1)Stack, Queue, Singly Linked List, Doubly Linked List Implementation.

import java.util.Scanner;

// Stack Implementation

class Stack {

private int[] stack;

private int top;

private int size;

Stack(int size) {

this.size = size;

stack = new int[size];

top = -1;

}

void push(int data) {

if (top == size - 1) {

System.out.println("Stack Overflow!");

} else {

stack[++top] = data;

System.out.println(data + " pushed to stack.");

}

}

void pop() {

if (top == -1) {

System.out.println("Stack Underflow!");

} else {

System.out.println(stack[top--] + " popped from stack.");

}

}

void display() {

if (top == -1) {

System.out.println("Stack is empty.");

} else {

System.out.print("Stack elements: ");

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

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

}

System.out.println();

}

}

}

// Queue Implementation

class Queue {

private int[] queue;

private int front, rear, size, count;

Queue(int size) {

this.size = size;

queue = new int[size];

front = 0;

rear = -1;

count = 0;

}

void enqueue(int data) {

if (count == size) {

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

} else {

rear = (rear + 1) % size;

queue[rear] = data;

count++;

System.out.println(data + " added to queue.");

}

}

void dequeue() {

if (count == 0) {

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

} else {

System.out.println(queue[front] + " removed from queue.");

front = (front + 1) % size;

count--;

}

}

void display() {

if (count == 0) {

System.out.println("Queue is empty.");

} else {

System.out.print("Queue elements: ");

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

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

}

System.out.println();

}

}

}

// Singly Linked List Implementation

class SinglyLinkedList {

class Node {

int data;

Node next;

Node(int data) { this.data = data; }

}

private Node head = null;

void insertEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = newNode;

}

System.out.println(data + " inserted at end.");

}

void delete(int data) {

if (head == null) {

System.out.println("List is empty.");

return;

}

if (head.data == data) {

head = head.next;

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

return;

}

Node temp = head;

while (temp.next != null && temp.next.data != data)

temp = temp.next;

if (temp.next == null)

System.out.println("Element not found.");

else {

temp.next = temp.next.next;

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

}

}

void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node temp = head;

System.out.print("Linked List: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

// Doubly Linked List Implementation

class DoublyLinkedList {

class Node {

int data;

Node prev, next;

Node(int data) { this.data = data; }

}

private Node head = null;

void insertEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = newNode;

newNode.prev = temp;

}

System.out.println(data + " inserted at end.");

}

void delete(int data) {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node temp = head;

while (temp != null && temp.data != data)

temp = temp.next;

if (temp == null) {

System.out.println("Element not found.");

} else {

if (temp.prev != null) temp.prev.next = temp.next;

if (temp.next != null) temp.next.prev = temp.prev;

if (temp == head) head = temp.next;

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

}

}

void displayForward() {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node temp = head;

System.out.print("Forward: ");

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

void displayBackward() {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node temp = head;

while (temp.next != null)

temp = temp.next;

System.out.print("Backward: ");

while (temp != null) {

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

temp = temp.prev;

}

System.out.println();

}

}

// Main Application

public class DataStructureApp {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Stack stack = new Stack(5);

Queue queue = new Queue(5);

SinglyLinkedList sll = new SinglyLinkedList();

DoublyLinkedList dll = new DoublyLinkedList();

while (true) {

System.out.println("\n--- Data Structure Menu ---");

System.out.println("1. Stack");

System.out.println("2. Queue");

System.out.println("3. Singly Linked List");

System.out.println("4. Doubly Linked List");

System.out.println("5. Exit");

System.out.print("Enter choice: ");

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("1.Push 2.Pop 3.Display");

int ch1 = sc.nextInt();

if (ch1 == 1) stack.push(sc.nextInt());

else if (ch1 == 2) stack.pop();

else stack.display();

break;

case 2:

System.out.println("1.Enqueue 2.Dequeue 3.Display");

int ch2 = sc.nextInt();

if (ch2 == 1) queue.enqueue(sc.nextInt());

else if (ch2 == 2) queue.dequeue();

else queue.display();

break;

case 3:

System.out.println("1.Insert 2.Delete 3.Display");

int ch3 = sc.nextInt();

if (ch3 == 1) sll.insertEnd(sc.nextInt());

else if (ch3 == 2) sll.delete(sc.nextInt());

else sll.display();

break;

case 4:

System.out.println("1.Insert 2.Delete 3.Display Forward 4.Display Backward");

int ch4 = sc.nextInt();

if (ch4 == 1) dll.insertEnd(sc.nextInt());

else if (ch4 == 2) dll.delete(sc.nextInt());

else if (ch4 == 3) dll.displayForward();

else dll.displayBackward();

break;

case 5:

System.out.println("Exiting...");

sc.close();

return;

default:

System.out.println("Invalid choice.");

} } }}

2)Binary Search using recursion.

import java.util.Scanner;

public class BinarySearchRecursion {

// Recursive function for binary search

static int binarySearch(int arr[], int low, int high, int key) {

if (low > high)

return -1;

int mid = (low + high) / 2;

if (arr[mid] == key)

return mid; else if (arr[mid] > key)

return binarySearch(arr, low, mid - 1, key); // Search left half

else

return binarySearch(arr, mid + 1, high, key); // Search right half

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter " + n + " sorted elements:");

for (int i = 0; i < n; i++)

arr[i] = sc.nextInt();

System.out.print("Enter element to search: ");

int key = sc.nextInt();

int result = binarySearch(arr, 0, n - 1, key);

if (result == -1)

System.out.println("Element not found.");

else

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

sc.close();

}

}