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)?

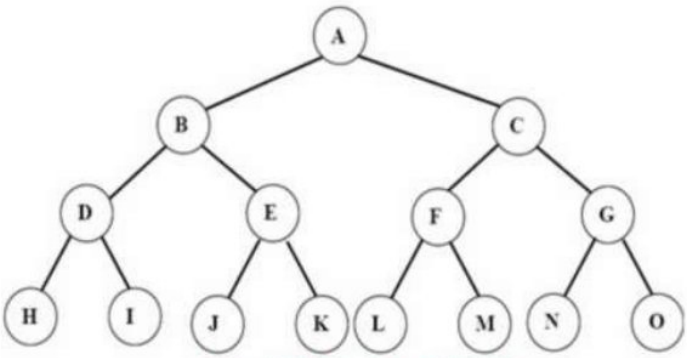
The Postfix of
(a+b/c*(d*e) is
a b c / d e * * +

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.



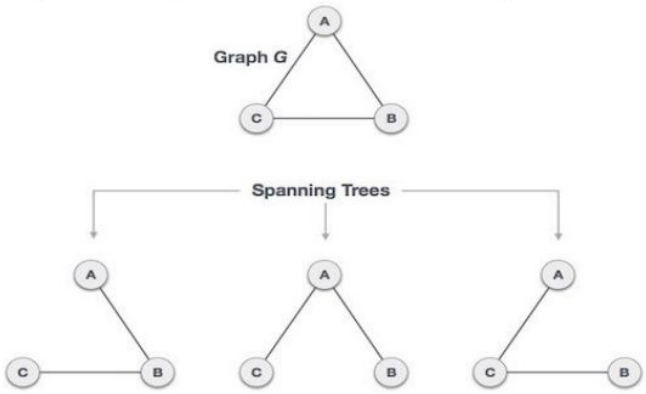
A complete Binary Tree of depth 3

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.

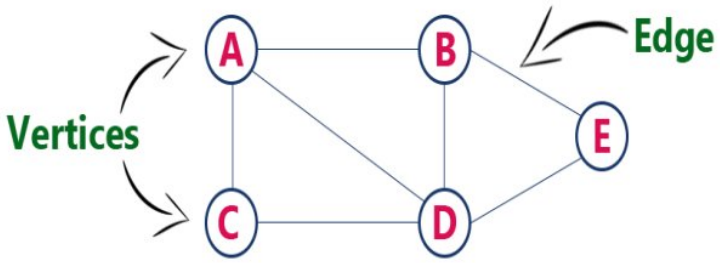
xiii. What is a spanning tree?

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.



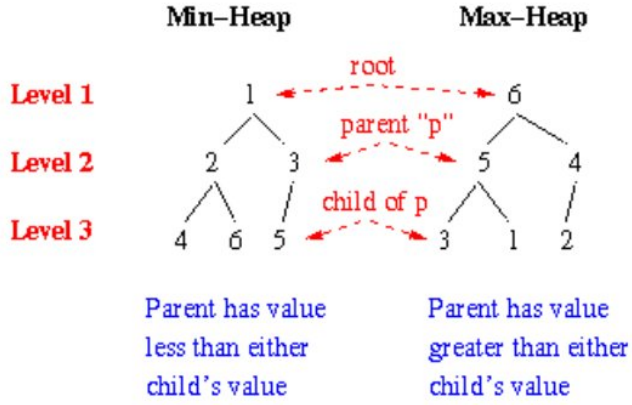
xiv. What is a graph?

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);
 }
}
```

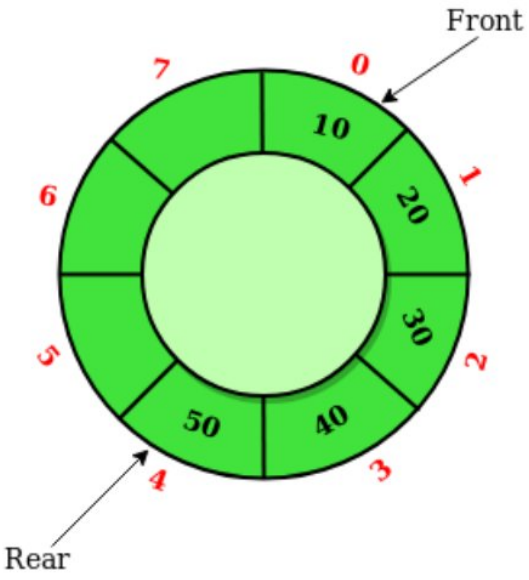
- **The output is:**

```
[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.

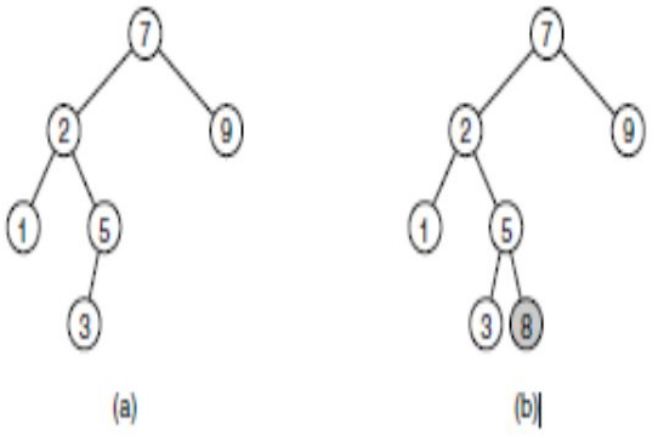
45,32,90,21,78,65,87,132,90,96,41,74,92

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.

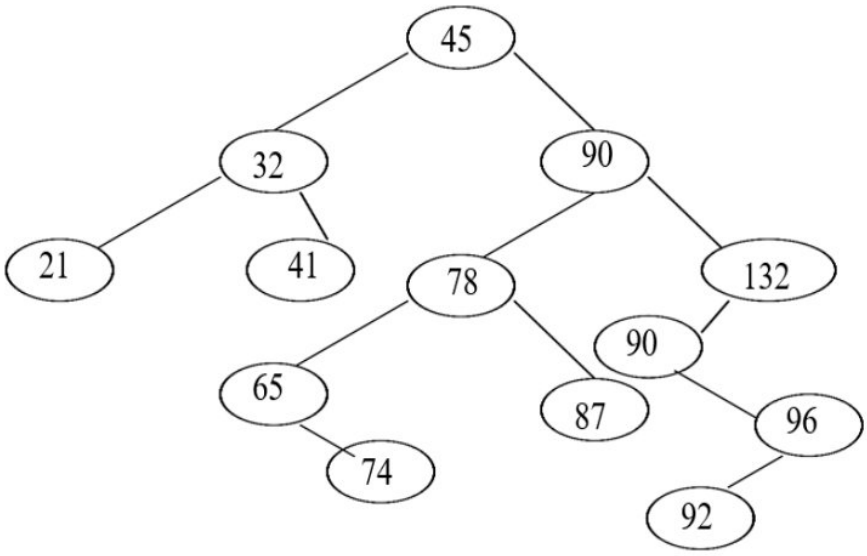
The binary search tree, a simple data structure that can be viewed as extending the binary search algorithm to allow insertions and deletions. The running time for most operations is O(log N) on average. Unfortunately, the worst-case time is O(N) per operation.

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



BST for following sequence

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

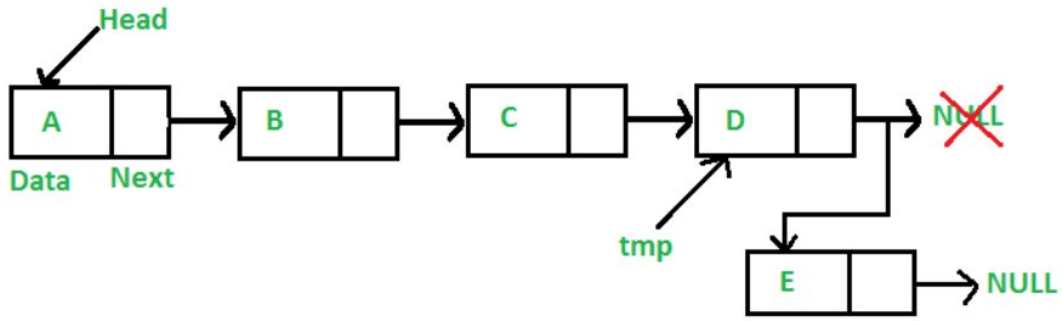


Q 5. Write a Program/Algorithm to do the following operations

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.

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)?

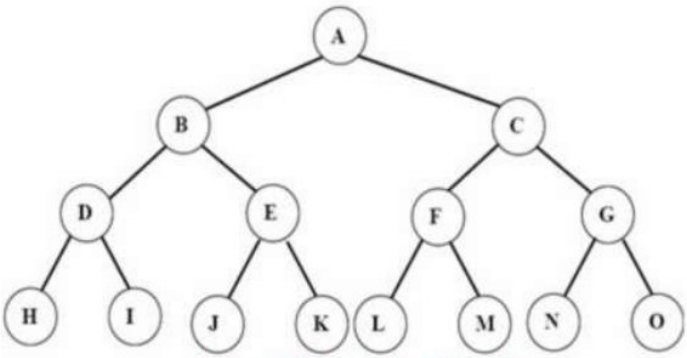
The Postfix of
(a+b/c*(d*e) is
a b c / d e * * +

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.



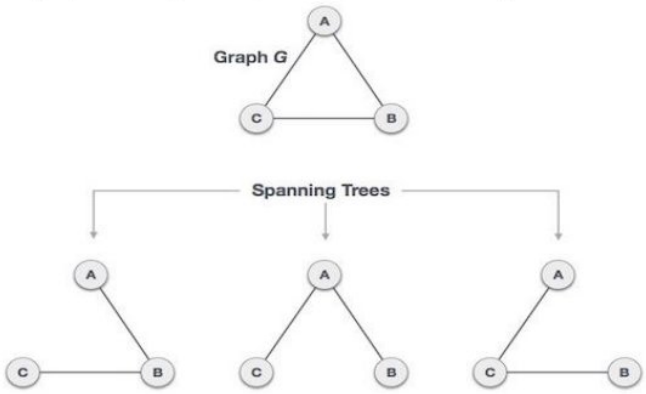
A complete Binary Tree of depth 3

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.

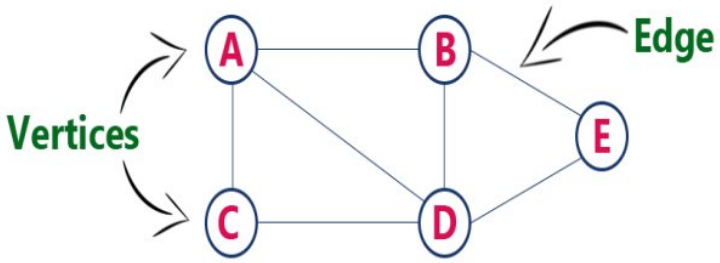
xiii. What is a spanning tree?

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.



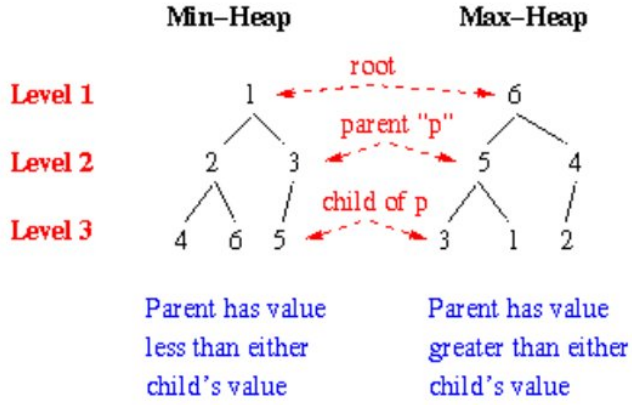
xiv. What is a graph?

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);
 }
}
```

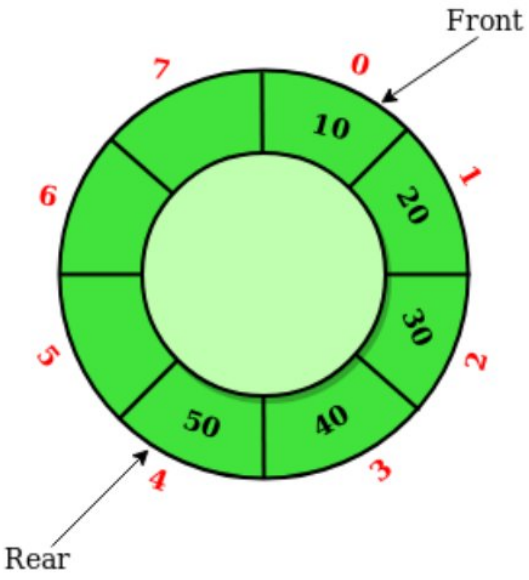
- **The output is:**

```
[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.

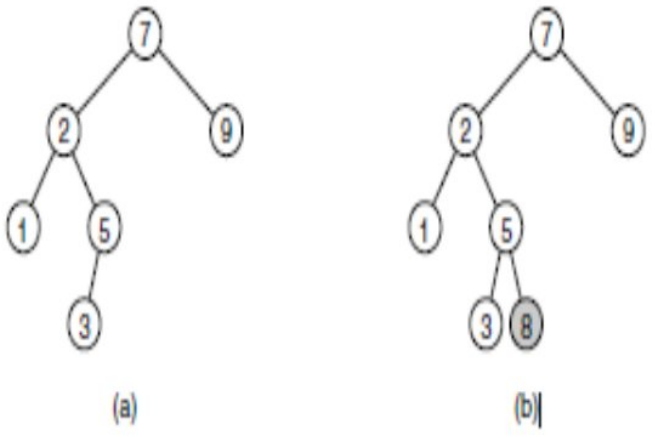
45,32,90,21,78,65,87,132,90,96,41,74,92

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.

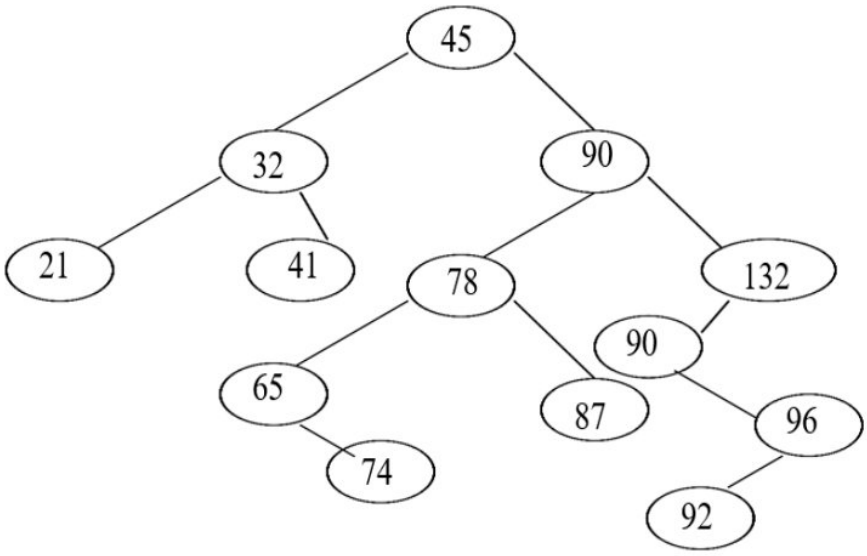
The binary search tree, a simple data structure that can be viewed as extending the binary search algorithm to allow insertions and deletions. The running time for most operations is O(log N) on average. Unfortunately, the worst-case time is O(N) per operation.

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



BST for following sequence

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

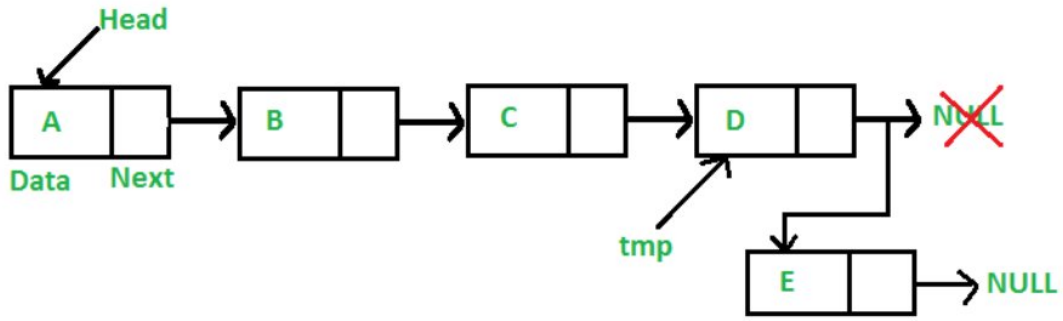


Q 5. Write a Program/Algorithm to do the following operations

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.

University of Sargodha

BS 3rd Term Examination 2016

Subject: I. T Paper: Data Structure & Algorithm (CMP-3112)

Time Allowed: 2:30 Hours Maximum Marks: 80

Objective Part (Compulsory)

Q.No.1. Write short answers of the following questions in 2-4 lines only. (16x2=32)

1. What is a linked-list?
2. What is stack?
3. What operations can be performed on Queues?
4. Why we need to do algorithm analysis?
5. What is binary search?
6. What is selection sort?
7. What is a graph?
8. What is a tree?
9. What is shell sort?
10. How breadth first traversal works?
11. What is a heap in data structure?
12. What is a recursive function?
13. What is tower of hanoi?
14. What is fibonacci series?
15. What is adjacency list?
16. What is hashing?

Subjective Part

Note: Attempt any four questions. (4x12=48)

Q.No.2. The following algorithm is supposed to compute the product of the elements of its input array. Prove that the algorithm is correct.

```
Input: Array A[1,..,n], n >= 1
Output: Product of the array's elements

PROD(A)
1.   tulo = 1
2.   i = 1
3.   while i<=n
4.       tulo = tulo*A[i]
5.       i = i+1
6.   return tulo
```

Q.No.3. Depict binary trees with heights 2, 3, 4, 5, and 6. All should be depicted with the following seven keys 1,4,5,10,16,17, and 21.

Q.No.4. Given an array A={12, 11, 13, 5, 6}. Sort it out using a technique illustrated in insertion sort. You have to discuss only the passes in detail and there is no need to write an algorithm of insertion sort.

Q.No.5. Algorithms A and B sort their input arrays. Algorithm A performs $32 \times n\lg(n)$ operations and algorithm B performs $3 \times n^2$ operations, when the array is of size n. Figure out, when to use algorithm A and when to use algorithm B if the size of the array is known.

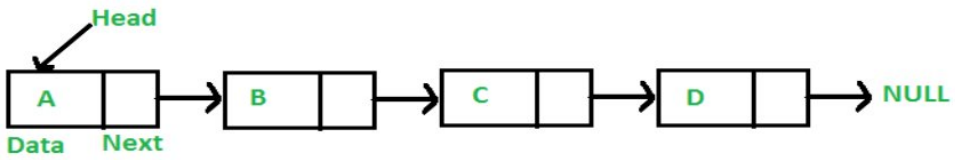
Q.No.6. Write a program to insert or delete item from a circular queue.

OBJECTIVE

Q 1: Short Questions Answers

i. What is a inked-list?

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.



Arrays can be used to store linear data of similar types, but arrays have following limitations.
1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

Advantages of Linked lists over arrays

1) Dynamic size

2) Ease of insertion/deletion

ii. What is Stack?

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**.

iii. What operations can be performed on Queues?

- **Traversal**- This is the process of accessing each and every element of the queue.
- **Enque-** This is an operation where you can add an element to the end of the queue.
- **Deque-** This is an operation where you can delete an element from the starting of the queue.
- **Searching-** You can search through the queue for an element of your choice and display it's position in the queue.
- **Merging-** You can merge two queues together, one behind the other.

iv. Why we need to do algorithm analysis?

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following –

A Priori Analysis This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.

A Posterior Analysis – This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

We shall learn about a priori algorithm analysis. Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

v. What is binary search?

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[]. A simple approach is to do **search. The** time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

vi. What is selection sort?

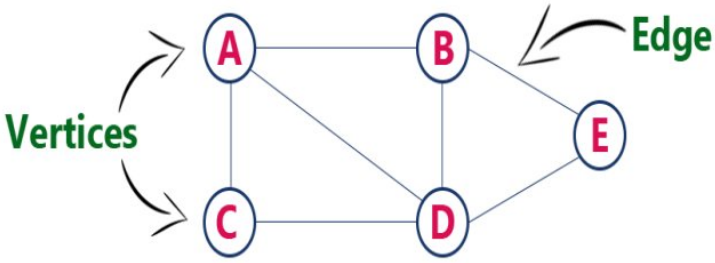
The list is divided into two sub lists, sorted and unsorted, which are divided by an imaginary wall. We find the smallest element from the unsorted sub list and swap it with the element at the beginning of the unsorted data.

After each selection and swapping, the imaginary wall between the two sub lists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.

Each time we move one element from the unsorted **sub list** to the sorted **sub list**, we say that we have completed a sort pass (round). A list of ***n*** elements requires ***n-1*** passes to completely rearrange the data.

vii. What is a graph?

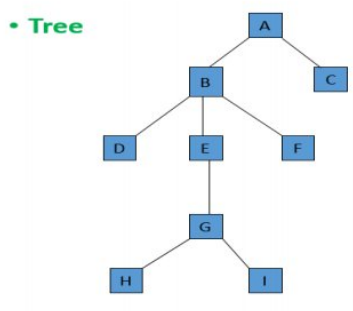
A **graph** is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.



viii. What is a Tree?

Trees are non-linear data structures. Trees are mainly used to represent data containing a **hierarchical relationship** between elements, for example, records, family trees and table of contents. Consider a parent-child relationship.

A tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship. Tree is a sequence of nodes. There is a starting node known as a root node. Every node other than the root has a parent node. Nodes may have any number of children.



ix. What is Shell Sort?

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval.

Running time analysis:

Best case: O(N logn)

Worst case: O(N2)

x. How breadth first traversal works?

Breadth First search (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a "**search** key"') and explores the neighbor nodes **first**, before moving to the next level neighbors. The level-by-level traversal is called a ***breadth-first***

traversal because we explore the *breadth*, i.e., full width of the tree at a given level, before going *deeper*.

xi. What is a heap in data structures?

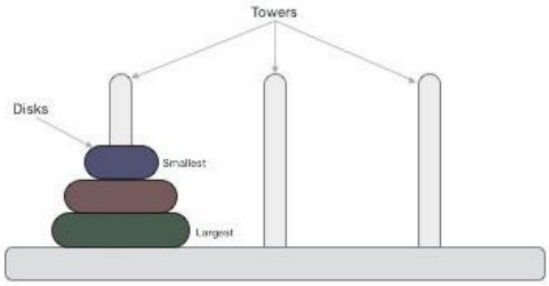
In computer science, a heap is a specialized tree-based data structure that satisfies the heap property: if P is a parent node of C, then the key (the value) of P is either greater than or equal to (in a ***max heap***) or less than or equal to (in a ***min heap***) the key of C. The node at the "***top***" of the heap (with no parents) is called the ***root*** node.

xii. What is a recursive function?

A **recursive function** (DEF) is a **function** which either calls itself or is in a potential cycle of **function** calls. As the definition specifies, there are two types of **recursive functions**. Consider a **function** which calls itself: we call this type of **recursion** immediate **recursion.**

xiii. What is tower of Hanoi?

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted. These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.



The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

xiv. What is Fibonacci series?

Fibonacci series generates the subsequent number by adding two previous numbers.
Fibonacci series starts from two numbers − **F0 & F1**. The initial values of **F0 & F1** can be taken **0, 1** or **1, 1** respectively.
Fibonacci series satisfies the following conditions −

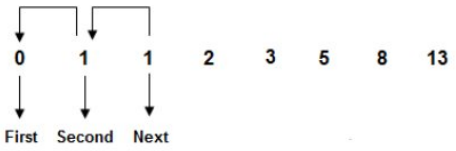
Fn = Fn-1 + Fn-2

Hence, a Fibonacci series can look like this −

F8 = 0 1 1 2 3 5 8 13

or, this −

F8 = 1 1 2 3 5 8 13 21




```
next = first + second;
first = second;
second = next;
```

xv. What is adjacency list?

Graph is a data structure that consists of following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph (di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.
Following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List

Adjacency List:
An array of linked lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists.

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:

Q.No.2. The following algorithm is supposed to compute the product of the elements of its input array. Prove that the algorithm is correct.

```
Input: Array A[1,..,n], n >= 1
Output: Product of the array's elements

PROD(A)
1.   tulo = 1
2.   i = 1
3.   while i<=n
4.        tulo = tulo*A[i]
5.        i = i+1
6.   return tulo
```

➢ **Total Correctness of an Algorithm** depends on the following conditions.

i. Correct input data is the data which satisfies the initial condition of the speciation.
ii. Correct output data is the data which satisfies the final condition of the speciation.

✓ **Dentition**

An algorithm is called **totally correct** for the given speciation if and only if for any correct input data it:

a) stops and
b) returns correct output

➢ **Partial Correctness of Algorithm**

Usually, while checking the correctness of an algorithm it is easier to separately:

i. First check whether the algorithm stops
ii. then checking the "remaining part". This "remaining part" of correctness is called **Partial Correctness** of algorithm.

✓ **Dentition**

An algorithm is partially correct if satisfies the following condition: If the algorithm receiving correct input data stops then its result is correct.

❖ The Given Algorithm is correct as it satisfied all the stages/steps of Algorithms.

- Variables are declared and initialized
- The condition **While i<=n** justify the working to find the output i.e Product of Array's elements.
- The formula is correct according to the condition

Tulo=tulo*A[i]

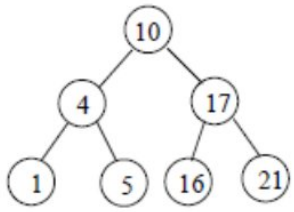
Here A[i] give the elements of Array names **A**, and the next statement i=i+ 1 provides the increment in the array's indexes.

and loop will run **n** times. And at the end it'll return **tulo** the final product of array's elements.

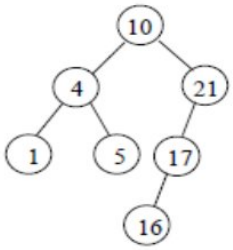
Q 3: Depict binary tree with heights 2,3,4,5 and 6. All should be depicted with the following seven keys 1, 4,5,10,16,17,21.

For Key values 1, 4,5,10,16,17,21 the binary trees of heights 2,3,4,5 and 6 are following

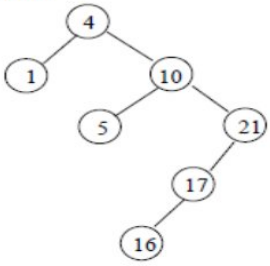
Height 2



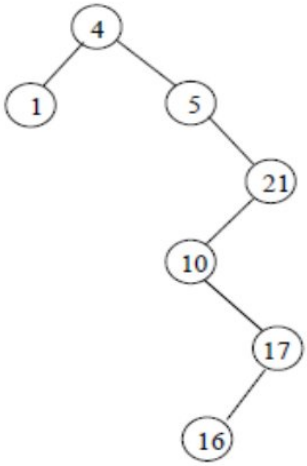
Height 3



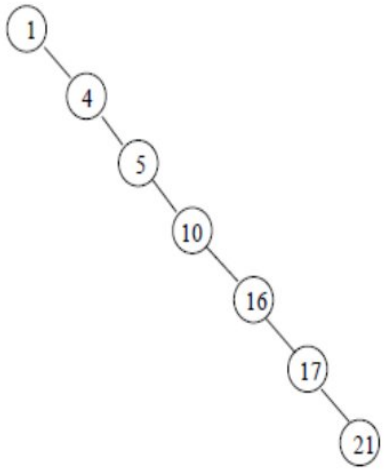
Height 4



height 5



Height 6



Q 4: Given an Array A= {12, 11, 13, 5, 6}. Sort it out using a technique illustrated in insertion sort. You have to discuss only the passes in details and there is no need to write an algorithm of insertion sort.

In insertion sort we have two group of items:

i. sorted group, and
ii. unsorted group

Initially, all items in the unsorted group and the sorted group is empty. We assume that items in the unsorted group unsorted. We have to keep items in the sorted group sorted. Pick any item from unsorted, then insert the item at the right position in the sorted group to maintain sorted property. Repeat the process until the unsorted group becomes empty.

Original Array: {12, 11, 13, 5, 6}

After Pass 1: {11, 12, 13, 5, 6}

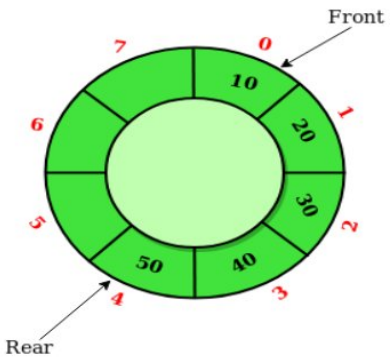
After Pass 2: {11, 12, 13, 5, 6}

After Pass 3: {5, 11, 12, 13, 6}

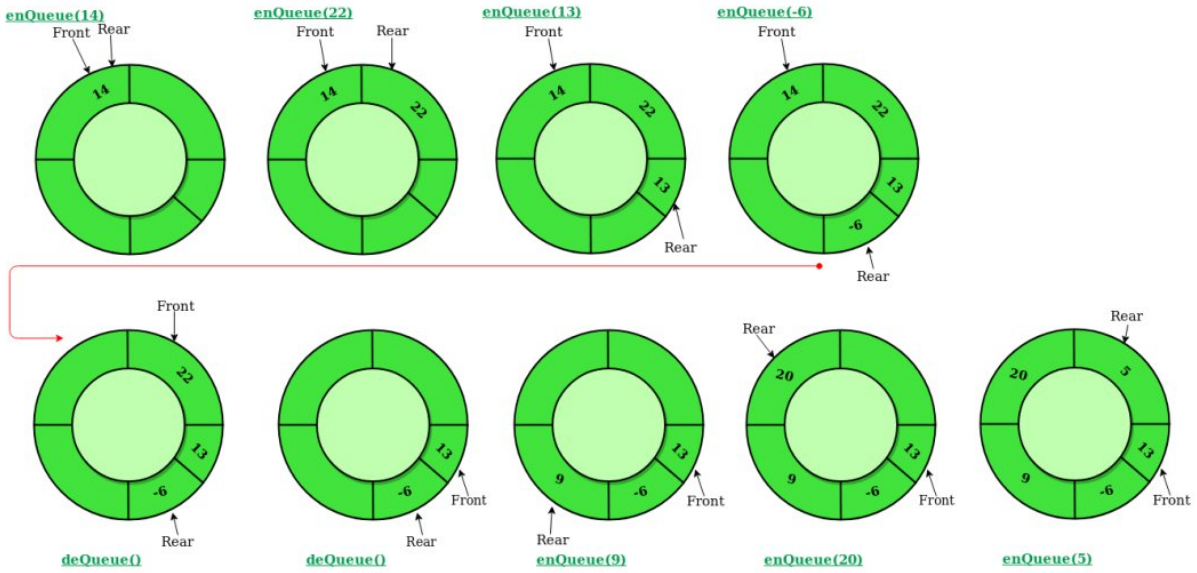
After Pass 4: {5, 6, 11, 12, 13}

Q 6: write a program to insert or delete item from circular queue.

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'**.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue.




```
//Circular Queue implementation using Array.
class CircularQueue {
int maxSize;
int rear = -1;
int front = -1;
int arr[];

public CircularQueue(int n) {

maxSize = n;
arr = new int[maxSize];

}

public boolean isEmpty() {
if (rear == -1 && front == -1) {
return true;
}
return false;

}
```

```
// Inserting element at rear end in Circularqueue.
public void enQueue(int item) {

if (((rear + 1) % maxSize) == front) {
System.out.println("Queue is overflow ");
return;
}
else if (isEmpty()) {
front = rear = 0;
}
else {
rear = (rear + 1) % maxSize;
}

arr[rear] = item;

System.out.println(item + " is inserted at position " + rear);

}

// Deleting element in CircularQueue from front end.
public void deQueue() {
int pos;
int item;
if (isEmpty()) {
System.out.println("Queue is under flow");
return;
}
else if (rear == front) {
pos = front;
item = arr[front];
rear = front = -1;
}
else {
pos = front;
item = arr[front];
front = (front + 1) % maxSize;
}
System.out.println(item + " is removed from position " + pos);

}
```