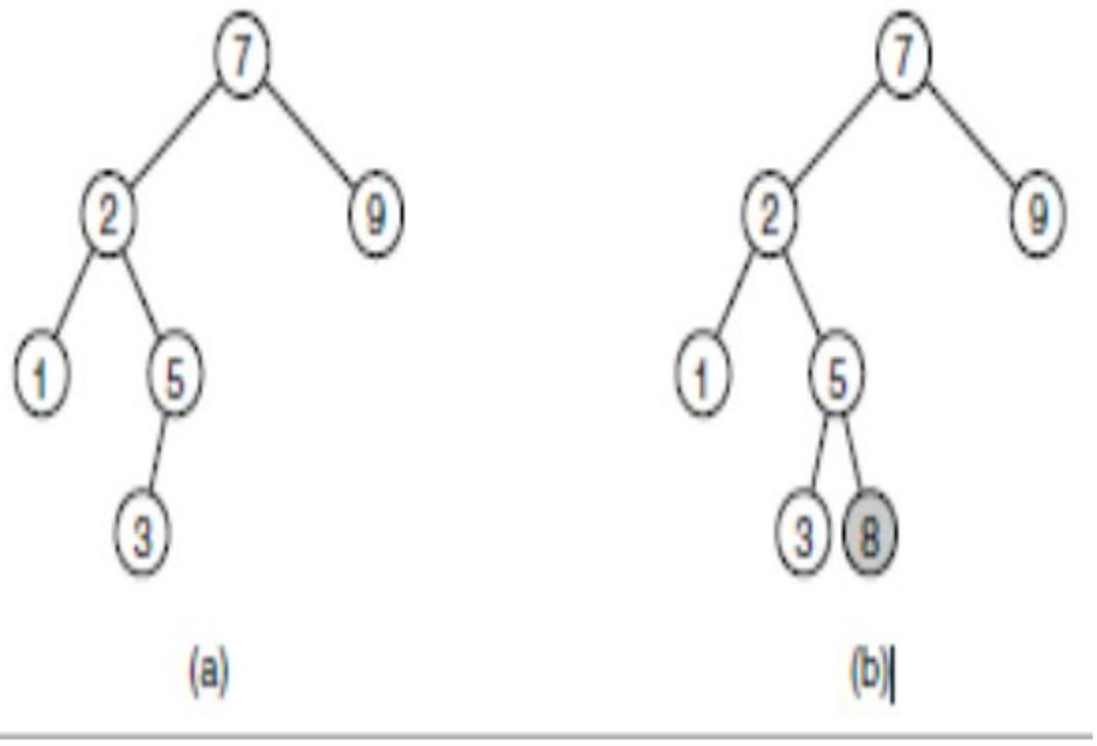
45,32,90,21,78,65,87,132,90,96,41,74,92

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.

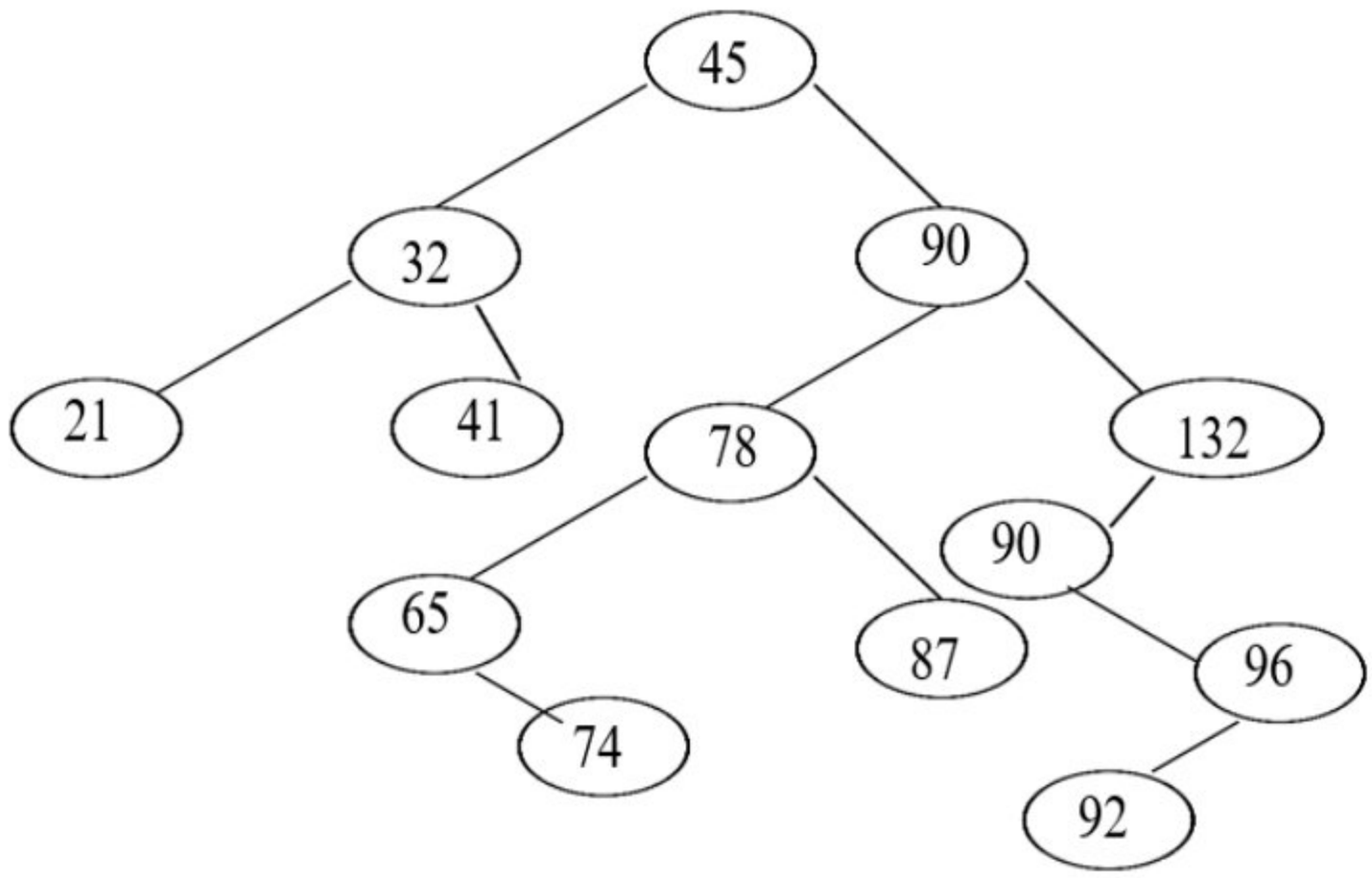
In the general case, we search for an item (or element) by using its key. For instance, a student transcript could be searched on the basis of a student ID number. In this case, the ID number is referred to as the item's key. The binary search tree satisfies the search order property; that is, for every 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



(a) (b)

BST for following sequence

45,32,90,21,78,65,87,132,90,96,41,74,92 is

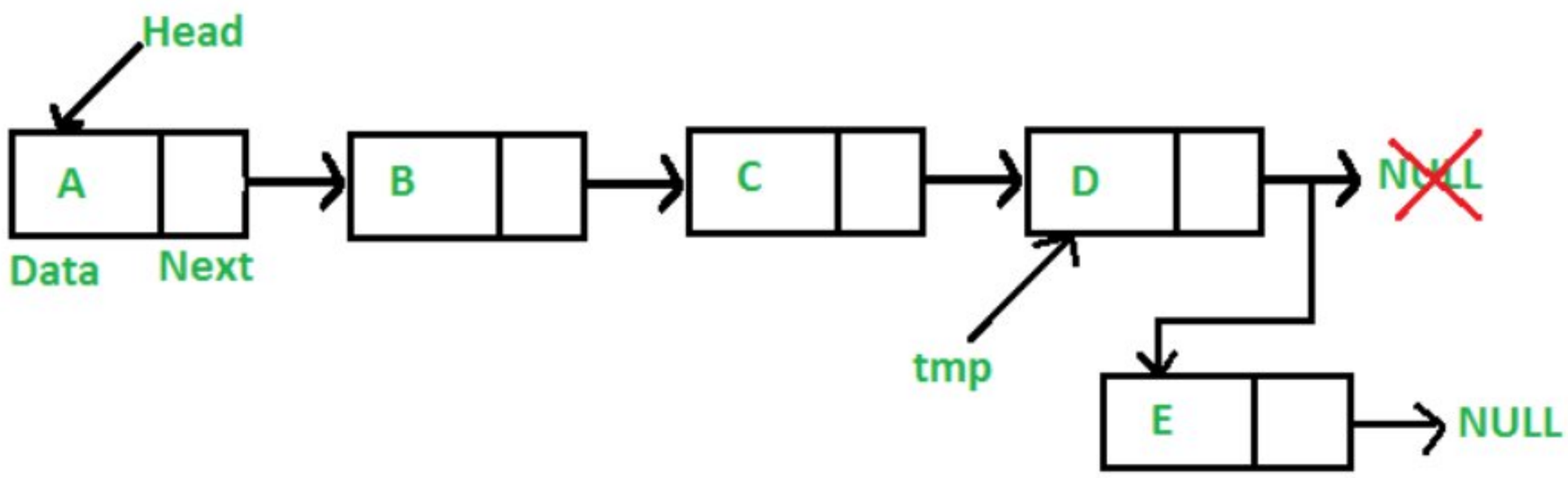


Q 5. Write a Program/Algorithm to do the following operations

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;
}
```

ii) 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;
}
```

Q 6. Write an Algorithm of Quick Sort Method? Describe the behavior of quick sort when input is already sorted.

Ans:

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.

```
// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
```

```
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
               now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
```

```
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = arr.length;

    QuickSort ob = new QuickSort();
    ob.sort(arr, 0, n-1);

    System.out.println("sorted array");
    printArray(arr);
  }
}
```

Describe the behavior of quick sort when input is already sorted.

The answer depends on strategy for choosing pivot. In early versions of Quick Sort where leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.

1) Array is already sorted in same order.
2) Array is already sorted in reverse order.
3) All elements are same (special case of case 1 and 2)

Since these cases are very common use cases, the problem was easily solved by choosing either a random index for the pivot, choosing the middle index of the partition or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot. With these modifications, the worst case of Quick sort has less chances to occur, but worst case can still occur if the input array is such that the maximum (or minimum) element is always chosen as pivot.