**Orchid International College**

**Bijay chowk, Gaushala**

**Department Of Information Technology**

****

**LAB MANUAL**

**of**

**Data Structure and Algorithm**

**Laboratory Works:**

* Array and linked list implementation of List
* Stack operations and queue operations
* Recursion
* Linked list implementation of stack and queue
* Binary search tree
* Graph
* Sorting, searching, and hashing algorithm

**LAB-1:** **Linked Lists**

**Introduction**

A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the data and the address of the next node. A big benefit of using linked lists is that nodes are stored wherever there is free space in memory, the nodes do not have to be stored contiguously right after each other like elements are stored in arrays. Linked lists can be of multiple types: **singly**, **doubly**, and **circular linked lists**.

**Objectives:**

1. Understand the basic concepts of linked lists:

2. Implement linked list operations

3. Manipulate data within linked lists

4. Understand memory management

1. **A Java program to demonstrate a Singly linked list.**

Source code:

class SLLNode {

int info;

SLLNode next;

}

class SLL {

SLLNode first, last;

public void insertFirst(int data) {

SLLNode newnode = new SLLNode();

newnode.info = data;

if (first == null) {

first = newnode;

last = newnode;

} else {

newnode.next = first;

first = newnode;

}

}

public void deleteFirst() {

if (first == null) {

System.out.println("Empty list");

} else if (first == last) {

first = null;

last = null;

} else {

first = first.next;

}

}

public void deleteLast() {

if (first == null) {

System.out.println("Empty list");

} else if (first == last) {

first = null;

last = null;

} else {

SLLNode temp = first;

while (temp.next != last) {

temp = temp.next;

}

temp.next = null;

last = temp;

}

}

public void insertAt(int data, int pos) {

SLLNode newnode = new SLLNode();

newnode.info = data;

if (first == null) {

first = newnode;

last = newnode;

} else {

//find the previous node of given position

SLLNode temp = first;

for (int i = 1; i < pos - 1; i++) {

temp = temp.next;

}

newnode.next = temp.next;

temp.next = newnode;

}

}

public void search(int data) {

int x = 0;

boolean isfound = false;

if (first == null) {

System.out.println("empty line");

} else {

SLLNode temp = first;

while (temp != null) {

x++;

if (temp.info == data) {

System.out.println("search succesful");

isfound=true;

System.out.println("Data found at node:" + x);

break;

}

temp = temp.next;

}if(isfound == false) {

System.out.println("Data doesnot exist");

}

}

}

public void display() {

if (first == null) {

System.out.println("Empty list");

} else {

SLLNode temp = first;

while (temp != null) {

System.out.print(temp.info + " ");

temp = temp.next;

}

System.out.println();

}

}

}

public class SinglylinkedList {

public static void main(String[] args) {

SLL mylist = new SLL();

mylist.insertFirst(5);

mylist.insertFirst(6);

mylist.display();

mylist.deleteFirst();

mylist.display();

mylist.deleteLast();

mylist.display();

mylist.insertAt(50, 2);

mylist.display();

mylist.search(5);

mylist.display();

}

}

2. **A Java program to demonstrate a circular linked list**

Source code:

class CLLNode {

int info;

CLLNode next;

}

class CLL {

CLLNode first, last;

public void insertLast(int data) {

CLLNode newnode = new CLLNode();

newnode.info = data;

if (last == null) {

first = newnode;

last = newnode;

last.next = first;

} else {

last.next = newnode;

last = newnode;

last.next = first;

}

}

public void deleteFirst() {

if (first == null) {

System.out.println("Empty list");

} else if (first == last) {

first = null;

last = null;

} else {

first = first.next;

last.next = first;

}

}

public void deleteLast() {

if (last == null) {

System.out.println("Empty list");

} else if (last == first) {

last = null;

first = null;

} else {

CLLNode temp = first;

while (temp.next != last) {

temp = temp.next;

}

last.next = null;

temp.next = first;

last = temp;

}

}

public void insertAt(int data, int pos) {

CLLNode newnode = new CLLNode();

newnode.info = data;

if (first == null) {

first = newnode;

last = newnode;

} else {

CLLNode temp = first;

for(int i=1;i<pos-1;i++){

temp=temp.next;

}

newnode.next = temp.next;

temp.next = newnode;

}

}

public void deleteAt(int pos) {

if (first == null) {

System.out.println("Empty list");

} else if (first == last) {

first = null;

last = null;

} else {

CLLNode temp = first;

for(int i=1;i<pos-1;i++){

temp=temp.next;

}

CLLNode cur = temp.next;

temp.next = cur.next;

cur.next = null;

}

}

public void display() {

if (first == null) {

System.out.print("Empty list");

} else {

CLLNode temp = first;

while (temp != last) {

System.out.print(temp.info + " ");

temp = temp.next;

}

System.out.println(temp.info);

}

}

}

public class CircurlaryLinkedList {

public static void main(String[] args) {

CLL myList = new CLL();

myList.insertLast(8);

myList.insertLast(9);

myList.display();

myList.deleteFirst();

myList.display();

myList.deleteLast();

myList.display();

myList.insertAt(45, 2);

myList.display();

myList.deleteAt(3);

myList.display();

}

}

**Tasks:**

1.write a java function to insert node at the end of singly linked list.

2. write a java function to delete from a specific position of singly linked list.

3. write a java function to insert a node at the beginning of circular linked list.

4. write a java function to delete the first node of a doubly linked list.

5.write a java function to insert at specific position of doubly linked list.

**LAB-2: Stack**

**Introduction**

A Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.

**Objective**

1. Implement Stack Using Arrays and Linked Lists

2. Perform Basic Stack Operations (push, pop, peek)

3. Handle Edge Cases (underflow and overflow)

1. **A program to demonstrate stack using Array.**

Source code:

interface stack {

public void push(int data);

public int pop();

public int peek();

public void view();

}

class Mystack implements stack {

int TOP = -1, STACKSIZE = 5;

int stack[] = new int[STACKSIZE];

public void push(int data) {

if (TOP == STACKSIZE - 1) {

System.out.println("stack overflow");

} else {

TOP++;

stack[TOP] = data;

}

}

public int pop() {

if (TOP == -1) {

System.out.println("stack underflow");

return 0;

} else {

int item = stack[TOP];

TOP--;

return item;

}

}

public int peek() {

if (TOP == -1) {

System.out.println("stack underflow");

return 0;

} else {

int item = stack[TOP];

return item;

}

}

public void view() {

if (TOP == -1) {

System.out.println("stack Empty");

} else {

for (int i = TOP; i >= 0; i--) {

System.out.println(stack[i]);

}

}

}

}

public class StackUsingArray {

public static void main(String[] args) {

Mystack ob=new Mystack();

ob.push(5);

ob.push(10);

ob.push(15);

ob.push(20);

ob.push(30);

ob.push(40);

ob.push(3);

ob.view();

System.out.println("Item Removed:"+ob.pop());

ob.view();

ob.push(50);

ob.view();

System.out.println("Item at top:"+ob.peek());

}

}

2. **A program to demonstrate the postfix.**

Source code:

package stack;

import java.util.\*;

public class PostfixEvaluation {

public static void main(String[] args) {

String exp = "6 2 3 + - 3 8 2 / + \* 2 $ 3 +";

Scanner sc = new Scanner(exp);

java.util.Stack<Integer> opndstk = new java.util.Stack();

while (sc.hasNext()) {

if (sc.hasNextInt()) {

opndstk.push(sc.nextInt());

} else {

int op2 = opndstk.pop();

int op1 = opndstk.pop();

String operator = sc.next();

switch (operator) {

case "+":

int value = op1 + op2;

opndstk.push(value);

break;

case "-":

value = op1 - op2;

opndstk.push(value);

break;

case "\*":

value = op1 \* op2;

opndstk.push(value);

break;

case "/":

value = op1 / op2;

opndstk.push(value);

break;

case "$":

value = (int) Math.pow(op1, op2);

opndstk.push(value);

}

}

}

System.out.println("Result=" + opndstk.pop());

}

}

Tasks:

1. write a complete program to show the push, pop and view operation in a stack using array implementation.

2. write a program to show the prefix evaluation.

**LAB-3: Queues**

**Introduction**

A queue is a linear data structure that follows a specific order in which operations are performed. The order is First In First Out (FIFO), which means that the element added to the queue first will be the one to be removed first. Queues are widely used in various applications such as scheduling processes, handling web server requests, and managing operating system tasks.

**Objectives**

1. Understand the Queue Data Structure

2. Implement Queue Using Arrays and Linked Lists

3. Perform Basic Queue Operations (enqueue, dequeue, front)

4. Handle Edge Cases (underflow and overflow)

1. **A program to demonstrate a simple queue using an array.**

Source code:

import java.util.\*;

interface Queue {

void enqueue(int data);

void dequeue();

void view();

}

class MyQueue implements Queue {

int QSIZE = 5;

int queue[] = new int[QSIZE];

int REAR = -1, FRONT = -1;

public void enqueue(int data) {

if (REAR == QSIZE - 1) {

System.out.println("Queue Overflow");

} else if (REAR == -1) {

REAR++;

queue[REAR] = data;

FRONT = 0;

} else {

REAR++;

queue[REAR] = data;

}

}

public void dequeue() {

if (FRONT == -1) {

System.out.println("Queue Underflow");

} else {

System.out.println("Item deleted" + queue[FRONT]);

queue[FRONT] = 0;//optional

FRONT++;

}

}

public void view() {

if (FRONT == -1) {

System.out.println("Queue Empty");

} else {

for (int i = FRONT; i <= REAR; i++) {

System.out.print(queue[i] + " ");

}

System.out.println();

}

}

}

public class SimpleQueueUsingArray {

public static void main(String[] args) {

MyQueue ob=new MyQueue();

ob.enqueue(8);

ob.enqueue(10);

ob.enqueue(7);

ob.view();

ob.dequeue();

ob.view();

}

}

2.**A program to demonstrate a priority ascending queue.**

Source code:

interface Queue {

void enqueue(int data);

void dequeue();

void view();

}

class MyPriorityQueue implements Queue {

int QSIZE = 5;

int queue[] = new int[QSIZE];

int REAR = -1, FRONT = -1;

public void enqueue(int data) {

if (REAR == QSIZE - 1) {

System.out.println("Queue Overflow");

} else if (REAR == -1) {

REAR++;

queue[REAR] = data;

FRONT = 0;

} else {

REAR++;

queue[REAR] = data;

}

}

public void dequeue() {

if (FRONT == -1) {

System.out.println("Queue Underflow");

} else {//find the smallest value and its index

int small = queue[FRONT];

int index=FRONT;

for (int i = FRONT; i <= REAR; i++) {

if (queue[i] < small) {

small = queue[i];

index = i;

}

}

System.out.println("Item deleted:" + small);

//shift the elements one step back

for (int i =index;i <=REAR; i++) {

queue[i] = queue[i+1];

}

REAR--;

}

}

public void view() {

if (FRONT == -1) {

System.out.println("Queue Empty");

} else {

for (int i = FRONT; i <= REAR; i++) {

System.out.print(queue[i] + " ");

}

System.out.println();

}

}

}

public class PriorityAscendingQueue {

public static void main(String[] args) {

MyPriorityQueue ob=new MyPriorityQueue();

ob.enqueue(8);

ob.enqueue(10);

ob.enqueue(7);

ob.view();

ob.dequeue();

ob.view();

}

}

**Task:**

1. write a program to show simple queue using linked list.

2. write a java function to enqueue, dequeue and view using circular queue.

**LAB-4: Recursion**

**Introduction**

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.

**objective**

1. understand the Concept of Recursion

2. Implement Recursive Functions

3. Solve Mathematical Problems Using Recursion (factorial,Fibonacci sequence)

4. Recognize and Use Tail Recursion for Optimization

1. **A program to print the first 10 Fibonacci number**

Source code:

public class Fibonacii {

public static void main(String[] args) {

//to print the first 10 Fibonacci number

for(int i=1;i<=10;i++){

System.out.println(fibo(i));

}

}

static int fibo(int n)

{

if(n==0)

{

return 0;

} else if(n==1)

return 1;

else

return fibo(n-1)+fibo(n-2);

}

}

2. **To find HCF (gcd) of two number**

Source code:

public class HCF {

public static void main(String[] args) {

System.out.println(gcd(20, 25));

}

static int gcd(int n1, int n2) {

if (n2 == 0) {

return n1;

} else if (n1 == 0) {

return n2;

} else {

return gcd(n2, n1 % n2);

}

}

}

3**. A program to demonstrate the Tower of Hanoi**

Source code:

public class TowerOfHanoi {

public static void main(String[] args) {

TOH(3,'A','B','C');

}

public static void TOH(int N,char A,char B, char C){

if(N>0){

TOH(N-1,A,C,B);

System.out.println(A+"->"+C);

TOH(N-1,B,A,C);

}

}

}

**Tasks:**

1. write a java program to find the factorial of a number using Recursive function.

2. write a java program to convert decimal to binary using recursive function.

3. write a java program to display the series of the 10th term.

4. write a java program to find the sum of natural numbers.

**LAB-5: Trees**

**Introduction**

**Tree data structure** is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

**Objectives:**

1. Understand the Binary Search Tree (BST)

2. Implement BST Operations

3. Explore Tree Traversal Algorithms

1.**A program to demonstrate the binary search tree.**

Source code:

class BSTNode {

int data;

BSTNode left, right;

BSTNode(int data) {

this.data = data;

}

}

class BST {

BSTNode root;

public BSTNode Search(BSTNode root, int key) {

if (root == null) {

return null;

} else if (key < root.data) {

return Search(root.left, key);

} else {

return Search(root.right, key);

}

}

public BSTNode insert(BSTNode node, int data) {

if (node == null) {

node = new BSTNode(data);

} else if (data < node.data) {

node.left = insert(node.left, data);

} else {

node.right = insert(node.right, data);

}

return node;

}

public void insert(int data) {

root = insert(root, data);

}

public void preorder(BSTNode root) {

if (root == null) {

return;

}

System.out.println(root.data + " ");

preorder(root.left);

preorder(root.right);

}

public void preorder() {

preorder(root);

}

}

public class BinarySearchTreeExample {

public static void main(String[] args) {

BST bst = new BST();

bst.insert(10);

bst.insert(5);

bst.insert(20);

bst.preorder();

}

}

**Tasks:**

1.write a java function for post order traversal in a BST.

2. write a java function for in order traversal in a BST.

**LAB-6: Graph**

**Introduction**

Graph Data Structure is a collection of nodes connected by edges. It’s used to represent relationships between different entities. Graph algorithms are methods used to manipulate and analyze graphs, solving various problems like finding the shortest path or detecting cycles.

**Objectives**

Understand the Graph Data Structure and Graph Traversal Techniques (Breadth-First Search, Depth-First Search)

1. **A program to demonstrate BFS traversal.**

Source code:

package Graph;  
 import java.io.\*;  
import java.util.\*;  
public class BFSTraversal{  
private int node; /\* total number number of nodes in the graph  
h \*/  
private LinkedList<Integer> adj[]; /\* adjacency list \*/  
private Queue<Integer> que; /\* maintaining a queue \*/  
BFSTraversal(int v){  
node = v;  
adj = new LinkedList[node];  
for (int i=0; i<v; i++){  
adj[i] = new LinkedList<>();  
}  
que = new LinkedList<Integer>();  
}  
void insertEdge(int v,int w){  
adj[v].add(w); /\* adding an edge to the adjacency list (edges are bidirectional in this example) \*/  
}  
void BFS(int n){  
boolean nodes[] = new boolean[node]; /\* initialize boolean array for holding the data \*/  
int a = 0;  
nodes[n]=true;  
que.add(n); /\* root node is added to the top of the queue \*/  
while (que.size() != 0){  
n = que.poll(); /\* remove the top element of the queue \*/  
System.out.print(n+" "); /\* print the top element of the queue \*/  
for (int i = 0; i < adj[n].size(); i++){ /\* iterate through the linked list and push all neighbors into queue  
\*/  
a = adj[n].get(i);  
if (!nodes[a]){ /\* only insert nodes into queue if they have not been explored already \*/  
nodes[a] = true;  
que.add(a);  
      }  
   }  
 }  
}  
public static void main(String args[]){  
BFSTraversal graph = new BFSTraversal(6);  
graph.insertEdge(0, 1);  
graph.insertEdge(0, 3);  
graph.insertEdge(0, 4);  
graph.insertEdge(4, 5);  
graph.insertEdge(3, 5);  
graph.insertEdge(1, 2);  
graph.insertEdge(1, 0);  
graph.insertEdge(2, 1);  
graph.insertEdge(4, 1);  
graph.insertEdge(3, 1);  
graph.insertEdge(5, 4);  
graph.insertEdge(5, 3);  
System.out.println("Breadth First Traversal for the graph is:");  
graph.BFS(0);  
  }  
}

**Tasks**

1. write a Java program for traversing a graph by implementing the DFS algorithm

**LAB-7: Sorting**

**Introduction:**

Sorting in Data Structures and Algorithms (DSA) refers to the process of arranging elements in a specific order, typically numerical or lexicographical, to facilitate efficient searching, merging, and other operations. Sorting plays a crucial role in various applications, from simple data organization to complex algorithm design.

**Objectives:**

1. Identify Different Types of Sorting Algorithms

2. Implement and Analyze Bubble Sort, Selection Sort, and Insertion Sort

3. Implement and Analyze Merge Sort and Quick Sort

1. **A Java function to demonstrate merge sort algorithm.**

Source code:

package sorting;

public class Mergesort {

static void merge(int L[], int R[], int A[]) {

int nL = L.length, nR = R.length;

int i = 0, j = 0, k = 0;

//merge

while (i < nL && j < nR) {

if (L[i] <= R[j]) {

A[k++] = L[i++];

} else {

A[k++] = R[j++];

}

}

//put the remaning elements on left array

while (i < nL) {

A[k++] = L[i++];

}

//put the remaning elements on right array

while (j < nR) {

A[k++] = R[j++];

}

}

static void mergeSort(int A[]) {

int n = A.length;

if (n < 2)//single element

{

return;

}

int mid = n / 2;

int Left[] = new int[mid];

int Right[] = new int[n - mid];

for(int i = 0; i < mid; i++) {

Left[i] = A[i];

}

for (int i = mid; i < n; i++) {

Right[i - mid] = A[i];

}

mergeSort(Left);

mergeSort(Right);

merge(Left, Right, A);

}

public static void main(String[] args) {

int A[] = {10, 5, 4, 9, 7, 6, 1, 3, 2};

mergeSort(A);

System.out.println("Array after sorting");

for (int m : A) {

System.out.print(m + " ");

}

}

}

2. **A Java function to demonstrate a bubble sort algorithm.**

Source code:

public class Bubblesort {

static void bubblesort(int a[]) {

int n = a.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (a[j] > a[j + 1]) {

int temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

public static void main(String[] args) {

int a[] = {5, 8, 4, 7, 3};

bubblesort(a);

System.out.println("After sorting");

for (int i = 0; i < a.length; i++) {

System.out.println(a[i]);

}

}

}

Tasks:

1. write a Java program to sort the data using the selection sort algorithm.

2. write a Java program to sort the data using a quick sort algorithm.

3. write a Java program to sort the data using the merge sort algorithm.

**LAB-8: searching and Hashing**

**Introduction**

**Hashing** is a technique that enables efficient storage, retrieval, and deletion of data using a data structure called a hash table. It involves mapping keys to indexes in the table using a hash function, allowing for constant average-time operations.

**Searching** refers to the process of finding a specific item or element within a data structure. It involves various algorithms such as linear search, binary search (for sorted arrays), and hash-based search techniques (using hash tables).

**Objectives**

1. Understand Searching Techniques

2. Implement Linear Search and Binary Search Algorithms

3. Implement Hash Tables and Understand Hashing Techniques

1. **A Java function to demonstrate a binary search algorithm.**

Source code:

import java.util.\*;

public class BinarySearch {

static int binarysearch(int array[],int key,int low,int high){

//reapeat until the pointer low and high meet each other

while(low<=high){

int mid=low +(high-low)/2;

if(array[mid]==key)

return mid;

if(array[mid]<key)

low=mid+1;

else

high=mid-1;

}

return -1;//not found

}

public static void main(String[] args) {

int A[]={1,2,4,5,6,7,8};

Scanner sc=new Scanner(System.in);

System.out.println("Enter value to search");

int key=sc.nextInt();

int result=binarysearch(A,key,0,A.length-1);

if(result==-1)

System.out.println("search unsuccesful");

else

System.out.println("Item found at index"+result);

}

}

2. **A Java function to demonstrate Linear probing.**

Source code:

package Hashing;

public class LinearProbing {

static int a[]=new int[10];

static int hashfunction(int n){

int key=n%10;

//check if the key is already occupied

if(a[key]==0)

return key;

else{

//find the key from key plus 1 to last of array

for(int i=key+1;i<a.length;i++){

if(a[i]==0){

key=i;

break;

}

}

//find the key from 0 to key minus 1

for(int i=0;i<key;i++){

if(a[i]==0){

key=i;

break;

}

}

}

return key;

}

public static void main(String[] args) {

int key;

key=hashfunction(89);

a[key]=89;

key=hashfunction(18);

a[key]=18;

key=hashfunction(49);

a[key]=49;

key=hashfunction(58);

a[key]=58;

key=hashfunction(69);

a[key]=69;

for(int i=0;i<a.length;i++){

System.out.print(a[i]+" ");

}

}

}

**Tasks:**

1. write a java function to implement a hash table using quadratic probing.

2. write a java function to implement linear search.