**1.STACK:**

**package** com.cdac;

**public** **class** Stack {

**private** **int** top;

**private** **int**[] arr;

**public** Stack() {

top = -1;

arr = **new** **int**[5];

}

**public** Stack(**int** size) {

top = -1;

arr = **new** **int**[size];

}

**public** **boolean** push(**int** element) {

**if** (isFull()) {

System.***out***.println("Stack is full. Cannot push an element.");

**return** **false**;

} **else** {

System.***out***.println("Pushing the element in the stack:");

arr[++top] = element;

**return** **true**;

}

}

**public** **int** pop() {

**if** (isEmpty()) {

**throw** **new** StackEmptyException();

} **else** {

System.***out***.println("Poping the element from the stack:");

**return** arr[top--];

}

}

**public** **int** peek() {

**if** (isEmpty()) {

**throw** **new** StackEmptyException();

} **else** {

**return** arr[top];

}

}

**public** **boolean** isFull() {

**return** (top == arr.length - 1);

}

**public** **boolean** isEmpty() {

**return** (top == -1);

}

}

---------------------------------------------------------------------------------------------

**package** com.cdac;

@SuppressWarnings("serial")

**public** **class** StackEmptyException **extends** RuntimeException {

**public** StackEmptyException()

{

**super**("Stack is empty");

}

**public** StackEmptyException(String msg)

{

**super**(msg);

}

}

------------------------------------------------------------------------------------------------------

**package** com.cdac;

**public** **class** StackDemo {

**public** **static** **void** main(String[] args) {

Stack st = **new** Stack(5);

System.***out***.println("Push:"+st.push(12));

System.***out***.println("Push:"+st.push(5));

System.***out***.println("Push:"+st.push(8));

System.***out***.println("Push:"+st.push(20));

System.***out***.println(st.pop());

System.***out***.println(st.pop());

st.push(30);

st.push(55);

st.push(81);

st.push(24);

//pop out all the element of the stack untill stack is empty

**while**(!st.isEmpty())

{

System.***out***.println(st.pop());

}

//the code below demonstrate exception when pop() method is envoke on an empty stack

**try**

{

System.***out***.println(st.pop());

}

**catch**(Exception ex)

{

System.***out***.println(ex.getMessage());

}

}

}

OUTPUT:

Pushing the element in the stack:

Push:true

Pushing the element in the stack:

Push:true

Pushing the element in the stack:

Push:true

Pushing the element in the stack:

Push:true

Poping the element from the stack:

20

Poping the element from the stack:

8

Pushing the element in the stack:

Pushing the element in the stack:

Pushing the element in the stack:

Stack is full. Cannot push an element.

Poping the element from the stack:

81

Poping the element from the stack:

55

Poping the element from the stack:

30

Poping the element from the stack:

5

Poping the element from the stack:

12

Stack is empty

----------------------------------------------------------------

2.Double Stack:

**public** **class** DoubleStack {

**private** **int** arr[];

**int** size;

**int** top1,top2;

**public** DoubleStack(**int** size)

{

**this**.size=size;

arr=**new** **int**[size];

top1=-1;

top2=size;

}

**boolean** isEmpty1()

{

**return** top1==-1;

}

**boolean** isEmpty2()

{

**return** top2==size; //20

}

**boolean** isFull()

{

**return** (top1+1)==top2;

}

**boolean** push1(**int** data)

{

**if**(isFull())

**return** **false**;

arr[++top1]=data;

**return** **true**;

}

**boolean** push2(**int** data)

{

**if**(isFull())

**return** **false**;

arr[--top2]=data;

**return** **true**;

}

**int** pop1()

{

**if**(isEmpty1())

**return** -999;

**return** arr[top1--];

}

**int** pop2()

{

**if**(isEmpty2())

**return** -999;

**return** arr[top2++];

}

**int** peek2()

{

**if**(isEmpty2())

**return** -999;

**return** arr[top2];

}

}

-------------------------------------------------

**public** **class** TestDoubleStack {

**public** **static** **void** main(String[] args) {

DoubleStack ds = **new** DoubleStack(5);

System.***out***.println("push1 :" + ds.push1(10));

System.***out***.println("push1 :" + ds.push1(20));

System.***out***.println("push1 :" + ds.push1(30));

System.***out***.println("push1 :" + ds.push1(40));

System.***out***.println("push1 :" + ds.push2(50));

System.***out***.println("push2: " + ds.push2(80));

// System.out.println("pop2 :"+ds.pop2());

System.***out***.println("peek2 :" + ds.peek2());

}

}

OUTPUT:

push1 :true

push1 :true

push1 :true

push1 :true

push1 :true

push2: false

peek2 :50

2.QUEUE:

**package** queue;

**class** QueueUsingArray {

**int** MAX = 5;

**int** front = 0;

**int** rear = -1;

**int** size = 0;

**int**[] a = **new** **int**[MAX];

**private** **boolean** isFull() {

**if**(size == MAX) {

**return** **true**;

}

**return** **false**;

}

**public** **void** insert(**int** val) {

**if**(isFull()) {

System.***out***.println("Queue is full. Remove some elements");

**return**;

}

rear = (rear + 1) % MAX;

size++;

a[rear] = val;

}

**public** **boolean** ifEmpty() {

**if**(size == 0) {

**return** **true**;

}

**return** **false**;

}

**public** **int** remove() {

**if**(ifEmpty()) {

System.***out***.println("Queue is empty. Nothing to remove");

**return** Integer.***MIN\_VALUE***;

}

front = front % MAX;

size--;

**return** a[front++];

}

**public** **int** getSize() {

**return** size;

}

**public** **int** getFront() {

**if**(ifEmpty()) {

System.***out***.println("Queue is Empty. Nothing at Front");

**return** Integer.***MIN\_VALUE***;

}

**return** a[front];

}

**public** **int** getRear() {

**if**(ifEmpty()) {

System.***out***.println("Queue is Empty. Nothing at Rear");

**return** Integer.***MIN\_VALUE***;

}

**return** a[rear];

}

}

**package** queue;

**public** **class** QueueUsingArrayDemo {

**public** **static** **void** main(String[] args) {

QueueUsingArray a = **new** QueueUsingArray();

a.insert(12);

a.insert(2);

a.insert(7);

a.insert(27);

a.insert(18);

a.insert(80);

System.***out***.println("Size: " + a.getSize());

System.***out***.println("Front: " + a.getFront());

System.***out***.println("Rear: " + a.getRear());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Size: " + a.getSize());

System.***out***.println("Front: " + a.getFront());

System.***out***.println("Rear: " + a.getRear());

a.insert(80);

System.***out***.println("Size: " + a.getSize());

System.***out***.println("Front: " + a.getFront());

System.***out***.println("Rear: " + a.getRear());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Removed Element: " + a.remove());

System.***out***.println("Removed Element: " + a.remove());

}

}

OUTPUT:

Queue is full. Remove some elements

Size: 5

Front: 12

Rear: 18

Removed Element: 12

Size: 4

Front: 2

Rear: 18

Size: 5

Front: 2

Rear: 80

Removed Element: 2

Removed Element: 7

Removed Element: 27

Removed Element: 18

Removed Element: 80

Queue is empty. Nothing to remove

Removed Element: -2147483648

--------------------------------------------------------

Circular Queue:

**package** queue;

**public** **class** QueueC {

**private** **int** front;

**private** **int** rear;

**private** **int** count;

**private** **int**[] arr;

**public** QueueC() {

front = 0;

rear = -1;

count = 0;

arr = **new** **int**[5];

}

**public** QueueC(**int** size) {

front = 0;

rear = -1;

count = 0;

arr = **new** **int**[5];

}

**public** **boolean** isFull() {

**return** (count == arr.length);

}

**public** **boolean** isEmpty() {

**return** (count == 0);

}

**public** **void** enqueue(**int** element) {

**if** (isFull()) {

System.***out***.println("Queue is full");

} **else** {

rear = (rear + 1) % arr.length;

arr[rear] = element;

count++;

}

}

**public** **int** dequeue() {

**if** (isEmpty()) {

**throw** **new** RuntimeException("Queue is Empty");

} **else** {

**int** temp = arr[front];

front = (front + 1) % arr.length;

count--;

**return** temp;

}

}

}

**package** queue;

**public** **class** QueueCDemo {

**public** **static** **void** main(String[] args) {

QueueC q = **new** QueueC();

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

q.enqueue(40);

q.enqueue(50);

q.enqueue(60);

q.enqueue(70);

q.enqueue(80);

**while** (!q.isEmpty()) {

System.***out***.println(q.dequeue());

}

}

}

OUTPUT:

10

20

Queue is full

30

40

50

60

70

Doubly CircularLinkedList:

**package** doublylinkedlist;

**public** **class** Node {

**private** **int** info;

**private** Node prev;

**private** Node next;

**public** Node() {

info = 0;

prev = next = **null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getPrev() {

**return** prev;

}

**public** **void** setPrev(Node prev) {

**this**.prev = prev;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

**package** doublylinkedlist;

**public** **class** DoublyLinkedList {

**private** Node head;

**private** Node tail;

**public** DoublyLinkedList() {

head = tail = **null**;

}

**public** **void** insertAtEnd(**int** element) {

Node temp = **new** Node();

temp.setInfo(element);

**if** (head == **null**) {

head = tail = temp;

} **else** {

tail.setNext(temp);

temp.setPrev(tail);

tail = temp;

}

}

**public** **void** insertAtFront(**int** element) {

Node temp = **new** Node();

temp.setInfo(element);

**if** (head == **null**) {

head = tail = temp;

} **else** {

temp.setNext(head);

head.setPrev(temp);

head = temp;

}

}

**public** **void** traverseForward() {

Node curr = head;

System.***out***.println("Traverse forward");

**while** (curr != **null**) {

System.***out***.println(curr.getInfo());

curr = curr.getNext();

}

}

**public** **void** traverseBackward() {

Node curr = tail;

System.***out***.println("Traverse backward");

**while** (curr != **null**) {

System.***out***.println(curr.getInfo());

curr = curr.getPrev();

}

}

**void** display() {

Node curr = head;

**while** (curr != **null**) {

System.***out***.println(curr.getInfo());

curr = curr.getNext();

}

}

}

**package** doublylinkedlist;

**public** **class** DoublyLLDemo {

**public** **static** **void** main(String[] args) {

DoublyLinkedList dlist=**new** DoublyLinkedList();

dlist.insertAtEnd(5);

dlist.insertAtEnd(15);

dlist.display();

}

}

OUTPUT:

5

15

Doubly Circular LinkedLIst:

**package** com.cdac;

**public** **class** DoublyCircularLinkedList {

**private** Node tail;

**public** DoublyCircularLinkedList() {

tail = **null**;

}

**public** **void** insertAtEnd(**int** element) {

Node temp = **new** Node();

temp.setInfo(element);

**if** (tail == **null**) {

tail = temp;

tail.setNext(tail);

tail.setPrev(tail);

} **else** {

// Set temp's next to head

temp.setNext(tail.getNext());

// set temp's prev to tail

temp.setPrev(tail);

// set head's prev to new last node(temp).

tail.getNext().setPrev(temp);

// set 2nd last node's (tail) next to last node (temp)

tail.setNext(temp);

tail = temp;

}

}

**public** **void** traverseForward() {

Node curr = tail.getNext();

System.***out***.println("Traverse forward");

**do** {

System.***out***.println(curr.getInfo());

curr = curr.getNext();

} **while** (curr != tail.getNext());

}

**public** **void** traverseBackward() {

Node curr = tail;

System.***out***.println("Traverse backward");

**do** {

System.***out***.println(curr.getInfo());

curr = curr.getPrev();

} **while** (curr != tail);

}

}

**package** com.cdac;

**public** **class** DCLLDemo {

**public** **static** **void** main(String[] args) {

DoublyCircularLinkedList dclink = **new** DoublyCircularLinkedList();

dclink.insertAtEnd(10);

dclink.insertAtEnd(20);

dclink.insertAtEnd(30);

dclink.insertAtEnd(40);

dclink.insertAtEnd(50);

dclink.insertAtEnd(60);

dclink.traverseForward();

dclink.traverseBackward();

}

}

OUTPUT:

Traverse forward

10

20

30

40

50

60

Traverse backward

60

50

40

30

20

10

**ASSIGNMENT NO-1 DAY-1**

/\*Assign a set of characters to a sting involving {([])}

\* this characters specify whether the brackets are match or no\*/

package ASS1;

import com.cdac.StackEmptyException;

public class Brackets

{

private char [] arr;

private int top;

public Brackets()

{

arr=new char[6];

top=-1;

}

public Brackets(int size)

{

arr=new char[size];

top=-1;

}

public void push(char brac)

{

if(isFull())

{

System.out.println("stack is full");

}

else

{

arr[++top]=brac;

}

}

public char pop()

{

if(isEmpty())

{

throw new StackEmptyException();

}

else

{

return arr[top--];

}

}

public int peek()

{

if (isEmpty())

{

throw new StackEmptyException();

}

else

{

return arr[top];

}

}

public boolean isFull()

{

return (top== arr.length-1);

}

public boolean isEmpty()

{

return( top== -1);

}

}

**package** ASS1;

**public** **class** BracketMatching {

**private** **char**[]arr;

Brackets st=**new** Brackets();

**public** BracketMatching()

{

arr=**new** **char**[6];

}

**public** **void** bracketMatching(String str)

{

**for**(**int** i=0;i<str.length();i++)

{

**if**(str.charAt(i)=='{'|| str.charAt(i)=='[' || str.charAt(i)=='(' )

{

st.push(str.charAt(i));

}

**else** **if**(!st.isEmpty() &&

((str.charAt(i)==']' && st.peek()=='[') ||

(str.charAt(i)=='}' && st.peek()=='{') ||

(str.charAt(i)==')' && st.peek()=='(' )))

{

st.pop();

}

**else**

{

st.push(str.charAt(i));

}

}

**if**(st.isEmpty())

{

System.***out***.println("Balanced");

}

**else**

System.***out***.println("Unbalanced");

}

}

**package** ASS1;

**import** java.util.Scanner;

**public** **class** BracketDemo {

**public** **static** **void** main(String[] args) {

BracketMatching b=**new** BracketMatching();

System.***out***.println("Enter String: ");

Scanner sc=**new** Scanner(System.***in***);

String str=sc.next();

b.bracketMatching(str);

}

}

OUTPUT:

Enter String:

(){}({})[]

Balanced

Enter String:

([](){}]

Unbalanced

Assignment-2 Day2

Q.1.Create an implementation of stack using singly linked list:

**package** com.cdac;

**public** **class** Node {

**private** **int** info;

**private** Node next;

**public** Node()

{

info = 0;

next = **null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

**package** com.cdac;

**public** **class** StackLL {

**private** Node top;

**public** StackLL()

{

top = **null**;

}

**public** **void** push(**int** element)

{

Node temp = **new** Node();

temp.setInfo(element);

**if** (top == **null**)

{

top = temp;

}

**else**

{

temp.setNext(top);

top = temp;

}

}

**public** **int** pop()

{

**if** (top == **null**)

**throw** **new** RuntimeException("Empty stack.");

**else**

{

Node temp = top;

top = top.getNext();

**return** temp.getInfo();

}

}

}

**package** com.cdac;

**public** **class** StackLLDemo {

**public** **static** **void** main(String[] args) {

StackLL st = **new** StackLL();

st.push(5);

st.push(10);

st.push(50);

st.push(15);

st.push(25);

**try** {

System.***out***.println(st.pop());

System.***out***.println(st.pop());

System.***out***.println(st.pop());

System.***out***.println(st.pop());

System.***out***.println(st.pop());

System.***out***.println(st.pop());

}

**catch**(Exception ex)

{

System.***out***.println(ex.getMessage());

}

}

}

OUTPUT:

25

15

50

10

5

Empty stack.

Q.2.Create an implementation of Queue using singly linked list:

**public** **class** QNode {

**private** **int** info;

**private** QNode next;

**public** QNode()

{

**this**.info = 0;

**this**.next = **null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** QNode getNext() {

**return** next;

}

**public** **void** setNext(QNode next) {

**this**.next = next;

}

}

**public** **class** QueueLL{

**private** QNode rear;

**private** QNode front;

**public** QueueLL() {

rear=front=**null**;

}

**public** **void** enqueue(**int** element)

{

QNode temp=**new** QNode();

temp.setInfo(element);

**if**(rear==**null**)

{

rear=front=temp;

}

**else**

{

QNode curr=rear;

**while**(curr.getNext()!=**null**)

{

curr=curr.getNext();

}

curr.setNext(temp);

}

}

**public** **int** dequeue()

{

**if**(front==**null**)

{

**throw** **new** RuntimeException("Queue is Empty..");

}

**else**

{

QNode temp=front;

front=front.getNext();

**return** temp.getInfo();

}

}

**public** **void** display()

{

QNode curr=front;

**while**(curr!=**null**)

{

System.***out***.println(curr.getInfo());

curr=curr.getNext();

}

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

QueueLL q = **new** QueueLL();

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(40);

**try** {

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

System.***out***.println(q.dequeue());

}**catch**(RuntimeException ex )

{

System.***out***.println(ex.getMessage());

}

}

}

Output:

10

20

30

40

Queue is Empty..

Assignment-3

Q1. Circular Queue using Linked List.

//Q1. Circular Queue using Linked List.

**package** Assignment3;

**public** **class** Node {

**private** **int** info;

**private** Node next;

**public** Node(**int** data)

{

info = 0;

next = **null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

**package** Assignment3;

**public** **class** CircularQueueLL {

**private** Node tail;

**public** CircularQueueLL() {

tail=**null**;

}

**public** **void** enqueue(**int** element) {

Node temp=**new** Node(element);

//Node temp=tail;

temp.setInfo(element);

**if**(tail==**null**) {

tail=temp;

tail.setNext(tail);

}

**else** {

temp.setNext(tail.getNext());

tail.setNext(temp);

tail=temp;

}

}

**public** **void** display()

{

**if**(tail!=**null**)

{

Node curr=tail.getNext();

**do** {

System.***out***.println(curr.getInfo());

curr =curr.getNext();

}**while**(curr!=tail.getNext());

}

**else** {

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

}

}

**public** **int** dequeue() {

Node curr=**null**;

**if**(tail==**null**) {

**throw** **new** RuntimeException("empty list");

}

**else** **if**(tail.getNext()==tail) {

curr=tail.getNext();

tail=**null**;

**return** curr.getInfo();

}

**else** {

curr=tail.getNext();

curr=curr.getNext();

tail.setNext(curr);

**return** curr.getInfo();

}

}

}

**package** Assignment3;

**public** **class** Tester {

/\* Tester of the program \*/

**public** **static** **void** main(String args[])

{

CircularQueueLL q = **new** CircularQueueLL();

System.***out***.println("Insert element in circular queue:");

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.dequeue();

q.display();

}

}

OUTPUT:

20

30

1

2

3

Assignment-4

//create a singly linkedlist of integers such that all elements are inserted in asending order....

**package** Assignment4;

**public** **class** Node {

**private** **int** info;

**private** Node next;

**public** Node()

{

info = 0;

next = **null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

//create a singly linkedlist of integers such that all elements are inserted in asending order....

**package** Assignment4;

**public** **class** SinglyLLStruct {

**private** Node head;

SinglyLLStruct() {

head = **null**;

}

**public** **void** insertInAscending(**int** element) {

Node temp = **new** Node();

temp.setInfo(element);

Node curr = **new** Node();

curr = head;

**if** ((curr == **null**) || curr.getInfo() >= temp.getInfo()) {

temp.setNext(curr);

head = temp;

} **else** {

curr = head;

**while** ((curr.getNext() != **null**) && ((curr.getNext().getInfo()) < (temp.getInfo()))) {

curr = curr.getNext();

}

temp.setNext(curr.getNext());

curr.setNext(temp);

}

}

**public** **void** display() {

Node curr = head;

**while** (curr != **null**) {

System.***out***.println(curr.getInfo());

curr = curr.getNext();

}

}

}

**package** Assignment4;

**public** **class** Demo {

**public** **static** **void** main(String[] args) {

SinglyLLStruct list=**new** SinglyLLStruct();

list.insertInAscending(20);

list.insertInAscending(5);

list.insertInAscending(10);

list.insertInAscending(30);

list.insertInAscending(25);

list.display();

}

}

OUTPUT:

5

10

20

25

30

Assignment-5

Q1.Consider a singly Linked List,rotate the list from K`th position from the last one

e.g k=2 10,20,30,40,50

after rotation output should be as follows

40,50,10,20,30..

**package** Assignment5;

**public** **class** Node {

**private** **int** info;

**private** Node next;

**public** Node()

{

info=0;

next=**null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

**package** Assignment5;

**public** **class** LinkedList {

**private** Node head;

**public** LinkedList()

{

head=**null**;

}

**public** **void** insertAtEnd(**int** element)

{

Node temp=**new** Node();

temp.setInfo(element);

**if**(head == **null**)

{

head=temp;

}

**else**

{

Node curr=head;

**while**(curr.getNext()!=**null**)

{

curr=curr.getNext();

}

curr.setNext(temp);//curr.next=temp

//stored the reference of temp in curr next

}

}

**public** **void** insertAfter(**int** element,**int** after)

{

Node temp=**new** Node();

temp.setInfo(element);

Node curr=head;

**while**((curr!=**null**) && (curr.getInfo()!=after))

curr=curr.getNext();

**if**(curr!=**null**)

{

temp.setNext(curr.getNext());

curr.setNext(temp);

}

**else**

{

System.***out***.println("node with element "+after+" does not exist");

}

}

**public** **void** rotateByPosition(**int** n)

{

Node curr=**null**;

Node cut=**null**;

**int** counter=1;

curr=cut=head;

**if**(head==**null** || head.getNext()==**null**)

{

System.***out***.println("List cannot be rotated....");

}

**else**

{

**while**(curr.getNext()!=**null**)

{

curr=curr.getNext();

counter++;

}

**for**(**int** i=1;i<counter-n;i++)

{

cut=cut.getNext();

}

**if**(n%counter!=0)

{

curr.setNext(head);

head=cut.getNext();

cut.setNext(**null**);

}

}

}

//traverse the linkedlist from first node still the end(null)

**void** display()

{

Node curr=head;

**while**(curr!=**null**)

{

System.***out***.println(curr.getInfo());

curr=curr.getNext();

}

}

}

**package** Assignment5;

**import** java.util.Scanner;

**public** **class** LinkedListDemo {

**public** **static** **void** main(String[] args)

{

LinkedList list=**new** LinkedList();

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.insertAtEnd(40);

list.insertAtEnd(50);

list.insertAtEnd(60);

list.insertAtEnd(70);

list.insertAtEnd(80);

list.insertAtEnd(90);

list.insertAtEnd(100);

//list.insertAfter(17, 8);

//list.insertAfter(17, 20);

System.***out***.println("After at end Insertion");

list.display();

Scanner sc=**new** Scanner(System.***in***);

System.***out***.print("Enter k: ");

**int** k=sc.nextInt();

list.rotateByPosition(k);

System.***out***.println("After Rotation by kth position..");

list.display();

}

}

OUTPUT:

After at end Insertion

10

20

30

40

50

60

70

80

90

100

Enter k: 4

After Rotation by kth position..

70

80

90

100

10

20

30

40

50

60

Assignment-6

Q.1.Write a Recursive method to return n`th term of Finonacci series.

1st term = 0

2nd term = 1

3rd = 1

4th = 2

3,5,

t = t + t

n n-1 n-2

fibo(5)

**package** recursion;

**public** **class** RecursionDemo {

**int** fact(**int** n)

{

**if**(n==0)

**return** 1;

**else**

**return** n\*fact(n-1);

}

**int** fibo(**int** n)

{

**if**(n<=2)

**return**(n-1);

**else**

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

}

**public** **void** printIntegers(**int** n)

{

**if**(n>0)

{

printIntegers(n-1);

System.***out***.println(n+" ");

}

}

//tower of hannai

**public** **void** toh(**int** n,String src,String tgt,String aux)

{

**if**(n==0)

**return**;

**else**

{

toh(n-1,src,aux,tgt);

System.***out***.println("Move disk "+n+" from "+src+" to "+tgt);

toh(n-1,aux,tgt,src);

}

}

**public** **static** **void** main(String[] args)

{

RecursionDemo obj=**new** RecursionDemo();

System.***out***.println("Factoial of given number:"+obj.fact(5));

System.***out***.println("First 10 series of fibonacci series..");

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

{

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

}

System.***out***.println("First 5 Integer..");

obj.printIntegers(5);

System.***out***.println("Move 3 disk");

obj.toh(3,"A","C","B");

}

}

OUTPUT:

Factoial of given number:120

First 10 series of fibonacci series..

0

1

1

2

3

5

8

13

21

34

First 5 Integer..

1

2

3

4

5

Move 3 disk

Move disk 1 from A to C

Move disk 2 from A to B

Move disk 1 from C to B

Move disk 3 from A to C

Move disk 1 from B to A

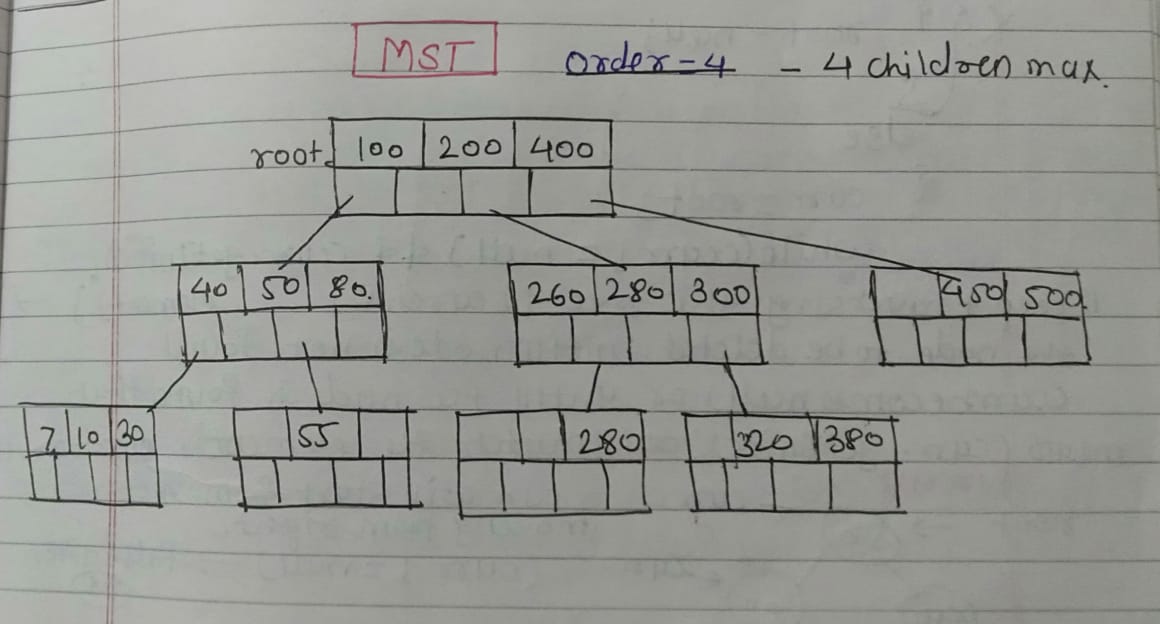
Move disk 2 from B to C

Move disk 1 from A to C

Assignment-6

Assign. Create a Multiway Search Tree of order 4 on paper if the elements arrive in the following order: Hint: Each node can have at most 3 values and 4 children

100,200,400,50,300,450,80,40,260,280,320,380,280,500,30,10,7,55



Assign. Create a BST and traverse it using BFT (Breadth First Traversal).

Hint: Use Queue of Node

Assign. Create a BST and traverse it using DFT (Depth First Traversal).

Hint: Use Stack of Node.

**package** BTS;

**public** **class** BinarySearchTree {

**private** Node root;

**public** BinarySearchTree() {

root = **null**;

}

**public** **void** insert(**int** element) {

Node temp = **new** Node();

temp.setInfo(element);

**if** (root == **null**) {

root = temp;

} **else** {

Node curr = root;

Node parent = **null**;

**while** (curr != **null**) {

parent = curr;

**if** (element <= curr.getInfo())

curr = curr.getLeft();

**else**

curr = curr.getRight();

}

// After the above loop completes execution, temp node will be

// made a child of parent node.

**if** (element <= parent.getInfo())

parent.setLeft(temp);

**else**

parent.setRight(temp);

}

}

**protected** **void** inorder(Node treeroot) {

**if** (treeroot != **null**) {

inorder(treeroot.getLeft());

System.***out***.print(treeroot.getInfo() + ", ");

inorder(treeroot.getRight());

}

}

// Overload the method preorder

**public** **void** inorder() {

inorder(root);

}

**protected** **void** preorder(Node treeroot) {

**if** (treeroot != **null**) {

System.***out***.print(treeroot.getInfo() + ", ");

preorder(treeroot.getLeft());

preorder(treeroot.getRight());

}

}

// Overload the method preorder

**public** **void** preorder() {

preorder(root);

}

**protected** **void** postorder(Node treeroot) {

**if** (treeroot != **null**) {

postorder(treeroot.getLeft());

postorder(treeroot.getRight());

System.***out***.print(treeroot.getInfo() + ", ");

}

}

// Overload the method postorder

**public** **void** postorder() {

postorder(root);

}

**public** **void** delete(**int** element) {

Node curr, parent = **null**;

**if** (root == **null**) {

System.***out***.println("\nTree is empty");

} **else** {

curr = root;

// Keep traversing the tree and search for the node

// to be deleted until node is not found (curr becomes null) or

// until the node is found (curr.getInfo() will match the element)

**while** ((curr != **null**) && (curr.getInfo() != element)) {

parent = curr;

**if** (element > curr.getInfo())

curr = curr.getRight();

**else**

curr = curr.getLeft();

}

// After the above loop

// If curr is null, element not found

// otherwise curr is the node to be deleted.

// parent is the parent node of curr.

// If node not found

**if** (curr == **null**) {

System.***out***.println("\nNode not present");

}

// Node to be removed is the root node

**else** **if** (parent == **null**) {

System.***out***.println("\nNode to be removed is the root node.");

Node temp=curr.getLeft();

//set temp to rightmost descendant of LST

**while**(temp.getRight()!=**null**)

temp=temp.getRight();

temp.setRight(curr.getRight());

curr.setRight(**null**);

root=curr.getLeft();

curr=**null**;

}

// Node to be removed is a leaf node

**else** **if** (curr.getLeft() == **null** && curr.getRight() == **null**) {

System.***out***.println("\nNode to be removed is a leaf node.");

// Check whether curr is the left child or right child of the parent.

// Set appropriate reference to null.

**if** (parent.getLeft() == curr)

parent.setLeft(**null**);

**else**

parent.setRight(**null**);

}

// Node to be removed has only 1 child

**else** **if** (curr.getLeft() == **null** || curr.getRight() == **null**) {

System.***out***.println("\nNode to be removed has only 1 child");

// check if curr is left child

**if** (curr.getLeft() != **null**) {

Node temp = curr.getLeft();

// if curr is left child of its parents

**if** (curr == parent.getLeft())

parent.setLeft(temp);

**else**

// Otherwise curr is right child of its parent

parent.setRight(temp);

}

// Otherwise curr has right child

**else** **if** (curr.getRight() != **null**) {

Node temp = curr.getRight();

**if** (curr == parent.getRight())

parent.setRight(temp);

**else**

parent.setLeft(temp);

}

} **else** {

System.***out***.println("\nNode to be removed has 2 children");

// set temp to curr s left node

Node temp = curr.getLeft();

// set temp to rightmost descendent of LST

**while** (temp.getRight() != **null**)

temp = temp.getRight();

temp.setRight(curr.getRight());// Rst is merged with Lst

curr.setRight(**null**);

// examin if curr is the right or left child of its parent

**if** (parent.getRight() == curr)

parent.setRight(curr.getLeft());

**else**

parent.setLeft(curr.getLeft());

curr.setLeft(**null**);

}

}

}

// Breath first travelsal

**public** **void** bft() {

Queue q = **new** Queue(10);

q.enqueue(root);

**while** (!q.isEmpty()) {

Node temp = q.dequeue();

**if** (temp.getLeft() != **null**)

q.enqueue(temp.getLeft());

**if** (temp.getRight() != **null**)

q.enqueue(temp.getRight());

System.***out***.print(temp.getInfo()+",");

}

}

**public** **void** dft() {

Stack s = **new** Stack();

s.push(root);

**while** (!s.isEmpty()) {

Node temp = s.pop();

System.***out***.print(temp.getInfo() + ",");

**if** (temp.getLeft() != **null**) {

s.push(temp.getLeft());

}

**if** (temp.getRight() != **null**) {

s.push(temp.getRight());

}

}

}

}

**package** BTS;

**public** **class** BSTDemo {

**public** **static** **void** main(String[] args) {

BinarySearchTree tree=**new** BinarySearchTree();

//System.out.println("Befor insert");

tree.insert(50);

tree.insert(60);

tree.insert(40);

tree.insert(35);

tree.insert(45);

tree.insert(55);

tree.insert(70);

tree.insert(52);

//tree.insert(80);

tree.insert(58);

tree.insert(56);

tree.insert(30);

System.***out***.println("Inorder Travelsal..");//Left-Node-Right

tree.inorder();

System.***out***.println("\nPreorder Travelsal..");//Node-Left-Right

tree.preorder();

System.***out***.println("\nPostorder Travelsal..");//Left-Right-Node

tree.postorder();

tree.delete(50);

tree.inorder();

System.***out***.println("\nBreath first Travelsal");

tree.bft();

System.***out***.println("\nDepth first Travelsal");

tree.dft();

}

}

OUTPUT:

Inorder Travelsal..

30, 35, 40, 45, 50, 52, 55, 56, 58, 60, 70,

Preorder Travelsal..

50, 40, 35, 30, 45, 60, 55, 52, 58, 56, 70,

Postorder Travelsal..

30, 35, 45, 40, 52, 56, 58, 55, 70, 60, 50,

Node to be removed is the root node.

30, 35, 40, 45, 52, 55, 56, 58, 60, 70,

Breath first Travelsal

40,35,45,30,60,55,70,52,58,56,

Depth first Travelsal

40,45,60,70,55,58,56,52,35,30,

Assign.Create an array of n elements in random order.

Write a program to search for an element in the array.

If the element is present, return the index of the element in the array, else return position

What is the time complexity in the above case??-->Time complexity is O(n) becouze single loop is used.for best case O(1) and for worst case O(n).....

**package** searching;

**import** java.util.Scanner;

**public** **class** LinearSearch {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("enter no of values:");

**int** n = sc.nextInt();

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

System.***out***.println("Enter" + n + "values:");

**for** (**int** i = 0; i < arr.length; i++)

arr[i] = sc.nextInt();

System.***out***.println("enter element you want to search:");

**int** target = sc.nextInt();

**int** position = -1;

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

**if** (arr[i] == target) {

position = i;

**break**;

}

}

**if** (position != -1)

System.***out***.println(target + "found! at position" + (position + 1));

**else**

System.***out***.println(target + "not found!");

}

}

OUTPUT:

enter no of values:

5

Enter 5 values:

11

22

33

33

33

enter element you want to search:

33

33 found! at position 3

2.Assign. Sort the array in the above assignment using Selection sort or Insertion sort, and then perform the search in the sorted array.

What is the time complexity now??

package insertionsort;

import java.util.Arrays;

import java.util.Scanner;

public class InsertionSort {

public static int[] insertionSort(int[]arr)

{

for(int i=1;i<arr.length;i++)

{

int j=i;

while((j>0) && (arr[j-1]>arr[j]))

{

int temp=arr[j];

arr[j]=arr[j-1];

arr[j-1]=temp;

j--;

}

}

return arr;

}

public static int[] Searching(int[]arr)

{

System.out.println("Aleready sorted array");

Scanner sc=new Scanner(System.in);

arr=insertionSort(arr);

//for (int i = 0; i < arr.length; i++)

//arr[i] = sc.nextInt();

System.out.println("enter element you want to search:");

int target = sc.nextInt();

int position = -1;

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

{

if (arr[i] == target)

{

position = i;

break;

}

}

if (position != -1)

System.out.println(target + " found! at position " + (position + 1));

else

System.out.println(target + " not found! ");

return arr;

}

public static void main(String[] args) {

int arr[]=new int[] {2,6,5,1,3,4};

arr=insertionSort(arr);

System.out.println(Arrays.toString(arr));

arr=Searching(arr);

}

}

OUTPUT:

[1, 2, 3, 4, 5, 6]

Aleready sorted array

enter element you want to search:

5

5 found! at position 5

Assign. To be done on paper only Consider the following equation

f(n) = 10n2 + 4n + 2 Find +ve integers c and n0 such that

f(n) <= c \* g(n)

g(n) = n2

Assign. Create a Singly linked list and implement all the following methods:

1) insertAtFront

2) insertAt End

3) insertBefore an element

4) insertAfter an element

5) insertAtPosition

6) Delete a specific node

7) Delete front most node

8) Delete last node

9) Delete node at specific position

10) Delete node after a specific element

11) Delete node before a specific element

**package** LLAssignment;

**public** **class** Node {

**private** **int** info;

**private** Node next;

**public** Node()

{

info=0;

next=**null**;

}

**public** **int** getInfo() {

**return** info;

}

**public** **void** setInfo(**int** info) {

**this**.info = info;

}

**public** Node getNext() {

**return** next;

}

**public** **void** setNext(Node next) {

**this**.next = next;

}

}

-------------------------------------------------------------------------------------

**package** LLAssignment;

**public** **class** LinkedList {

**private** Node head;

**public** LinkedList()

{

head = **null**;

}

//1)insert at front

**public** **void** insertAtFront(**int** element)

{

Node newNode = **new** Node();

newNode.setInfo(element);

**if**(head == **null**)

{

head = newNode;

}

**else**

{

newNode.setNext(head);

head = newNode;

}

}

//2)insert at end

**public** **void** insertAtEnd(**int** element)

{

Node newNode = **new** Node();

newNode.setInfo(element);

**if**(head == **null**)

{

head = newNode;

}

**else**

{

Node curr = head;

**while**(curr.getNext() != **null**)

{

curr = curr.getNext();

}

curr.setNext(newNode); //curr.next = newNode;

}

}

//3)insertBefore an element

**public** **void** insertBefore(**int** element, **int** before)

{

Node newNode = **new** Node();

newNode.setInfo(element);

Node curr = head;

Node prev = head;

**while**((curr != **null**) && (before != curr.getInfo()))

{

prev = curr;

curr = curr.getNext();

}

**if**(curr == head)

{

newNode.setNext(head);

head = newNode;

}

**else** **if**(curr == **null**)

{

**throw** **new** RuntimeException("Element not found.");

}

**else**

{

prev.setNext(newNode);

newNode.setNext(curr);

}

}

//4)insertAfter an element

**public** **void** insertAfter(**int** element, **int** after)

{

Node newNode = **new** Node();

newNode.setInfo(element);

Node curr = head;

**while**((curr != **null**) && (curr.getInfo() != after))

{

curr = curr.getNext();

}

**if**(curr != **null**)

{

newNode.setNext(curr.getNext());

curr.setNext(newNode);

}

**else**

{

System.***out***.println("\nNode with element "+after+" does not exist.");

}

}

//5)insert by position

@SuppressWarnings("unused")

**public** **boolean** insertByPos(**int** element, **int** pos)

{

**if**(pos <= 0)

{

**return** **false**;

}

Node newNode = **new** Node();

newNode.setInfo(element);

**if**(newNode == **null**)

{

System.***out***.println("Error in creation of node.");

**return** **false**;

}

//if position is 1 //insert at first

**if**(pos == 1)

{

newNode.setNext(head);

head = newNode;

**return** **true**;

}

//position is other than 1

//1. Locate node at pos - 1 i.e. prev

**else**

{

Node prev = head;

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

{

prev = prev.getNext();

**if**(prev == **null**)

{

**return** **false**;

}

}

newNode.setNext(prev.getNext());

prev.setNext(newNode);

**return** **true**;

}

}

//6)Delete a specific node

**public** **int** deleteByValue(**int** element)

{

**if**(head == **null**)

{

**throw** **new** RuntimeException("Empty List.");

}

//if element is present at head

**if**(head.getInfo() == element)

{

Node temp = head;

head = head.getNext();

**return** temp.getInfo();

}

//if element is not present at head

**else**

{

Node prev, del;

prev = del = head;

**while**(del.getInfo() != element)

{

prev = del;

del = del.getNext();

**if**(del == **null**)

**throw** **new** RuntimeException("Element is not present.");

}

prev.setNext(del.getNext());

**return** del.getInfo();

}

}

//7)Delete from front

**public** **int** deleteAtFront()

{

**if**(head == **null**)

{

**throw** **new** RuntimeException("Empty List.");

}

Node del = head;

head = head.getNext();

**return** del.getInfo();

}

//8)Delete at end

**public** **int** deleteAtEnd()

{

**if**(head == **null**)

{

**throw** **new** RuntimeException("Empty List.");

}

**else**

{

Node curr, prev;

curr = prev = head;

**while**(curr.getNext() != **null**)

{

prev = curr;

curr = curr.getNext();

}

prev.setNext(**null**);

**return** curr.getInfo();

}

}

//9)Delete by position

**public** **int** deleteByPos(**int** pos)

{

**if**(head == **null** || pos <=0)

{

**throw** **new** RuntimeException("Empty List OR Invalid Position.");

}

//if position is 1 //delete from front

**if**(pos == 1)

{

Node temp = head;

head = head.getNext();

**return** temp.getInfo();

}

//if position > 1

**else**

{

Node prev = head;

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

{

prev = prev.getNext();

**if**(prev == **null**)

{

System.***out***.println("Invalid Positon.");

}

}

Node del = prev.getNext();

**if**(del == **null**)

{

System.***out***.println("Element is not present.");

}

prev.setNext(del.getNext());

**return** del.getInfo();

}

}

//10)delete before

**public** **int** deleteBefore(**int** before)

{

**if**(head != **null**)

{

Node curr = head;

Node prev = head;

**while**(curr.getNext() != **null** && curr.getNext().getInfo() != before)

{

prev = curr;

curr = curr.getNext();

}

**if**(curr == head)

{

head = head.getNext();

}

**else**

{

prev.setNext(curr.getNext());

//curr.setNext(null);

}

**return** curr.getInfo();

}

**else**

{

**throw** **new** RuntimeException("Empty List.");

}

}

//11)delete after

**public** **int** deleteAfter(**int** after)

{

**if**(head != **null**)

{

Node curr = head;

//Node prev = head;

**while**(curr != **null** && curr.getInfo() != after)

{

curr = curr.getNext();

}

**if**(curr == head)

{

head = head.getNext();

}

**else**

{

curr.setNext(curr.getNext().getNext());

//curr.setNext(null);

}

//return curr.getInfo();

**return** (curr.getNext().getInfo());

}

**else**

{

**throw** **new** RuntimeException("Empty List.");

}

}

//traverse the linked list from first node till the end

**public** **void** display()

{

Node curr = head;

//System.out.print("Elements in linked list are : ");

**while**(curr != **null**)

{

System.***out***.print(curr.getInfo()+" ");

curr = curr.getNext();

}

}

}

----------------------------------------------------------------------------------------

**package** LLAssignment;

**public** **class** TestLinkedList {

**public** **static** **void** main(String[] args) {

LinkedList list = **new** LinkedList();

//list.insertAfter(20, 5); //Node with element 5 does not exist.

list.insertAtEnd(5);

list.insertAtEnd(15);

list.insertAtEnd(23);

list.insertAtEnd(8);

list.insertAtEnd(10);

list.insertAtEnd(12);

//list.insertAfter(25, 8);

//list.insertAfter(1, 30); //Node with element 30 does not exist // 30 is not in the list

System.***out***.print("Elements in linked list are : ");

list.display(); // 5 15 23 8 10 12

list.insertAfter(25, 8);

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Insert 25 after 8, elements in linked list are : ");

list.display(); // 5 15 23 8 25 10 12

list.insertAfter(1, 30); //Node with element 30 does not exist

list.insertAfter(17, 5);

System.***out***.print("Insert 17 after 5, elements in linked list are : ");

list.display(); // 5 17 15 23 8 25 10 12

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Insert 100 at front, elements in linked list are : ");

list.insertAtFront(100);

list.display();//100 5 17 15 23 8 25 10 12

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Insert 50 before 25, elements in linked list are : ");

list.insertBefore(50,25);

list.display();// 100 5 17 15 23 8 50 25 10 12

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Insert 75 at position 2, elements in linked list are : ");

list.insertByPos(75,2);

list.display();//100 75 5 17 15 23 8 50 25 10 12

System.***out***.println("\n\n\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Delete element 23, elements in linked list are : ");

list.deleteByValue(23);

list.display();//100 75 5 17 15 8 50 25 10 12

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Delete element (100) at front, elements in linked list are : ");

list.deleteAtFront();

list.display();//75 5 17 15 8 50 25 10 12

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Delete element(12) at end, elements in linked list are : ");

list.deleteAtEnd();

list.display();//75 5 17 15 8 50 25 10

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Delete element(15) before 8, elements in linked list are : ");

list.deleteBefore(8);

list.display();//75 5 17 8 50 25 10

System.***out***.println("\n--------------------------------------------------------------------------------------------------------");

System.***out***.print("Delete eleement(50) after 8, elements in linked list are : ");

list.deleteAfter(8);

list.display();

}

}

------------------------------------------------------------------------------------------

OUTPUT:

Elements in linked list are : 5 15 23 8 10 12

--------------------------------------------------------------------------------------------------------

Insert 25 after 8, elements in linked list are : 5 15 23 8 25 10 12

Node with element 30 does not exist.

Insert 17 after 5, elements in linked list are : 5 17 15 23 8 25 10 12

--------------------------------------------------------------------------------------------------------

Insert 100 at front, elements in linked list are : 100 5 17 15 23 8 25 10 12

--------------------------------------------------------------------------------------------------------

Insert 50 before 25, elements in linked list are : 100 5 17 15 23 8 50 25 10 12

--------------------------------------------------------------------------------------------------------

Insert 75 at position 2, elements in linked list are : 100 75 5 17 15 23 8 50 25 10 12

--------------------------------------------------------------------------------------------------------

Delete element 23, elements in linked list are : 100 75 5 17 15 8 50 25 10 12

--------------------------------------------------------------------------------------------------------

Delete element (100) at front, elements in linked list are : 75 5 17 15 8 50 25 10 12

--------------------------------------------------------------------------------------------------------

Delete element(12) at end, elements in linked list are : 75 5 17 15 8 50 25 10

--------------------------------------------------------------------------------------------------------

Delete element(15) before 8, elements in linked list are : 75 5 17 8 50 25 10

--------------------------------------------------------------------------------------------------------

Delete eleement(50) after 8, elements in linked list are : 75 5 17 8 25 10

ASSIGNMENT:  8

Assign. Implement Bubble sort

Assign. Implement Shell Sort

Assign. Implement Quick sort.

**1.Assign. Implement Bubble sort**

**package** bubblesort;

**public** **class** BubbleSort {

**public** **void** bubbleSort(**int** arr[])

{

**int** n=arr.length;

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

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

**if**(arr[j]>arr[j+1])

{

//swap temp and arr[i]

**int** temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

//display the array

**public** **void** display(**int** arr[])

{

**int** n=arr.length;

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

System.***out***.println(arr[i]+" ");

System.***out***.println();

}

}

**package** bubblesort;

**import** java.util.Scanner;

**public** **class** TestBubbleSort {

**public** **static** **void** main(String[] args) {

BubbleSort bs = **new** BubbleSort();

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter array size");

**int** n = sc.nextInt();

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

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

**for** (**int** i = 0; i < arr.length; i++)

arr[i] = sc.nextInt();

bs.bubbleSort(arr);

System.***out***.println("Sorted Array:");

bs.display(arr);

}

}

OUTPUT:

Enter array size

5

Enter 5 array elements:

33

22

66

56

78

Sorted Array:

22

33

56

66

78

**2.Assign. Implement Shell Sort**

**package** com.cdac;

**public** **class** ShellSort {

**public** **void** shellSorting(**int**[] arr) {

**int** i, k, h, hcnt, j, temp;

// An array to store shell values

**int**[] shells = **new** **int**[10];

// Create shells array

**for** (i = 0, k = 1; k < arr.length; i++) {

shells[i] = k;

k = 2 \* k + 1;

}

/\*

\* If array is of size 20, after the above loop shells[0] = 1, shells[1] = 3,

\* shells[2] = 7, shells[3] = 15 After the above loop ends, value of i will

\* become 4 shells[4] = 0. Hence decrement the value of i

\*/

i--;

**for** (; i >= 0; i--) // 1st consider shells[3], then shells[2], etc

{

h = shells[i];

// Loop on the sub-arrays

**for** (hcnt = h; hcnt < 2 \* h; hcnt++) {

// Apply insertion on every hth element of the sub-array

**for** (j = hcnt; j < arr.length; j += h) {

temp = arr[j];

k = j;

**while** ((k - h) >= 0 && temp < arr[k - h]) {

arr[k] = arr[k - h];

k = k - h;

}

arr[k] = temp;

}

}

}

}

}

**package** com.cdac;

**public** **class** ShellSortDemo {

**public** **static** **void** main(String[] args) {

// Created an array of size 20

**int**[] arr = {30,15,20,1,4,9,8,23,32,40,5,7,10,12,26,29,33,6,11,14};

ShellSort sort = **new** ShellSort();

sort.shellSorting(arr);

System.***out***.println("After applying shell sort algorithm");

**for** (**int** i=0;i<arr.length;i++)

{

System.***out***.print(arr[i] + " , ");

}

}

}

Output:

After applying shell sort algorithm

1 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 14 , 15 , 20 , 23 , 26 , 29 , 30 , 32 , 33 , 40 ,

**3.Assign. Implement Quick sort**

**package** com.cdac;

**public** **class** QuickSort {

**int** getPivot(**int**[] arr,**int** down,**int** up)

{

**int** low = down;

**int** high = up;

**int** pivotIdx = (low+high)/2;

**int** pivot = arr[pivotIdx]; // Current value of pivot

swap(arr,low,pivotIdx);

**while** (low < high)

{

**while** (arr[low]<=pivot && low < up)

low++;

**while** (arr[high]>pivot)

high--;

**if** (low < high)

swap(arr,low,high);

}

swap(arr,down,high);

**return** high;

}

// 5,7,20,15,1,10,8,12,3,4,9 down = 0 up = 10

// low = 0 high = 10

// pi = 5 pivot = 10

// swap on line 11 arr is 10,7,20,15,1,5,8,12,3,4,9

// low = 2

// high = 10

// 1st time line 19 swap

// 10,7,9,15,1,5,8,12,3,4,20

// Stops at low = 3 high = 9

// 10,7,9,4,1,5,8,12,3,15,20

// low = 7 high = 8

// 10,7,9,4,1,5,8,3,12,15,20

// low = 8 high = 7

// While loop ends down = 0 high = 7

// Swap arr[0] with arr[7]

// 3,7,9,4,1,5,8, 10, 12,15,20

**public** **void** quickSorting(**int**[] arr,**int** down,**int** up)

{

**int** pivotIndex;

**if**(down < up)

{

pivotIndex = getPivot(arr,down,up);

System.***out***.println("Pivot Value = " + arr[pivotIndex]);

quickSorting(arr, down, pivotIndex-1);

quickSorting(arr,pivotIndex+1,up);

}

}

**public** **void** swap(**int**[] arr,**int** x,**int** y)

{

**int** temp;

temp = arr[x];

arr[x] = arr[y];

arr[y] = temp;

}

}

**package** com.cdac;

**public** **class** QuickSortDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = {5,7,20,15,1,10,8,12,3,4,9};

QuickSort sort = **new** QuickSort();

sort.quickSorting(arr, 0, arr.length-1);

System.***out***.println("After Quick Sort");

**for** (**int** i=0;i<arr.length;i++)

{

System.***out***.print(arr[i] + " , ");

}

}

}

Output:

Pivot Value = 10

Pivot Value = 4

Pivot Value = 3

Pivot Value = 7

Pivot Value = 9

Pivot Value = 15

After Quick Sort

1 , 3 , 4 , 5 , 7 , 8 , 9 , 10 , 12 , 15 , 20 ,

ii) Using Radix to the base 11

iii) Using Multiplication method

Use constant c = 0.496235

Consider the size of array in each case as 100

Write the demo code to test each hashing method.

iv) Resolve collision in division method using Linear probing.

v) Resolve collision in division method using Quadratic probing

vi) Combine any of the above 2 hashing methods (From i to iii) and demonstrate double hashing

--------------------------------------------------------------------------------------

ASSIGNMENT : 9

Assign. Develop the following hashing functions:

i) Using division method

  Hint: Use largest prime number smaller than the array size

**Division hashing code**

**package** divisionhashing;

**public** **class** HashingByDivision

{

**public** **int** myHash(**int** element )

{

**int** copyElement=element;

**int** newNumber=0;

**if**(copyElement>0)

{

newNumber=copyElement%97;

}

**return** newNumber;

}

}

**Demo code**

**package** divisionhashing;

**import** java.util.Scanner;

**public** **class** HashingByDivisionDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = **new** **int**[100];

**int** element;

System.***out***.println("Input 5 +ve integers of the array");

Scanner sc = **new** Scanner(System.***in***);

HashingByDivision obj = **new** HashingByDivision ();

**int** hashCode;

**for** (**int** i=1;i<=5;i++)

{

element = sc.nextInt();

hashCode = obj.myHash(element);

//System.out.println("Hash Code = " + hashCode);

**if** (arr[hashCode]==0) // Array index is vacant

{

arr[hashCode] = element;

}

**else** // If not zero, then element already exists.

{

System.***out***.println("Collision for element number " + i);

}

}

System.***out***.println("Non zero elements of the array are:");

**for** (**int** i=0;i<arr.length;i++)

{

**if** (arr[i]!=0)

{

System.***out***.println("arr[" + i + "] = " + arr[i]);

}

}

sc.close();

}

}

OUTPUT:

Input 5 +ve integers of the array

11

22

33

44

55

Non zero elements of the array are:

arr[11] = 11

arr[22] = 22

arr[33] = 33

arr[44] = 44

arr[55] = 55

ii) Using Radix to the base 11

**Radix hashing code**

**package** radixhashing;

**public** **class** HashingByRadix {

// Radix Transformation hash method

// New Radix base as 11

**public** **int** myHash(**int** element)

{

**int** newValue=0;

**int** copyElement = element;

**int** i=0;

**int** dig;

**while** (copyElement > 0)

{

dig = copyElement % 10;

newValue = newValue + (**int**)Math.*pow*(11, i)\*dig;

i++;

copyElement /= 10;

}

**return** newValue%100;

}

}

**package** radixhashing;

**import** java.util.Scanner;

**public** **class** HashingByRadixDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = **new** **int**[100];

**int** element;

System.***out***.println("Input 5 +ve integers of the array");

Scanner sc = **new** Scanner(System.***in***);

HashingByRadix obj = **new** HashingByRadix ();

**int** hashCode;

**for** (**int** i=1;i<=5;i++)

{

element = sc.nextInt();

hashCode = obj.myHash(element);

//System.out.println("Hash Code = " + hashCode);

**if** (arr[hashCode]==0) // Array index is vacant

{

arr[hashCode] = element;

}

**else** // If not zero, then element already exists.

{

System.***out***.println("Collision for element number " + i);

}

}

System.***out***.println("Non zero elements of the array are:");

**for** (**int** i=0;i<arr.length;i++)

{

**if** (arr[i]!=0)

{

System.***out***.println("arr[" + i + "] = " + arr[i]);

}

}

sc.close();

}

}

**OUTPUT:**

Input 5 +ve integers of the array

11

12

34

43

55

Non zero elements of the array are:

arr[12] = 11

arr[13] = 12

arr[37] = 34

arr[47] = 43

arr[60] = 55

iii) Using Multiplication method : Use constant c = 0.496235

**package** multiplicationhashing;

**public** **class** HashingByMultipication

{

**public** **int** myHash(**int** element)

{

**double** copy;

**int** newElement=0;

**double** constant=0.496235;

copy=constant\*element;

//System.out.println(copy);

copy=copy%1;

//System.out.println(copy);

newElement=(**int**)Math.*floor*(copy\*100);

//System.out.println(newElement);

**return** newElement;

}

}

**package** multiplicationhashing;

**import** java.util.Scanner;

**public** **class** HashingByMultipicationDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = **new** **int**[100];

**int** element;

System.***out***.println("Input 5 +ve integers of the array");

Scanner sc = **new** Scanner(System.***in***);

HashingByMultipication obj = **new** HashingByMultipication();

**int** hashCode;

**for** (**int** i=1;i<=5;i++)

{

element = sc.nextInt();

hashCode = obj.myHash(element);

//System.out.println("Hash Code = " + hashCode);

**if** (arr[hashCode]==0) // Array index is vacant

{

arr[hashCode] = element;

}

**else** // If not zero, then element already exists.

{

System.***out***.println("Collision forelement number" + i);

}

}

System.***out***.println("Non zero elements of the array are:");

**for** (**int** i=0;i<arr.length;i++)

{

**if** (arr[i]!=0)

{

System.***out***.println("arr[" + i + "] = " + arr[i]);

}

}

sc.close();

}

}

**Output:**

Input 5 +ve integers of the array

23

32

4

5

67

Non zero elements of the array are:

arr[24] = 67

arr[41] = 23

arr[48] = 5

arr[87] = 32

arr[98] = 4

iv) Resolve collision in division method using Linear probing.

**package** linearprobing;

**public** **class** HashingByDivision {

**public** **int** myHash(**int** element) {

**int** copyElement = element;

**int** newNumber = 0;

**if** (copyElement > 0) {

newNumber = copyElement % 97;

}

**return** newNumber;

}

}

**package** linearprobing;

**import** java.util.Scanner;

**public** **class** HashingByDivisionDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = **new** **int**[100];

**int** element;

System.***out***.println("Input 5 +ve integers of the array");

Scanner sc = **new** Scanner(System.***in***);

HashingByDivision obj = **new** HashingByDivision();

**int** hashCode;

**for** (**int** i=1;i<=5;i++)

{

element = sc.nextInt();

hashCode = obj.myHash(element);

//System.out.println("Hash Code = " + hashCode);

**if** (arr[hashCode]==0) // Array index is vacant

{

arr[hashCode] = element;

}

**else** // If not zero, then element already exists.

{

System.***out***.println("Collision for element number " + i);

**for**(**int** j=0;j<100;j++)

{

hashCode=(hashCode+j)%100;

**if**(arr[hashCode]==0)

{

arr[hashCode]=element;

**break**;

}

}

}

}

System.***out***.println("Non zero elements of the array are:");

**for** (**int** i=0;i<arr.length;i++)

{

**if** (arr[i]!=0)

{

System.***out***.println("arr[" + i + "] = " + arr[i]);

}

}

sc.close();

}

}

**OUTPUT:**

Input 5 +ve integers of the array

1

22

22

Collision for element number 3

12

33

Non zero elements of the array are:

arr[1] = 1

arr[12] = 12

arr[22] = 22

arr[23] = 22

arr[33] = 33

v) Resolve collision in division method using Quadratic probing

**package** quadraticprobing;

**public** **class** HashingByDivision

{

**public** **int** myHash(**int** element )

{

**int** copyElement=element;

**int** newNumber=0;

**if**(copyElement>0)

{

newNumber=copyElement%97;

}

**return** newNumber;

}

}

**package** quadraticprobing;

**import** java.util.Scanner;

**public** **class** HashingByDivisionDemo {

**public** **static** **void** main(String[] args) {

**int**[] arr = **new** **int**[100];

**int** element;

System.***out***.println("Input 5 +ve integers of the array");

Scanner sc = **new** Scanner(System.***in***);

HashingByDivision obj = **new** HashingByDivision();

**int** hashCode;

**for** (**int** i=1;i<=5;i++)

{

element = sc.nextInt();

hashCode = obj.myHash(element);

//System.out.println("Hash Code = " + hashCode);

**if** (arr[hashCode]==0) // Array index is vacant

{

arr[hashCode] = element;

}

**else** // If not zero, then element already exists.

{

System.***out***.println("Collision for element number " + i);

**for**(**int** j=0;j<100;j++)

{

hashCode=(hashCode+(j\*j))%100;

**if**(arr[hashCode]==0)

{

arr[hashCode]=element;

**break**;

}

}

}

}

System.***out***.println("Non zero elements of the array are:");

**for** (**int** i=0;i<arr.length;i++)

{

**if** (arr[i]!=0)

{

System.***out***.println("arr[" + i + "] = " + arr[i]);

}

}

sc.close();

}

}

**Output:**

Input 5 +ve integers of the array

11

11

Collision for element number 2

11

Collision for element number 3

11

Collision for element number 4

11

Collision for element number 5

Non zero elements of the array are:

arr[11] = 11

arr[12] = 11

arr[16] = 11

arr[25] = 11

arr[41] = 11

vi) Combine any of the above 2 hashing methods (From i to iii) and demonstrate double hashing:

**package** doublehashing;

**public** **class** HashingByDivision

{

**public** **int** myHash(**int** element )

{

**int** copyElement=element;

**int** newNumber=0;

**if**(copyElement>0)

{

newNumber=copyElement%97;

}

**return** newNumber;

}

}

**package** doublehashing;

**import** java.util.Scanner;

**public** **class** Demo

{

**public** **static** **void** main(String[] args) {

**int** element;

**int** []arr=**new** **int**[100];

Scanner sc=**new** Scanner(System.***in***);

HashingByDivision obj=**new** HashingByDivision();

**int** hashCode,hashCode2;

**for**(**int** i=0;i<5;i++)

{

System.***out***.println("enter "+(i+1)+" element ");

element=sc.nextInt();

hashCode=obj.myHash(element);

hashCode2=hashCode+hashCode;

**if**(arr[hashCode]==0 || arr[hashCode2]==0)

{

**if**(arr[hashCode]==0)

arr[hashCode]=element;

**else**

arr[hashCode2]=element;

}

**else**

{

System.***out***.println("collision at "+(i+1));

**for**(**int** j=0;j<100;j++)

{

hashCode=(hashCode+j)%100;

**if**(arr[hashCode]==0)

{

arr[hashCode]=element;

**break**;

}

}

}

}

System.***out***.println("array non zero element");

**for**(**int** i=0;i<100;i++)

{

**if**(arr[i]!=0)

{

System.***out***.println("arr["+i+"]="+arr[i]);

}

}

sc.close();

}

}

OUTPUT:

enter 1 element

11

enter 2 element

11

enter 3 element

22

enter 4 element

11

collision at 4

enter 5 element

22

collision at 5

array non zero element

arr[11]=11

arr[12]=11

arr[22]=11

arr[23]=22

arr[44]=22

ASSIGNMENT : 10

Assign.  Implement a graph using adjacency Matrix and traverse using Depth First Traversal using Recursion.

Assign. Implement a graph and traverse using Stack and Queue:

**package** com.cdac;

**import** java.util.Stack;

// Undirected graph

**public** **class** Graph {

**private** **int** vertices;

**private** **int**[][] adj;

**public** Graph(**int** v)

{

// Initialize no. of vertices

vertices = v;

// Define the size of Adjacency Matrix

// If 1 then there exists a path, else 0

adj = **new** **int**[vertices][vertices];

}

**public** **void** addEdge(**int** i,**int** j)

{

adj[i][j]=1;

adj[j][i]=1;

}

**public** **void** removeEdge(**int** i,**int** j)

{

adj[i][j]=0;

adj[j][i]=0;

}

**public** **void** show()

{

**int** i,j;

System.***out***.println("The Graph Matrix is as follows:");

System.***out***.print("\t");

**for** (i=0;i<vertices;i++)

{

System.***out***.print(i+"\t");

}

System.***out***.println();

**for** (i=0;i<vertices;i++)

{

System.***out***.print(i+"\t");

**for** (j=0;j<vertices;j++)

{

System.***out***.print(adj[i][j]+"\t");

}

System.***out***.println();

}

}

**public** **void** bft(**int** startVertex)

{

**boolean**[] visited=**new** **boolean**[vertices];

Queue q=**new** Queue(10);

q.enqueue(startVertex);

visited[startVertex]=**true**;

**while**(!q.isEmpty())

{

**int** v=q.dequeue();

System.***out***.println(v);

**for**(**int** i=0;i<vertices;i++)

{

**if**((adj[v][i]==1)&& !visited[i])

{

q.enqueue(i);

visited[i]=**true**;

}

}

}

}

**public** **void** dft(**int** startVertex)

{

**boolean**[] visited=**new** **boolean**[vertices];

Stack s=**new** Stack();

s.push(startVertex);

visited[startVertex]=**true**;

**while**(!s.isEmpty())

{

**int** v=(**int**) s.pop();

//int v=q.dequeue();

System.***out***.println(v+" ");

**for**(**int** i=0;i<vertices;i++)

{

**if**((adj[v][i]==1)&& !visited[i]){

s.push(i);

visited[i]=**true**;

}

}

}

}

}

**package** com.cdac;

**import** java.util.Scanner;

**public** **class** GraphDemo {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("How many vertices?");

**int** v = sc.nextInt();

Graph g = **new** Graph(v);

**int** src,tgt;

System.***out***.println("Input the Source and Target vertex for which edge exists -1 to end");

src = sc.nextInt();

tgt = sc.nextInt();

**do**

{

g.addEdge(src, tgt);

System.***out***.println("Input the Source and Target vertex for which edge exists -1 to end");

src = sc.nextInt();

tgt = sc.nextInt();

} **while** (src!=-1);

g.show();

System.***out***.println("BFT");

g.bft(0);// 0-4 values

System.***out***.println("DFT");

g.dft(0);

//g.show();

sc.close();

}

}

OUTPUT:

How many vertices?

5

Input the Source and Target vertex for which edge exists -1 to end

0 1

Input the Source and Target vertex for which edge exists -1 to end

0 2

Input the Source and Target vertex for which edge exists -1 to end

1 3

Input the Source and Target vertex for which edge exists -1 to end

1 4

Input the Source and Target vertex for which edge exists -1 to end

3 4

Input the Source and Target vertex for which edge exists -1 to end

-1 -1

The Graph Matrix is as follows:

0 1 2 3 4

0 0 1 1 0 0

1 1 0 0 1 1

2 1 0 0 0 0

3 0 1 0 0 1

4 0 1 0 1 0

BFT

0

1

2

3

4

DFT

0

2

1

4

3