DSA LAB Report By Sunil Neupane

1. Linear Search Algorithm

```
import java.util.Scanner;
class LinearSearch {
    public static boolean linearsearch(int arr[],int key){
        boolean found=false;
        for(int i=0;i<arr.length;i++){
            if(arr[i]==key){
                found=true;
                     break;
            }
        }
        return found;
    }
    public static void main(String[] args) {
        int arr[]={45,46,78,96,99};
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Search Key ...");
        int key=sc.nextInt();
        if(linearsearch(arr, key)){
            System.out.println("Element Found....");
        }
        else{
            System.out.println("Element not Found...");
        }
    }
}
/*
OUTPUT:
Enter Search Key...
78
Element Found...
*/
```

PROF

2. Binary Search Algorithm

```
import java.util.Scanner;


public class BinarySearch {
        public static Integer binarySearch(int arr[],int key){
        int lo=0,mid,hi=arr.length-1;
        while (lo<=hi) {
            mid=(lo+hi)/2;
```

```
            if(key<arr[mid]){
                hi=mid+1;
            }
            else if (arr[mid]<key) {
                lo=mid-1;
            }
            else{
                return mid;
            }

        }
        return null;
    }

    public static void main(String[] args) {
     Scanner sc=new Scanner(System.in);
     int arr[]={12,13,18,22,48,68};
     System.out.println("Enter Search Key");
     int key=sc.nextInt();
     Integer found= binarySearch(arr, key);
     if(found==null){
        System.out.println("Element not found");
     }
     else{
        System.out.println("Element Found..");
     }
}
}

/*
OUTPUT
Enter Search Key
13
Element Found
*/
```

PROF

3.Operation on Singly LinkedList

```
/*
 * Singly Linkedlist code done by Sunil Neupanea
*/
class Node{
    int data;
    Node next;
    Node(int data){
        this.data=data;
        this.next=null;
    }
    Node(int data,Node next){
        this.data=data;
```

```
        this.next=next;
    }
}
class Slinkedlist{
    Node head,tail;
    public boolean isEmpty(){
        return head==null && tail==null;
    }
    public void insertromtail(int el){
        if(isEmpty()){
            head=tail=new Node(el);
        }
        else{
            tail=tail.next=new Node(el, null);
        }
    }
    public void insertromhead(int el){
        if(isEmpty()){
            head=tail=new Node(el);
        }
        else{
            head=new Node(el, head);
        }
    }
    public void Display(){
        Node temp=head;
        while (temp!=null) {
            System.out.print(temp.data+"-->>");
            temp=temp.next;
        }
    }
    public void deletefromhead(){
        if (isEmpty()) {
            System.out.println("Unable to delete from empty node ");
        }
        else if (head==tail) {
            head=tail=null;
        }
        else{
            head=head.next;
        }
    }
    public void deletefromtail(){
        Node temp=head;
        while (temp.next!=null) {

            temp=temp.next;
            temp.next=null;

        }
        tail=temp;
    }
}
```

PROF

```
class Singlylinklist {
    public static void main(String[] args) {

        Slinkedlist ss=new Slinkedlist();
        ss.insertromhead(45);
        ss.insertromhead(89);
        ss.insertromhead(56);
        ss.insertromhead(47);
        ss.insertromtail(12);
        ss.insertromtail(23);
        System.out.println("Before Deletion ");
        ss.Display();
        ss.deletefromhead();
        ss.deletefromhead();
        ss.deletefromtail();
        ss.deletefromtail();
        System.out.println("After Deletion");
        ss.Display();

    }
}
/*
OUTPUT :
Before Deletion
47
56
89
45
12
23
After Deletion
89
45
*/
```

PROF

4.Operation on doubly Linked List

```
/**
 * Doubly Linked list Code Written By Sunil Neupane
 */
class Node{
    int data;
    Node prev,next;
    Node(Node prev,int data,Node next){
        this.prev=prev;
        this.next=next;
        this.data=data;
    }
}
```

```
class Doubly{
    Node head,tail;
    public boolean isEmpty(){
        return head==null && tail==null;
    }
    public void  insertfromhead(int el){
       if(isEmpty()){
        head=tail=new Node(null, el, null);
       }
   else{
        head=head.prev=new Node(null, el, head);

 }
    }
    public void insertfromtail(int el){
        if (isEmpty()) {
            head=tail=new Node(null, el, null);

        }
        else{
            tail=tail.next=new Node(tail, el, null);

        }
    }
    public void Display(){
        Node temp=head;
        while(temp!=null){
            System.out.print(temp.data+" -->");
            temp=temp.next;
        }
    }
    public void Displayreverse(){
        Node temp=tail;
        while (temp!=null) {
            System.out.print(temp.data+"-->>");
            temp=temp.prev;
        }
    }
    public void deletefromhead(){
        if (isEmpty()) {
            System.out.println("Unable To delete from empty node ");
        }
        else if(head==tail){
          head=tail=null;
        }
        else{
            head=head.next;
            head.prev=null;
        }

    }
    public void deletefromtail(){
        if (isEmpty()) {
```

```
            System.out.println("Unable to delete from Empty node ");
        }
        else if (head==tail) {
            head=tail=null;
        }
        else{
            tail=tail.prev;
            tail.next=null;
        }
    }
}
class DoublyLinkedList {
    public static void main(String[] args) {
        Doubly dd=new Doubly();
        System.out.println(dd.isEmpty());
        dd.insertfromhead(78);
        dd.insertfromhead(56);
        dd.insertfromhead(23);
        dd.insertfromhead(56);
        dd.insertfromhead(82);
        dd.insertfromhead(26);
        dd.insertfromtail(85);
        dd.insertfromtail(17);
        dd.insertfromtail(45);
        dd.deletefromhead();
        dd.deletefromtail();
        dd.Display();
        System.out.println("-----------------");
        dd.Displayreverse();
    }
}
/*
OUTPUT :
true
82
56
23
56
78
85
17
-----------------
17
85
78
56
23
56
82
*/
```

5. Operation on circular singly Linked List

```
class Node{
    int data;
    Node next;
    Node(int data,Node next){
        this.data=data;
        this.next=next;
    }
}
class CircularDemo{
    Node head,tail;
    public boolean isEmpty(){
        return head==null && tail==null;
    }
    public void insertHead(int el){
       if (isEmpty()) {
        head=tail=new Node(el, null);
       }
       else{
        head=new Node(el, head);
       }
    }
    public void insertTail(int el){
       if (isEmpty()) {
        head=tail=new Node(el, null);
       }
       else{
        tail=tail.next=new Node(el, head);
       }
    }
    public void deleteHead(){
       if (isEmpty()) {
        System.out.println("Unable To delete from Empty Node");
       }
       else if (head==tail) {
        head=tail=null;

       }
       else{
        head=head.next;
        tail.next=head;
       }
    }
    public void deleteTail(){
       if (isEmpty()) {
        System.out.println("Unable TO delete from empty node ");
       }
       else if (head==tail) {
        head=tail=null;
       }else{
```

```
        Node temp=head;
        while (temp.next!=tail) {
            temp=temp.next;
        }
        tail=temp;
        temp.next=head;
       }
    }
    public void display(){
      Node temp=head;
      do{
      System.out.println(temp.data);
      temp=temp.next;
      }while(temp!=tail.next);
    }
}

class circularsingly {
    public static void main(String[] args) {
     CircularDemo cc=new CircularDemo();
     cc.insertHead(890);
     cc.insertHead(780);
     cc.insertTail(74);
     cc.insertHead(250);
     cc.insertTail(25);
     cc.insertTail(23);
     cc.deleteHead();
     cc.deleteTail();
     cc.display();
    }
}
/*
OUTPUT :
780
890
74
25
*/
```

PROF

6.Operation On Circular Doubly Linked List

```
class Node{
    int data;
    Node prev,next;
    Node(Node prev,int data,Node next){
        this.data=data;
        this.next=next;
        this.prev=prev;
    }
}
```

```
class CircularDoubbly{
   Node head,tail;
   public boolean isEmpty(){
    return head==null && tail==null;
   }
   public void insertHead(int el){
    if (isEmpty()) {
        head=tail=new Node(null, el, null);
    }
    else{
        head=new Node(tail, el, head);
        head.next.prev=head;
        tail.next=head;
    }
   }
   public void insertTail(int el){
    if(isEmpty()){
        head=tail=new Node(null, el, null);
    }
    else{
        tail=new Node(tail, el, head);
        tail.prev.next=tail;
        head.prev=tail;
    }
   }
   public void deleteHead(){
    if (isEmpty()) {
        System.out.println("Unable to deletr from empty node ");
    }
    else if (head==tail) {
        head=tail=null;
    }else{
      head=head.next;
      head.prev=null;
      tail.next=head;
    }
   }
   public void deleteTail(){
    if(isEmpty()){
        System.out.println("Unable To delete From empty Node ");

    }
    else if (head==tail) {
        head=tail=null;
    }
    else{
        tail=tail.prev;
        tail.next=null;
        head.prev=tail;
    }
   }
   public void PrintForward(){
    Node temp=head;
```

PROF

```
    while (temp!=tail.next) {
        System.out.println(temp.data);
        temp=temp.next;
    }
   }
   public void PrintBackwatd(){
    Node temp=tail;
    do{
     System.out.println(temp.data);
     temp=temp.prev;
    }while(temp!=head.prev);
   }
}

public class CircularDoubly {
     public static void main(String[] args) {
      CircularDoubbly dd=new CircularDoubbly();
          System.out.println(dd.isEmpty());
          dd.insertHead(78);
          dd.insertHead(56);
          dd.insertTail(250);
          dd.insertHead(5);
          dd.insertTail(1);
          dd.insertHead(60);
          dd.insertTail(30);
          dd.deleteTail();
          dd.deleteTail();
          System.out.println("Print forward");
          dd.PrintForward();
          System.out.println("Print Backward");
          dd.PrintBackwatd();
    }
}
/*
OUTPUT :
true
Print forward
60
5
56
78
250
Print Backward
250
78
56
5
60
*/
```

PROF

7.ArrayList operation In java

```
import java.util.ArrayList;;
public class Demo {
    public static void main(String[] args) {
        ArrayList<String> aa=new ArrayList<>();
        aa.add("Neupane");
        aa.add("Sunil");
        System.out.println(aa.size());
        System.out.println(aa.get(0));
        System.out.println(aa);

    }
}
/*
OUTPUT:
2
Neupane
[Neupane, Sunil]

*/
```

8. Built in Linkedlist Operation In Java

```
import java.util.LinkedList;
public class Buiktlinkist {
    public static void main(String[] args) {
        LinkedList<Integer> ll=new LinkedList<>();
        ll.add(45);
        ll.addFirst(89);
        ll.addLast(83);
        ll.add(0, 23);
        System.out.println(ll.isEmpty());
        System.out.println(ll.toString());
        ll.offerFirst(56);
        ll.offerLast(45);
        System.out.println(ll);
    }
}
/*
false
[23, 89, 45, 83]
[56, 23, 89, 45, 83, 45]
*/
```

PROF

9. Built-in Stack Operation in Java

```
class Builtinstack {
    public static void main(String[] args) {
        Stack<Integer> ss=new Stack<>();
        ss.push(78);
        ss.push(25);

        ss.push(14);
        ss.pop();
        System.out.println(ss.peek());
        System.out.println("Printing all Element");
        System.out.println(ss);

    }
}
/*
OUTPUT:
25
Printing all Element
[78, 25]
*/
```

Stack Using Linkedlist in Java

```
import java.util.LinkedList;
class Stack{
    LinkedList<Integer> ll=new LinkedList<>();
    public void push(Integer el){
        ll.addFirst(el);;
    }
    public void pop(){
        if(ll.isEmpty()){
            System.out.println("Stack Underflow");
        }
        else{
            ll.removeFirst();
        }
    }
    public Integer peek(){
        if(ll.isEmpty()){
            return null;
        }
        else{
            return ll.getFirst();
        }
    }
}
```

PROF

```
class StackUsingLinklist {
    public static void main(String[] args) {
        Stack ss=new Stack();
        ss.push(89);
        ss.push(23);
        ss.push(90);
        ss.push(24);
        ss.push(12);
        ss.pop();
        ss.pop();
        System.out.println(ss.peek());
    }
}
/*
OUTPUT:
90
*/
```

Stack implemantation Using Array in Java

```
    Integer arr[]=new Integer[12];
    int tos=-1;
    public boolean isEmpty(){
        if(tos==-1){
            return true;
        }
        else{
            return false;
        }
    }
    public void push(int el){
        if(tos==arr.length-1){
            System.out.println("Stack Overflow");
            return;
        }
        else{
            tos++;
            arr[tos]=el;
        }
    }
    public void pop(){
        if(tos==-1){
            System.out.println("Stack Underflow");
            return;
        }
        else{
            tos--;
        }
    }
    public void peek(){
```

PROF

```
        if(tos==-1){
            System.out.println("Non such element");
        }
        else{
            System.out.println(arr[tos]);
        }
    }
}
public class StackUsingArrayinjava {
public static void main(String[] args) {
    StackArr  stk=new StackArr();
    stk.push(78);
    stk.push(23);
    stk.push(89);
    stk.pop();
    stk.peek();
}
}
/*
OUTPUT :
23
*/
```