DATA STRUCTURES AND ALGORITHMS (23CSE203)

LAB MANUAL

Submitted to: Dr. Bidyapati Thiyam

Submitted by: Mehuli Sarkar

Roll no: AV.SC.U4CSE24150

Section: CSE-B

Date:

LAB 1

1) **Write a Java program to:**
    - **Traverse a list and print all elements.**
    - **Insert an element at a given index (shift elements using list slicing).**
    - **Delete an element at a given index (remove and shift).**
    - **Search for an element using linear search (return index or -1).**
    - **Update an element at a given index.**

Test with input list [5, 3, 8, 1, 9]; insert 7 at index 2, delete index 1, search 8, update index 3 to 4.

Algorithm:

```
traverse():
for i = 0 to size-1
   print arr[i]
insert():
if index < 0 or index > size: error
for i = size-1 down to index
   arr[i+1] = arr[i]
arr[index] = element
size = size + 1
delete():
if index < 0 or index >= size: error
for i = index to size-2
   arr[i] = arr[i+1]
size = size – 1
linear_search():
for i = 0 to size-1
   if arr[i] == target: return i
return -1

update_index():
if index < 0 or index >= size: error
arr[index] = newValue
```

Code:

```
class ArrayOperations {
   int[] arr;
   int size;
```

```
public ArrayOperations(int capacity) {
    arr = new int[capacity];
    size = 0;
}

void traverse() {
    for (int i = 0; i < size; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}

void insert(int index, int element) {
    if (index < 0 || index > size) {
        System.out.println("Invalid index");
        return;
    }
    for (int i = size - 1; i >= index; i--) {
        arr[i + 1] = arr[i];
    }
    arr[index] = element;
    size++;
}

void delete(int index) {
    if (index < 0 || index >= size) {
        System.out.println("Invalid index");
        return;
    }
    for (int i = index; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    size--;
}

int search(int element) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == element)
            return i;
    }
```

```
        return -1;
    }

    void update(int index, int newValue) {
        if (index < 0 || index >= size) {
            System.out.println("Invalid index");
            return;
        }
        arr[index] = newValue;
    }

    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        ArrayOperations list = new ArrayOperations(10);
        int[] input = { 5, 3, 8, 1, 9 };

        for (int i = 0; i < input.length; i++) {
            list.arr[i] = input[i];
            list.size++;
        }

        System.out.print("Original list: ");
        list.traverse();

        list.insert(2, 7);
        System.out.print("After inserting 7 at index 2: ");
        list.traverse();

        list.delete(1);
        System.out.print("After deleting index 1: ");
        list.traverse();

        int index = list.search(8);
        System.out.println("Index of 8: " + index);

        list.update(3, 4);
        System.out.print("After updating index 3 to 4: ");
        list.traverse();
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\dsa lab manual>java ArrayOperations
shravani, 24130
Original list: 5 3 8 1 9
After inserting 7 at index 2: 5 3 7 8 1 9
After deleting index 1: 5 7 8 1 9
Index of 8: 2
After updating index 3 to 4: 5 7 8 4 9
```

LAB 2

1) **Implement a singly linked list in Python with a Node class (data, next) and methods: - Insert at front and end. - Delete at front and traverse.**

Algorithm:

InsertAtFront(data):
1. Create newNode with given data
2. newNode.next = head
3. head = newNode

InsertAtEnd(data):
1. Create newNode with given data
2. newNode.next = null
3. If head == null:
    head = newNode
    return
4. temp = head
5. While temp.next != null:
    temp = temp.next
6. temp.next = newNode

DeleteAtFront():
1. If head == null:
    Print "List is empty"
    return
2. head = head.next

Traverse():
1. If head == null:
    Print "List is empty"
    return
2. temp = head
3. While temp != null:
    Print temp.data
    temp = temp.next

Code:

```
class Node {
  int data;
  Node next;
```

```
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class SinglyLinkedList {
    private Node head;
    public void insertFront(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }
    public void insertEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
  public void deleteFront() {
        if (head == null) {
            System.out.println("List is empty. Nothing to delete.");
            return;
        }
        head = head.next;
    }
   public void traverse() {
        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();
    }
}
public class Lab1a {
    public static void main(String[] args) {
        System.out.println("Shravani, 24130");
        SinglyLinkedList list = new SinglyLinkedList();
        list.insertFront(10);
        list.insertFront(20);
        list.insertEnd(30);
        list.insertEnd(40);
        list.traverse();
        list.deleteFront();
        list.traverse();
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\JAVA DSA>java Lab1a
Shravani, 24130
Linked List: 20 10 30 40
Linked List: 10 30 40
```

2) Implement a stack using a linked list with push, pop, peek, and is_empty.

Algorithm:

```
Push(x):
    create newNode with data = x
    newNode.next = top
    top = newNode
    print x + " pushed"

Pop():
    if top == null:
        print "Stack Underflow"
        return
    else:
        temp = top
        top = top.next
        print temp.data + " popped"

Peek():
    if top == null:
        print "Stack is empty"
        return
    else:
        print "Top element = " + top.data

IsEmpty():
    if top == null:
        return true
    else:
        return false
```

Code:

```
class Node {
  int data;
  Node next;

  Node(int data) {
    this.data = data;
    this.next = null;
  }
}
```

```
class Stack {
    private Node top;

    public Stack() {
        this.top = null;
    }

    public void push(int value) {
        Node newNode = new Node(value);
        newNode.next = top;
        top = newNode;
        System.out.println(value + " pushed into stack");
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow! Cannot pop.");
            return -1;
        }
        int poppedValue = top.data;
        top = top.next;
        return poppedValue;
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return -1;
        }
        return top.data;
    }

    public boolean isEmpty() {
        return top == null;
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
```

```
        }
        Node temp = top;
        System.out.print("Stack elements: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}

public class StackUsingLL {
    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        Stack stack = new Stack();

        stack.push(10);
        stack.push(20);
        stack.push(30);

        stack.display();

        System.out.println("Top element is: " + stack.peek());
        System.out.println("Popped element: " + stack.pop());

        stack.display();
        System.out.println("Is stack empty? " + stack.isEmpty());
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\JAVA DSA>java StackUsingLL
shravani, 24130
10 pushed into stack
20 pushed into stack
30 pushed into stack
Stack elements: 30 20 10
Top element is: 30
Popped element: 30
Stack elements: 20 10
Is stack empty? false
```

3) Implement a stack using array with push, pop, peek, and is_empty.

Algorithm:

Push(x):
If top == MAX - 1
    Print "Stack Overflow" and exit
Else
    top = top + 1
    stack[top] = x
    Print x inserted into stack
Pop():
If top == -1
    Print "Stack Underflow" and exit
Else
    element = stack[top]
    top = top - 1
    Return element
Peek():
If top == -1
    Print "Stack is empty"
Else
    Return stack[top]
is_empty():
If top == -1
    Return true
Else
    Return false

Code:

```
class Stack {
   private int[] stack;
   private int top;
   private int maxSize;
public Stack(int size) {
      maxSize = size;
      stack = new int[maxSize];
      top = -1;
   }
   public void push(int value) {
      if (top == maxSize - 1) {
         System.out.println("Stack Overflow! Cannot push " + value);
```

```
            return;
        }
        stack[++top] = value;
        System.out.println(value + " pushed into stack");
    }
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow! Cannot pop.");
            return -1;
        }
        return stack[top--];
    }
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return -1;
        }
        return stack[top];
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
        }
        System.out.print("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
public class StackUsingArray {
    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        Stack stack = new Stack(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
```

```
        stack.display();
        System.out.println("Top element is: " + stack.peek());
        System.out.println("Popped element: " + stack.pop());
        stack.display();
        System.out.println("Is stack empty? " + stack.isEmpty());
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\JAVA DSA>java StackUsingArray
shravani, 24130
10 pushed into stack
20 pushed into stack
30 pushed into stack
Stack elements: 10 20 30
Top element is: 30
Popped element: 30
Stack elements: 10 20
Is stack empty? false
```

4) Write a Java program to reverse a string using a stack.

Algorithm:

1. Initialize an empty stack of characters.
2. For each character c in the string str:
   Push c into the stack.
3. Initialize an empty result string rev.
4. While the stack is not empty:
   Pop a character from the stack and append it to rev.
5. Return rev as the reversed string.

Code:

```
class CharStack {
  private char[] stack;
  private int top;
  private int maxSize;
  public CharStack(int size) {
    maxSize = size;
    stack = new char[maxSize];
    top = -1;
  }
  public void push(char c) {
    if (top == maxSize - 1) {
      System.out.println("Stack Overflow! Cannot push " + c);
      return;
    }
    stack[++top] = c;
  }
  public char pop() {
    if (isEmpty()) {
      System.out.println("Stack Underflow! Cannot pop.");
      return '\0';
    }
    return stack[top--];
  }
  public boolean isEmpty() {
    return top == -1;
  }
}
public class ReverseStringUsingStack {
  public static String reverse(String str) {
```

```
        CharStack stack = new CharStack(str.length());

        for (int i = 0; i < str.length(); i++) {
            stack.push(str.charAt(i));
        }
        StringBuilder rev = new StringBuilder();
        while (!stack.isEmpty()) {
            rev.append(stack.pop());
        }
        return rev.toString();
    }
    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        String str = "DATA STRUCTURES AND ALGORITHMS";
        System.out.println("Original String: " + str);
        String reversed = reverse(str);
        System.out.println("Reversed String: " + reversed);
    }
}
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\JAVA DSA>java ReverseStringUsingStack
shravani, 24130
Original String: DATA STRUCTURES AND ALGORITHMS
Reversed String: SMHTIROGLA DNA SERUTCURTS ATAD
```

LAB 3

1) **Implement a queue using an array with enqueue, dequeue, and display. The program should accept a sequence of operations and show the queue status after each operation.**

Algorithm:

Start
Initialize front = 0, rear = -1
Create an array queue[MAX]
Enqueue(x):
If rear == MAX - 1 → Print "Queue Overflow"
Else
    rear = rear + 1
    queue[rear] = x
Dequeue():
If front > rear → Print "Queue Underflow"
Else
    removed = queue[front]
    front = front + 1
    Print removed element
Display():
If front > rear, print "Queue is empty"
Else
    print elements from front to rear

Code:

```
class Queue {
   private int[] queue;
   private int front, rear, maxSize;
   public Queue(int size) {
      maxSize = size;
      queue = new int[maxSize];
      front = 0;
      rear = -1;
   }
   public void enqueue(int value) {
      if (rear == maxSize - 1) {
         System.out.println("Queue Overflow! Cannot enqueue " + value);
         return;
```

```
        }
        queue[++rear] = value;
        System.out.println("Operation: enqueue " + value);
        display();
    }
    public void dequeue() {
        if (front > rear) {
            System.out.println("Queue Underflow! Cannot dequeue.");
            return;
        }
        int removed = queue[front++];
        System.out.println("Operation: dequeue");
        System.out.println("Removed: " + removed);
        display();
    }
    public void display() {
        if (front > rear) {
            System.out.println("Queue: []");
            return;
        }
        System.out.print("Queue: [");
        for (int i = front; i <= rear; i++) {
            System.out.print(queue[i]);
            if (i < rear) System.out.print(", ");
        }
        System.out.println("]");
    }
}
public class QueueUsingArray {
    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        Queue q = new Queue(5);
        System.out.println("Initial Queue Size: 5");
        q.enqueue(10);
        q.enqueue(20);
        q.dequeue();
        q.enqueue(30);
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\JAVA DSA>java QueueUsingArray
shravani, 24130
Initial Queue Size: 5
Operation: enqueue 10
Queue: [10]
Operation: enqueue 20
Queue: [10, 20]
Operation: dequeue
Removed: 10
Queue: [20]
Operation: enqueue 30
Queue: [20, 30]
```

2) **Implement a queue using linked list with menu-driven operations.**

Algorithm:

```
Enqueue():
create newNode with value
if front == null
   front = rear = newNode
else
   rear.next = newNode
   rear = newNode
Dequeue():
if front == null: queue empty
else
   value = front.data
   front = front.next
   if front == null: rear = null
   return value
Traverse():
temp = front
while temp != null
   print temp.data
   temp = temp.next
Peek():
if front == null: queue empty
else
   return front.data
```

Code:

```
class Node{
   int data;
   Node next;
   Node(int data){
      this.data= data;
      this.next= null;
   }
}
class Queue{
   Node front, rear;
   public Queue(){
      front=rear=null;
   }
```

```
    public void enqueue(int value){
        Node newnode= new Node(value);
        if(rear==null){
            front=rear=newnode;
        }
        else{
            rear.next= newnode;
            rear= newnode;
        }
        System.out.println("Enqueued "+value);
    }

    public void dequeue(){
        if(front==null){
            System.out.println("Underflow");
            return;
        }
        int removed= front.data;
        front= front.next;
        if(front==null){
            rear=null;
        }
        System.out.println("Removed "+removed);
    }
    public void display(){
        if(front==null){
            System.out.println("Empty queue");
        }
        Node temp= front;
        System.out.print("Output: ");
        while(temp!=null){
            System.out.print(temp.data+"->");
            temp=temp.next;
        }
        System.out.println("NULL");
    }
}
public class QueueUsingLL{
    public static void main(String args[]){
        System.out.println("shravani, 24130");
        Queue q= new Queue();
```

```java
        System.out.println("1. Enqueue 15");
        q.enqueue(15);
        System.out.println("2. Enqueue 25");
        q.enqueue(25);
        System.out.println("3. Display");
        q.display();
        System.out.println("4. Dequeue");
        q.dequeue();
        System.out.println("5. Display");
        q.display();
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\dsa lab manual>java QueueUsingLL
shravani, 24130
1. Enqueue 15
Enqueued 15
2. Enqueue 25
Enqueued 25
3. Display
Output: 15->25->NULL
4. Dequeue
Removed 15
5. Display
Output: 25->NULL
```

3) **Implement a circular queue using arrays of size 4. Perform a series of operations to demonstrate circular movement.**

Algorithm:

```
Enqueue():
if (front == (rear+1) % size): queue full
else
   if front == -1: front = 0
   rear = (rear+1) % size
   arr[rear] = value
Dequeue():
if front == -1: queue empty
else
   value = arr[front]
   if front == rear: front = rear = -1
   else: front = (front+1) % size
   return value
Traverse():
if front == -1: queue empty
else
   i = front
   loop
      print arr[i]
      if i == rear: break
      i = (i+1) % size
Peek():
if front == -1: queue empty
else: return arr[front]
```

Code:

```
class CircularQueue{
   int queue[], front, rear, size;
   public CircularQueue(int size){
      this.size= size;
      queue= new int[size];
      front=rear=-1;
   }
   public void enqueue(int value){
      if((front==0 & rear==size-1)||(rear+1)%size==front){
         System.out.println("Operation: enqueue "+value);
         System.out.println("Overflow");
```

```
            return;
        }
        if(front==-1){
            front=rear=0;
            queue[rear]=value;
        }
        else{
            rear= (rear+1)%size;
            queue[rear]=value;
        }
        System.out.println("Operation: enqueue "+value);
        display();
    }
    public void dequeue(){
        if(front==-1){
            System.out.println("Underflow");
            return;
        }
        int removed= queue[front];
        if(front==rear){
            front=rear=-1;
        }
        else{
            front= (front+1)%size;
        }
        System.out.println("Operation: dequeue");
        System.out.println("Removed: "+removed);
        display();
    }
    public void display(){
        System.out.print("Queue: [");
        for(int i=0; i<size; i++){
            if(i>=front && i<=rear){
                System.out.print(queue[i]);
            }
            else{
                System.out.print("_");
            }
            if(i != size-1){
                System.out.print(", ");
            }
```

```
        }
        System.out.println("] front="+ front+" rear="+rear);
    }
}

public class CircularQueueArray{
    public static void main(String args[]){
        System.out.println("shravani, 24130");
        CircularQueue q= new CircularQueue(4);
        q.enqueue(1);
        q.enqueue(2);
        q.enqueue(3);
        q.dequeue();
        q.enqueue(4);
        q.enqueue(5);
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\dsa lab manual>java CircularQueueArray
shravani, 24130
Operation: enqueue 1
Queue: [1, _, _, _] front=0 rear=0
Operation: enqueue 2
Queue: [1, 2, _, _] front=0 rear=1
Operation: enqueue 3
Queue: [1, 2, 3, _] front=0 rear=2
Operation: dequeue
Removed: 1
Queue: [_, 2, 3, _] front=1 rear=2
Operation: enqueue 4
Queue: [_, 2, 3, 4] front=1 rear=3
Operation: enqueue 5
Queue: [_, _, _, _] front=1 rear=0
```

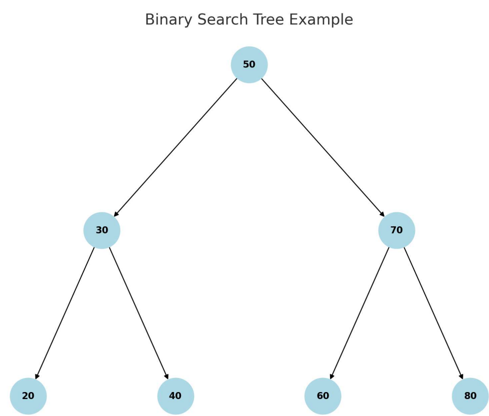
LAB 4

1)

LAB 5

1) To implement a Binary Search Tree (BST) and perform the following operations:

1. **Insertion of nodes into the BST.**
2. **Searching for a given key in the BST.**
3. **Deletion of nodes, covering all cases:**
   - **Node is a leaf (no children)**
   - **Node has one child**
   - **Node has two children.**



Algorithm:

Insert():

```
if root == null:
   return new Node(key)
if key < root.data:
   root.left = insert(root.left, key)
else if key > root.data:
   root.right = insert(root.right, key)
return root
```

Search():

```
function search(root, key):
   if root == null or root.data == key:
      return root
   if key < root.data:
      return search(root.left, key)
   else:
      return search(root.right, key)
```

Delete():

```
//case 1- node with no child
if node.left == null and node.right == null:
   return null
//case 2- node with one child
if node.left == null: return node.right
if node.right == null: return node.left
//case 3- node with two children
function delete(root, key):
   if root == null: return null
   if key < root.data:
      root.left = delete(root.left, key)
   else if key > root.data:
      root.right = delete(root.right, key)
   else:
      if root.left == null and root.right == null: return null
      else if root.left == null: return root.right
      else if root.right == null: return root.left
      else:
         successor = minValue(root.right)
         root.data = successor.data
         root.right = delete(root.right, successor.data)
   return root
```

Code:

```
class BST {
   class Node {
      int key;
      Node left, right;
      public Node(int item) {
         key = item;
         left = right = null;
      }
   }

   Node root;

   public BST() {
      root = null; //initially
   }

   void insert(int key) {
```

```
    root = insertNode(root, key);
}

Node insertNode(Node root, int key) {
    if (root == null) {
        root = new Node(key);
        return root;
    }
    if (key < root.key) {
        root.left = insertNode(root.left, key);
    } else if (key > root.key) {
        root.right = insertNode(root.right, key);
    }
    return root;
}

boolean search(int key) {
    return searchNode(root, key);
}

boolean searchNode(Node root, int key) {
    if (root == null) return false;
    if (root.key == key) return true;
    else if (key < root.key) {
        return searchNode(root.left, key);
    } else {
        return searchNode(root.right, key);
    }
}

void delete(int key) {
    root = deleteNode(root, key);
}

Node deleteNode(Node root, int key) {
    if (root == null) return root;

    if (key < root.key) {
        root.left = deleteNode(root.left, key);
    } else if (key > root.key) {
        root.right = deleteNode(root.right, key);
```

```
    } else {
        // node with no child
        if (root.left == null && root.right == null) return null;

        // node with one child
        if (root.left == null) return root.right;
        else if (root.right == null) return root.left;

        // node with two children
        root.key = minValue(root.right);
        root.right = deleteNode(root.right, root.key);
    }
    return root;
}

int minValue(Node root) {
    int minv = root.key;
    while (root.left != null) {
        root = root.left;
        minv = root.key;
    }
    return minv;
}

void inorder() {
    inorder(root);
    System.out.println();
}

void inorder(Node root) {
    if (root != null) {
        inorder(root.left);
        System.out.print(root.key + " ");
        inorder(root.right);
    }
}

public static void main(String args[]) {
    System.out.println("shravani, 24130");
    BST tree = new BST();
```

```
        tree.insert(50);
        tree.insert(30);
        tree.insert(70);
        tree.insert(20);
        tree.insert(40);
        tree.insert(60);
        tree.insert(80);

        System.out.println("Inorder traversal: ");
        tree.inorder();

        System.out.println("Insert 15 and 75: ");
        tree.insert(15);
        tree.insert(75);
        tree.inorder();

        System.out.println("Delete 20: ");
        tree.delete(20);
        tree.inorder();

        System.out.println("Delete 30: ");
        tree.delete(30);
        tree.inorder();

        System.out.println("Delete 50: ");
        tree.delete(50);
        tree.inorder();
        System.out.println("Search 60: " + tree.search(60));
        System.out.println("Search 100: " + tree.search(100));
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\dsa lab manual>java BST
shravani, 24130
Inorder traversal:
20 30 40 50 60 70 80
Insert 15 and 75:
15 20 30 40 50 60 70 75 80
Delete 20:
15 30 40 50 60 70 75 80
Delete 30:
15 40 50 60 70 75 80
Delete 50:
15 40 60 70 75 80
Search 60: true
Search 100: false
```

LAB 6

1) Implement a min-heap in Python using a list with:

- **Insert, extract-min, and heapify (use heapq optionally).**

Algorithm:

```
Insert():
 append value to end of heap
   i = index of last element
   while i > 0 and heap[parent(i)] > heap[i]:
      swap heap[i], heap[parent(i)]
      i = parent(i)
Heapify():
function heapify(heap, i, size):
   left = 2*i
   right = 2*i + 1
   smallest = i/2
   if left < size and heap[left] < heap[smallest]:
      smallest = left
   if right < size and heap[right] < heap[smallest]:
      smallest = right
   if smallest != i:
      swap heap[i], heap[smallest]
      heapify(heap, smallest, size)
extractMin():
function extractMin(heap):
   if heap empty: return None
   minValue = heap[0]
   heap[0] = heap[last]
   remove last element
   heapify(heap, 0, size)
   return minValue
```

Code:

```
class MinHeap {
   int[] heap;
   int size, capacity;

   public MinHeap(int capacity) {
      this.capacity = capacity;
      heap = new int[capacity + 1]; // index 0 unused
```

```
    size = 0;
}

public void insert(int val) {
    if (size == capacity) return;
    size++;
    heap[size] = val;
    int i = size;
    while (i > 1 && heap[i] < heap[i / 2]) {
        swap(i, i / 2);
        i = i / 2;
    }
}

public int extractMin() {
    if (size <= 0) return Integer.MAX_VALUE;
    int root = heap[1];
    heap[1] = heap[size];
    size--;
    heapify(1);
    return root;
}
void heapify(int i) {
    int smallest = i;
    int left = 2 * i;
    int right = 2 * i + 1;
    if (left <= size && heap[left] < heap[smallest]) smallest = left;
    if (right <= size && heap[right] < heap[smallest]) smallest = right;
    if (smallest != i) {
        swap(i, smallest);
        heapify(smallest);
    }
}
void swap(int i, int j) {
    int temp = heap[i];
    heap[i] = heap[j];
    heap[j] = temp;
}
public void printHeap() {
    for (int i = 1; i <= size; i++) System.out.print(heap[i] + " ");
    System.out.println();
```

```
    }
    public static void main(String[] args) {
        System.out.println("shravani, 24130");
        MinHeap h = new MinHeap(10);
        h.insert(4);
        h.insert(30);
        h.insert(3);
        h.insert(25);
        h.insert(16);
        h.insert(9);
        h.printHeap();
        System.out.println("Extract min: " + h.extractMin());
        h.printHeap();
    }
}
```

Output:

```
C:\Users\shrav\Desktop\JAVA\dsa lab manual>java MinHeap
shravani, 24130
3 16 4 30 25 9
Extract min: 3
4 16 9 30 25
```

2) Implement heapsort using min-heap.
    - Test with: [4, 30, 3, 25, 16, 9].