MCAL11 Advance Data Structures Lab using JAVA

Index

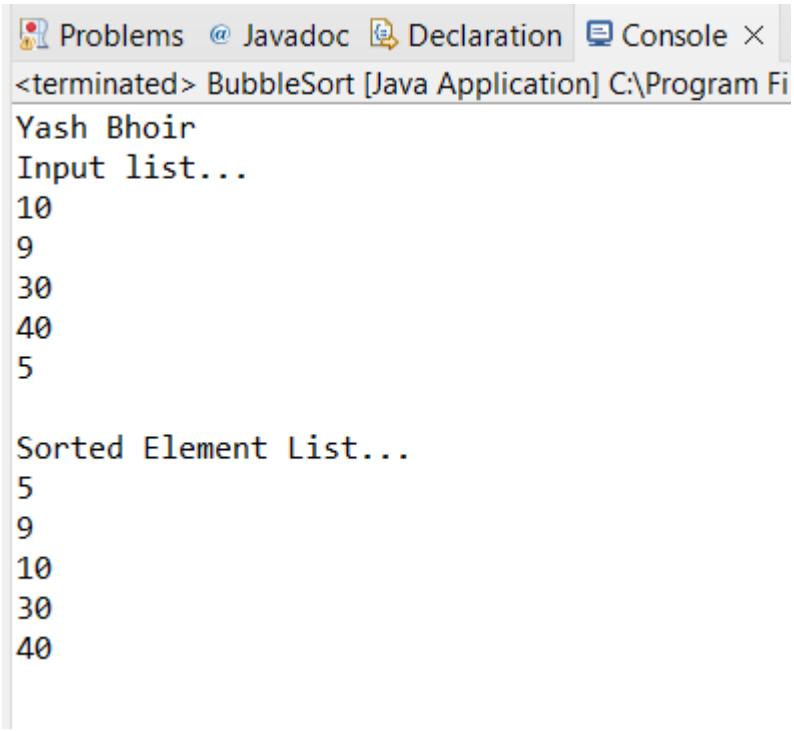
Sr. No.	Topic Name	Date	CO	Sign
1.	Implementation of different searching & sorting techniques	04-10-2024	**CO1**	
2.	Perform various hashing techniques with Linear Probe as collision resolution.	9-12-2024	**CO4**	
3.	Implementation of Stacks, Ordinary Queue & Circular queue (Using arrays)	16-10-2024	**CO2**	
4.	Implementation of Stack Applications like: o infix to postfix o Postfix evaluation o Balancing of Parenthesis	16-10-2024	**CO2**	
5	Implementation of all types of linked lists.	07-12-2024	**CO2**	
6	Demonstrate application of linked list (eg. Sparse matrix,Stack,Queue,Priority & Double ended Queue)	04-10-2024	**CO2**	
7	Create and perform various operations on BST.	04-10-2024	**CO3**	

8	Implementing Heap with different operations.	18-10-2024	**CO3**	
9	Create a Graph storage structure (eg. Adjacency matrix)	5-11-2024	**CO3**	
10	Implementation of Graph traversal. (DFS and BFS)	18-11-2024	**CO3**	
11	Create a minimum spanning tree using any method Kruskal's Algorithm or Prim's Algorithm	4-12-2024	**CO3**	
12	Group project (3 to 4 members) to be given to work on one application to a real world problem.		**CO5**	

PRACTICAL 1: Implementation of different searching & sorting techniques

1.1: Bubble Sort

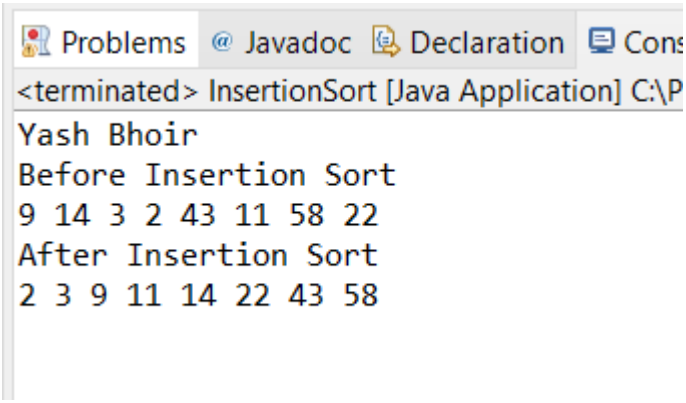
```
import java.io.*;
public class BubbleSort {
	public static void main(String args[])
	{
		int i, j, temp;
		int[] a={10,9,30,40,5};
		System.out.println("Input list...");
	for(i=0;i<5;i++)
		{
			System.out.println(a[i]);
		}
		System.out.println();
	for(i=0;i<5;i++)
	{
		for(j=i+1;j<5;j++)
		{
	if(a[j] < a[i])
	{
		temp = a[i];
		a[i] = a[j];
		a[j] = temp;
	}
		}
	}
	System.out.println("Sorted Element List...");
	for(i=0;i<5;i++)
	{
		System.out.println(a[i]);
	}
	}
}
```



1.2: Insertion Sort

```
import java.io.*;
public class InsertionSort {
   public static void insertionSort(int array[]) {
      int n = array.length;
      for (int j = 1; j < n; j++) {
         int key = array[j];
         int i = j-1;
         while ( (i > -1) && ( array [i] > key ) ) {
            array [i+1] = array [i];
            i--;
         }
         array[i+1] = key;
      }
   }


       public static void main(String[] args) {
      int[] arr1 = {9,14,3,2,43,11,58,22};
      System.out.println("Before Insertion Sort");
      for(int i:arr1){
         System.out.print(i+" ");
      }
      System.out.println();
```

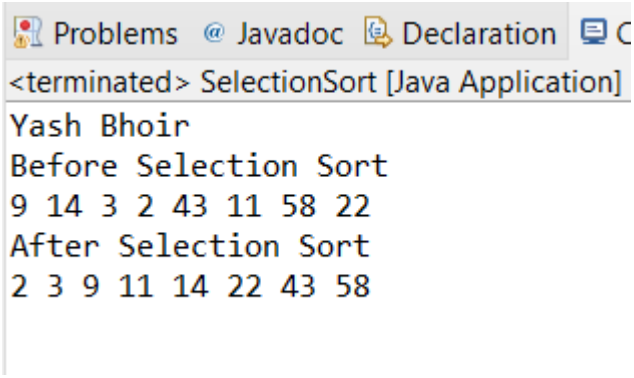
```
        insertionSort(arr1);//sorting array using insertion sort

        System.out.println("After Insertion Sort");
        for(int i:arr1){
           System.out.print(i+" ");
        }

            }

}
```



1.3: Selection Sort

```
package ds_java;
import java.io.*;
public class SelectionSort {
          public static void selectionSort(int[] arr){
        for (int i = 0; i < arr.length - 1; i++)
        {
           int index = i;
           for (int j = i + 1; j < arr.length; j++){
              if (arr[j] < arr[index]){
                 index = j;//searching for lowest index
              }
           }
           int smallerNumber = arr[index];
           arr[index] = arr[i];
           arr[i] = smallerNumber;
        }
     }

     public static void main(String a[]){
        int[] arr1 = {9,14,3,2,43,11,58,22};
        System.out.println("Before Selection Sort");
        for(int i:arr1){
```

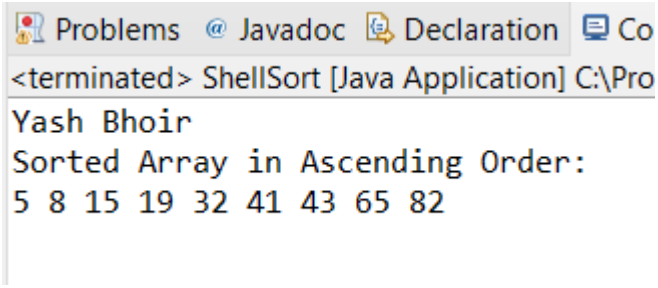
```
        System.out.print(i+" ");
      }
      System.out.println();

      selectionSort(arr1);//sorting array using selection sort

      System.out.println("After Selection Sort");
      for(int i:arr1){
        System.out.print(i+" ");
      }
   }

}
```



1.4: Shell Sort

```
package ds_java;
import java.io.*;
public class ShellSort {
   public void sort(int[] array)
   {
      int n = array.length;
      for (int gap = n / 2; gap > 0; gap /= 2)
      {
         for (int i = gap; i < n; i++)
         {
            int key = array[i];
            int j = i;
            while (j >= gap && array[j - gap] > key)
            {
               array[j] = array[j - gap];
               j -= gap;
            }
            array[j] = key;
         }
      }
```

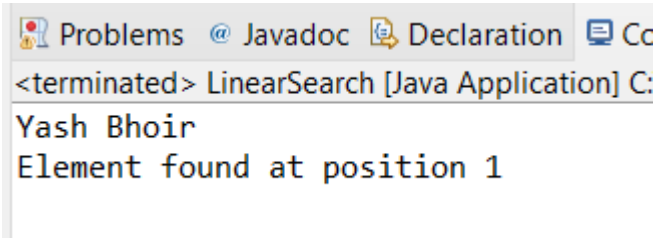
```
    }

    public static void main(String[] args)
    {
        ShellSort sorter = new ShellSort();
        int[] data = {41, 15, 82, 5, 65, 19, 32, 43, 8};
        sorter.sort(data);
        System.out.println("Sorted Array in Ascending Order: ");
        for (int num : data)
        {
            System.out.print(num + " ");
        }
    }

}
```



1.5: Linear Search

```
package ds_java;
import java.io.*;
public class LinearSearch {
        static int search(int arr[], int n, int x)
        {
                for (int i = 0; i < n; i++)
                {
                        if (arr[i] == x)
                                return i;
                }

                return -1;
        }

        public static void main(String[] args)
        {
                int[] arr = { 3, 4, 1, 7, 5 };
                int n = arr.length;

                int x = 4;
```

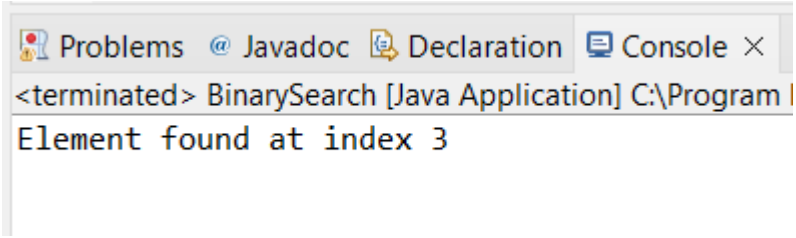
```
		int index = search(arr, n, x);
		if (index == -1)
			System.out.println("Element is not present in the array");
		else
			System.out.println("Element found at position " + index);
	}

}
```



1.6: Binary Search

```
package ds_java;
import java.io.*;
public class BinarySearch {
	int BinarySearch(int arr[], int l, int r, int x)
	{
		while (l <= r)
	  {
			int mid = (l + r) / 2;
			if (arr[mid] == x)
		{
				return mid;
			}
		else if (arr[mid] > x)
		{
				r = mid - 1;
			}
		else
		{
			l = mid + 1;
			}
		  }
		  return -1;
	}
	public static void main(String args[])
	{
		BinarySearch ob = new BinarySearch();
```

```
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
        int result = ob.BinarySearch(arr, 0, n - 1, x);

        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index "
                                                + result);
    }

}
```



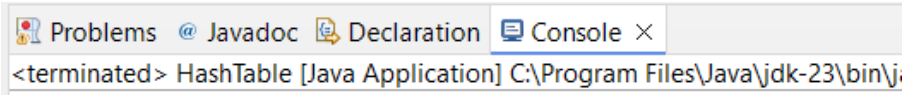
PRACTICAL 2: Perform various hashing techniques with Linear Probe as collision resolution.

2.1: Modulo Division with linear probe

```
package ds_java;

public class HashTable {
	private int[] table;
	private int size;

	// Constructor to initialize hash table with a specific size
	public HashTable(int size) {
	this.size = size;
	table = new int[size];
	// Initialize the table with -1 to indicate empty slots
	for (int i = 0; i < size; i++) {
	table[i] = -1;
	}
	}

	// Hash function using modulo division method
	public int hashFunction(int key) {
	return key % size;  // Return index by key % table_size
	}

	// Insert method using linear probing for collision resolution
	public void insert(int key) {
	int index = hashFunction(key);

	// Linear probing to find an empty slot
	while (table[index] != -1) {
	index = (index + 1) % size;  // Move to next slot in a circular way
	}

	table[index] = key;
	System.out.println("Inserted key " + key + " at index " + index);
	}

	// Search method to find a key in the hash table
	public int search(int key) {
	int index = hashFunction(key);
	int startIndex = index; // To avoid infinite loop in case we cycle through all slots

	// Linear probing to search for the key
```

```
while (table[index] != -1) {
if (table[index] == key) {
return index;  // Found the key at index
}
index = (index + 1) % size;  // Move to the next slot
if (index == startIndex) {
break;  // We've looped through the entire table
}
}

return -1;  // Key not found
}

// Delete method to remove a key from the hash table
public void delete(int key) {
int index = search(key);

if (index != -1) {
table[index] = -1;  // Mark the slot as empty
System.out.println("Deleted key " + key + " from index " + index);
} else {
System.out.println("Key " + key + " not found");
}
}

// Display the current state of the hash table
public void display() {
System.out.println("Hash Table:");
for (int i = 0; i < size; i++) {
if (table[i] == -1) {
System.out.println("Index " + i + ": Empty");
} else {
System.out.println("Index " + i + ": " + table[i]);
}
}
}

// Main method to demonstrate hashing using modulo division
public static void main(String[] args) {
HashTable hashTable = new HashTable(10);

// Insert keys into the hash table
hashTable.insert(10);
hashTable.insert(20);
```

```
        hashTable.insert(30);
        hashTable.insert(25);
        hashTable.insert(35);

        // Display the hash table
        hashTable.display();

        // Search for a key
        System.out.println("Search for key 25: Index " + hashTable.search(25));
        System.out.println("Search for key 99: Index " + hashTable.search(99));

        // Delete a key
        hashTable.delete(25);
        hashTable.display();
        }
```




```
Yash Bhoir
Inserted key 10 at index 0
Inserted key 20 at index 1
Inserted key 30 at index 2
Inserted key 25 at index 5
Inserted key 35 at index 6
Hash Table:
Index 0: 10
Index 1: 20
Index 2: 30
Index 3: Empty
Index 4: Empty
Index 5: 25
Index 6: 35
Index 7: Empty
Index 8: Empty
Index 9: Empty
Search for key 25: Index 5
Search for key 99: Index -1
Deleted key 25 from index 5
Hash Table:
Index 0: 10
Index 1: 20
Index 2: 30
Index 3: Empty
Index 4: Empty
Index 5: Empty
Index 6: 35
Index 7: Empty
Index 8: Empty
Index 9: Empty
}
```

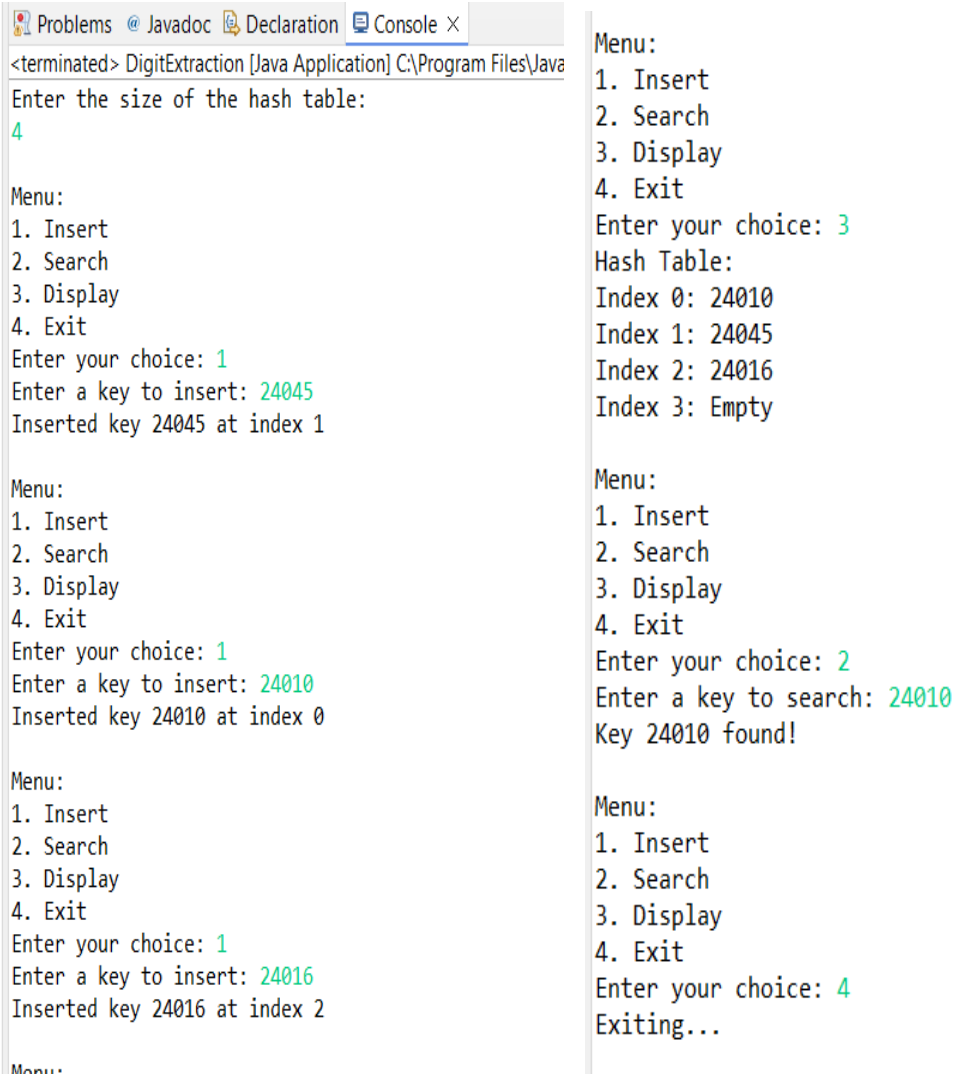
2.2: Digit extraction with linear probe.

```
package ds_java;

import java.util.Scanner;

class HashTableTemp {
   int[] table;
   int size;
   HashTableTemp(int size) {
         this.size = size;
         this.table = new int[size];
         for (int i = 0; i < size; i++) {
                  table[i] = -1;
      }
   }
   int digitExtractionHash(int key) {
         int extractedDigit = key % 10;
         return extractedDigit % size;
   }
   void insert(int key) {
         int hashIndex = digitExtractionHash(key);
         int originalIndex = hashIndex;
         while (table[hashIndex] != -1) {
                  hashIndex = (hashIndex + 1) % size;
                  if (hashIndex == originalIndex) {
            System.out.println("Hash table is full! Cannot insert " + key);
            return;
          }
      }
      table[hashIndex] = key;
      System.out.println("Inserted key " + key + " at index " + hashIndex);
   }
```

```
    boolean search(int key) {
            int hashIndex = digitExtractionHash(key);
            int originalIndex = hashIndex;

        while (table[hashIndex] != -1) {
            if (table[hashIndex] == key) {
              return true;
            }
            hashIndex = (hashIndex + 1) % size;
            if (hashIndex == originalIndex) {
              break;
            }
        }
        return false;
    }
    void display() {
        System.out.println("Hash Table:");
        for (int i = 0; i < size; i++) {
            System.out.println("Index " + i + ": " + (table[i] == -1 ? "Empty" : table[i]));
        }
    }
}
public class DigitExtraction {
            public static void main(String[] args) {
                        Scanner scanner = new Scanner(System.in);
                        System.out.println("Enter the size of the hash table:");
                        int size = scanner.nextInt();
                        HashTableTemp hashTable = new HashTableTemp(size);
        while (true) {
```

```
System.out.println("\nMenu:");
System.out.println("1. Insert");
System.out.println("2. Search");
System.out.println("3. Display");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
int choice = scanner.nextInt();
switch (choice) {
 case 1:
      System.out.print("Enter a key to insert: ");
      int key = scanner.nextInt();
      hashTable.insert(key);
      break;
   case 2:
      System.out.print("Enter a key to search: ");
      key = scanner.nextInt();
      boolean found = hashTable.search(key);
      System.out.println("Key " + key + (found ? " found!" : " not found!"));
      break;
   case 3:
      hashTable.display();
      break;
   case 4:
      System.out.println("Exiting...");
      scanner.close();
      return;
   default:
      System.out.println("Invalid choice! Please try again.");
}
```

```
    }
  }
}
```



```
Problems  Javadoc  Declaration  Console
<terminated> DigitExtraction [Java Application] C:\Program Files\Java
Enter the size of the hash table:
4

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 1
Enter a key to insert: 24045
Inserted key 24045 at index 1

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 1
Enter a key to insert: 24010
Inserted key 24010 at index 0

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 1
Enter a key to insert: 24016
Inserted key 24016 at index 2

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 3
Hash Table:
Index 0: 24010
Index 1: 24045
Index 2: 24016
Index 3: Empty

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 2
Enter a key to search: 24010
Key 24010 found!

Menu:
1. Insert
2. Search
3. Display
4. Exit
Enter your choice: 4
Exiting...
```

PRACTICAL 3: Implementation of stacks and queues using arrays.

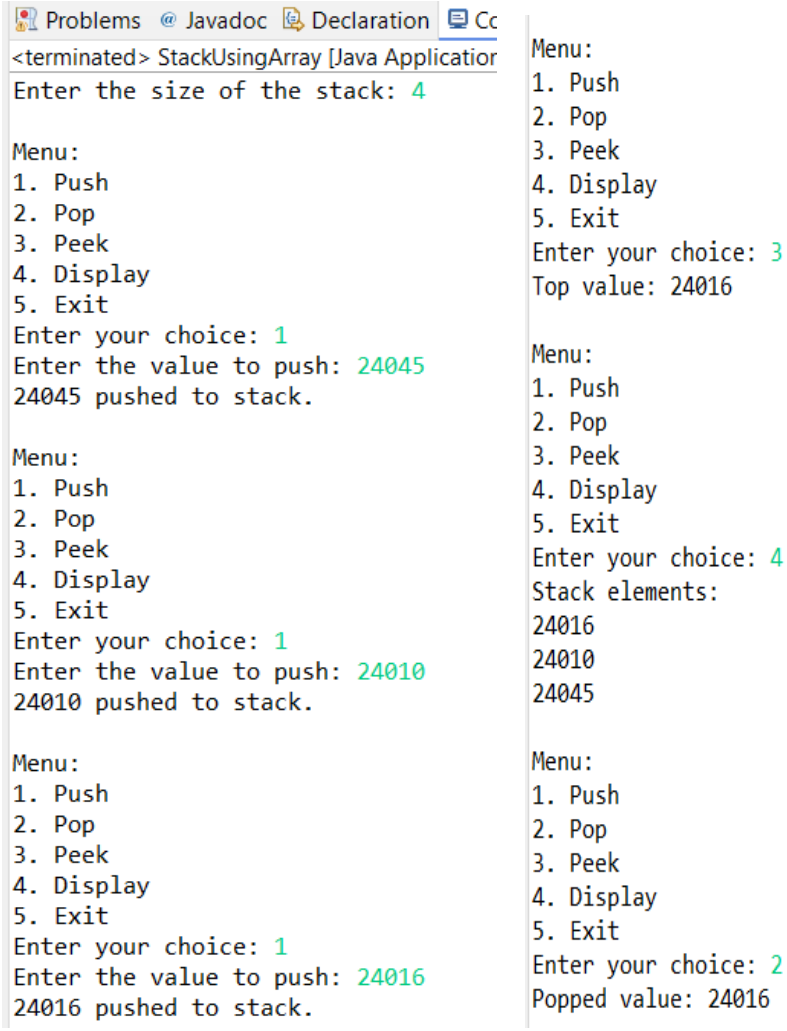
3.1: Stacks

```
package ds_java;
import java.util.Scanner;
class Stack {
        private final int[] stackArray;
        private final int capacity;
        private int top;
        public Stack(int size) {
                capacity = size;
                stackArray = new int[capacity];
    top = -1;
  }
  public void push(int value) {
    if (isFull()) {
      System.out.println("Stack is full " + value);
      return;
    }
    stackArray[++top] = value;
    System.out.println(value + " pushed to stack.");
  }
  public int pop() {
       if (isEmpty()) {
                System.out.println("Stack is Empty");
      return -1;
    }
    return stackArray[top--];
  }
  public int peek() {
```

```
            if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }
        return stackArray[top];
    }
    public boolean isEmpty() {
            return top == -1;
    }
    public boolean isFull() {
            return top == capacity - 1;
    }
    public void display() {
            if (isEmpty()) {
            System.out.println("Stack is empty!");
            return;
        }
        System.out.println("Stack elements:");
        for (int i = top; i >= 0; i--) {
            System.out.println(stackArray[i]);
        }
    }
}
public class StackUsingArray {
            public static void main(String[] args) {
                        Scanner scanner = new Scanner(System.in);
                        System.out.print("Enter the size of the stack: ");
                        int size = scanner.nextInt();
                        Stack stack = new Stack(size);
```

```
            while (true) {
System.out.println("\nMenu:");
System.out.println("1. Push");
System.out.println("2. Pop");
System.out.println("3. Peek");
System.out.println("4. Display");
System.out.println("5. Exit");
System.out.print("Enter your choice: ");
int choice = scanner.nextInt();
switch (choice) {
case 1 -> {
      System.out.print("Enter the value to push: ");
      int value = scanner.nextInt();
      stack.push(value);
}
case 2 -> {
 int poppedValue = stack.pop();
 if (poppedValue != -1) {
         System.out.println("Popped value: " + poppedValue);
   }
}
case 3 -> {
 int topValue = stack.peek();
 if (topValue != -1) {
         System.out.println("Top value: " + topValue);
   }
}
case 4 -> stack.display();
case 5 -> {
```

```
            System.out.println("Exiting...");
            scanner.close();
            return;
        }
        default -> System.out.println("Invalid choice! Please try again.");
        }
    }
  }
}
```



3.2: Queue

```
package ds_java;

import java.io.*;
import java.util.Scanner;

class Queue {
	private static int front, rear, capacity, count;
	private static int queue[];

	Queue(int size) {
		front = rear = -1;
		capacity = size;
		count = 0;
		queue = new int[capacity];
	}

	static void queueEnqueue(int item) {
		if (capacity == count) {
			System.out.printf("\nQueue is full\n");
			return;
		} else {
			rear++;
			if (count == 0)
				front = 0;
			queue[rear] = item;

			count++;
			System.out.println("Count is:" + count);
```

```
            }
            return;
        }

        static void queueDequeue() {
            if (front == -1) {
                System.out.printf("\nQueue is empty\n");
                return;
            } else {
                int dnum = queue[front];
                queue[front] = 0;
                front++;
                count--;
                System.out.println(dnum);
                System.out.println("Count is:" + count);
            }
            return;
        }

        static void queueDisplay() {
            int i;
            if (front == -1) {
                System.out.printf("Queue is Empty\n");
                return;
            }
            for (i = 0; i <= rear; i++) {
                System.out.printf(" %d , ", queue[i]);
                System.out.println("Count is:" + count);
            }
```

```
            return;
        }

        static void queueFront() {
            if (front == rear) {
                System.out.printf("Queue is Empty\n");
                return;
            }
            System.out.printf("\nFront Element of the queue: %d", queue[front]);
            return;
        }
}

public class QueueUsingArray {
    public static void main(String[] args) throws IOException {
        Queue q = new Queue(4);
        int choice;
        Scanner scanner = new Scanner(System.in);
        BufferedReader bfn = new BufferedReader(new InputStreamReader(System.in));
        do {
            System.out.println("Menu:");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Queue Display");
            System.out.println("4. Quit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
            case 1:
```

```
                    System.out.println("Enter Queue Element:");
                    int it = Integer.parseInt(bfn.readLine());
                    q.queueEnqueue(it);
                    break;
                case 2:
                    System.out.println("Dequeued Elements are:");
                    q.queueDequeue();
                    break;
                case 3:
                    System.out.println("Queue Elements are:");
                    q.queueDisplay();
                    break;
                default:
                    System.out.println("Invalid choice. Please choose a valid option.");
                }
            } while (choice != 4);
        }

}
```

Problems @ Javadoc Declaratio

<terminated> QueueUsingArray [Java Ap

```
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 1
Enter Queue Element:
24045
Count is:1
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 1
Enter Queue Element:
24010
Count is:2
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 1
Enter Queue Element:
24016
Count is:3
```

```
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 2
Dequeued Elements are:
24045
Count is:2
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 3
Queue Elements are:
 0 , Count is:2
 24010 , Count is:2
 24016 , Count is:2
Menu:
1. Enqueue
2. Dequeue
3. Queue Display
4. Quit
Enter your choice: 4
Invalid choice. Please choose a valid option.
```

PRACTICAL 5: Implement all different types of LinkedList.

5.1: Singly LinkedList

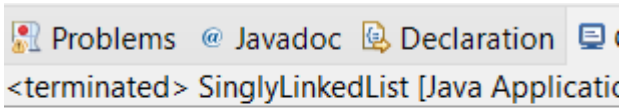
```
package ds_java;


public class SinglyLinkedList {


	Node head;


	class Node {
		int data;
		Node next;


		Node(int d) {
			data = d;
			next = null;
		}
	}


	public void insertAtBeginning(int new_data) {
		Node new_node = new Node(new_data);
		new_node.next = head;
		head = new_node;
	}


	public void insertAfter(Node prev_node, int new_data) {
		if (prev_node == null) {
			System.out.println("The given previous node cannot be null");
			return;
		}
```

```
        Node new_node = new Node(new_data);
        new_node.next = prev_node.next;
        prev_node.next = new_node;
    }

    public void insertAtEnd(int new_data) {
        Node new_node = new Node(new_data);
        if (head == null) {
            head = new Node(new_data);
            return;
        }
        new_node.next = null;
        Node last = head;
        while (last.next != null)
            last = last.next;
        last.next = new_node;
        return;
    }

    void deleteNode(int position) {
        if (head == null)
            return;
        Node temp = head;
        if (position == 0) {
            head = temp.next;
            return;
        }
        for (int i = 0; temp != null && i < position - 1; i++)
            temp = temp.next;
```

```
        if (temp == null || temp.next == null)
                return;
        Node next = temp.next.next;
        temp.next = next;
    }

    boolean search(Node head, int key) {
        Node current = head;
        while (current != null) {
            if (current.data == key)
                return true;
            current = current.next;
        }
        return false;
    }

    void sortLinkedList(Node head) {
        Node current = head;
        Node index = null;
        int temp;
        if (head == null) {
            return;
        } else {
            while (current != null) {
                index = current.next;
                while (index != null) {
                    if (current.data > index.data) {
                        temp = current.data;
                        current.data = index.data;
```

```
                                        index.data = temp;
                                }
                                index = index.next;
                        }
                        current = current.next;
                }
        }
}

public void printList() {
        Node tnode = head;
        while (tnode != null) {
                System.out.print(tnode.data + " ");
                tnode = tnode.next;
        }
}

public static void main(String[] args) {
        SinglyLinkedList llist = new SinglyLinkedList();
        llist.insertAtEnd(1);
        llist.insertAtBeginning(2);
        llist.insertAtBeginning(3);
        llist.insertAtEnd(4);
        llist.insertAfter(llist.head.next, 5);
        System.out.println("Linked list: ");
        llist.printList();
        System.out.println("\nAfter deleting an element: ");
        llist.deleteNode(3);
        llist.printList();
```

```
            System.out.println();
            System.out.println("Searching an element");
            int item_to_find = 3;
            if (llist.search(llist.head, item_to_find))
                System.out.println(item_to_find + " is found");
            else
                System.out.println(item_to_find + " is not found");

            llist.sortLinkedList(llist.head);
            System.out.println("Sorted List: ");
            llist.printList();
        }
}
```




```
Linked list:
3 2 5 1 4
After deleting an element:
3 2 5 4
Searching an element
3 is found
Sorted List:
2 3 4 5
```

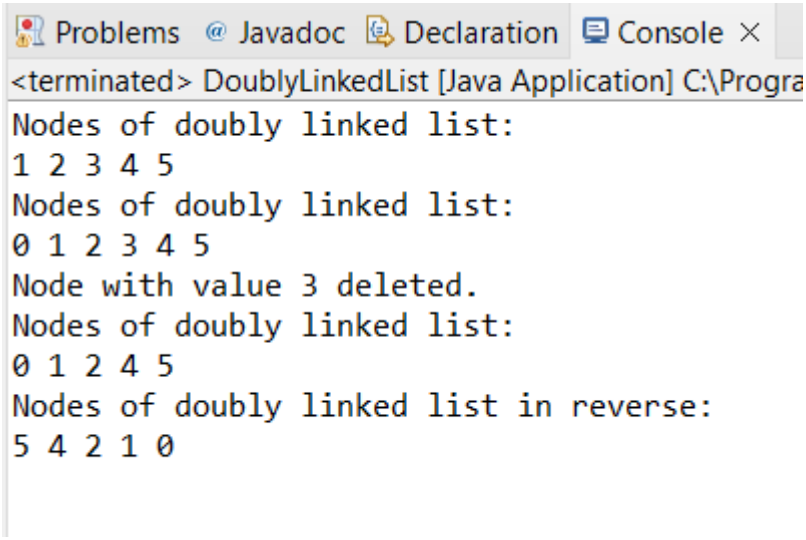
5.2: Doubly LinkedList

```
package ds_java;


public class DoublyLinkedList {
	class Node {
		int data;
		Node previous;
		Node next;

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

	Node head, tail = null;

	public void addNode(int data) {
		Node newNode = new Node(data);
		if (head == null) {
			head = tail = newNode;
			head.previous = null;
			tail.next = null;
		} else {
			tail.next = newNode;
			newNode.previous = tail;
			tail = newNode;
			tail.next = null;
		}
	}
```

```
public void addNodeAtStart(int data) {
        Node newNode = new Node(data);
        if (head == null) {
                head = tail = newNode;
                head.previous = null;
                tail.next = null;
        } else {
                newNode.next = head;
                head.previous = newNode;
                head = newNode;
        }
}

public void deleteNode(int value) {
        if (head == null) {
                System.out.println("List is empty.");
                return;
        }
        Node current = head;
        while (current != null) {
                if (current.data == value) {
                        if (current == head && current == tail) {
                                head = tail = null;
                        } else if (current == head) {
                                head = head.next;
                                head.previous = null;
                        } else if (current == tail) {
                                tail = tail.previous;
```

```
                    tail.next = null;
                } else {
                    current.previous.next = current.next;
                    current.next.previous = current.previous;
                }
                System.out.println("Node with value " + value + " deleted.");
                return;
            }
            current = current.next;
        }
        System.out.println("Node with value " + value + " not found.");
    }

    public void display() {
        Node current = head;
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        System.out.println("Nodes of doubly linked list: ");
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public void displayReverse() {
        Node current = tail;
```

```
        if (tail == null) {
            System.out.println("List is empty.");
            return;
        }
        System.out.println("Nodes of doubly linked list in reverse: ");
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.previous;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        DoublyLinkedList dList = new DoublyLinkedList();
        dList.addNode(1);
        dList.addNode(2);
        dList.addNode(3);
        dList.addNode(4);
        dList.addNode(5);

        dList.display();
        dList.addNodeAtStart(0);
        dList.display();

        dList.deleteNode(3);
        dList.display();
        dList.displayReverse();
    }
}
```


Problems
Javadoc
Declaration
Console
<terminated> DoublyLinkedList [Java Application] C:\Progra
Nodes of doubly linked list:
1 2 3 4 5
Nodes of doubly linked list:
0 1 2 3 4 5
Node with value 3 deleted.
Nodes of doubly linked list:
0 1 2 4 5
Nodes of doubly linked list in reverse:
5 4 2 1 0

PRACTICAL 7: Create and perform various operations on BST.

```
package ds_java;

import java.util.Scanner;

class Node {
	int data;
	Node left, right;

	public Node(int item) {
		data = item;
		left = right = null;
	}
}

class BinarySearchTreeTemp {
	Node root;
	int count;

	public BinarySearchTreeTemp() {
		root = null;
		count = 0;
	}

	public void insert(int data) {
		root = insertRec(root, data);
		count++;
	}
```

```
    private Node insertRec(Node root, int data) {
        if (root == null) {
            root = new Node(data);
            return root;
        }
        if (data < root.data)
            root.left = insertRec(root.left, data);
        else
            root.right = insertRec(root.right, data);
        return root;
    }

    public void preorder() {
        preorderRec(root);
    }

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

    public void inorder() {
        inorderRec(root);
    }

    private void inorderRec(Node root) {
```

```
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.data + " ");
            inorderRec(root.right);
        }
    }

    public void postorder() {
        postorderRec(root);
    }

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

    public Node search(int key) {
        return searchRec(root, key);
    }

    private Node searchRec(Node root, int key) {
        if (root == null || root.data == key)
            return root;
        if (key < root.data)
            return searchRec(root.left, key);
        return searchRec(root.right, key);
```

```
    }

    public void deleteKey(int key) {
        root = deleteRec(root, key);
    }

    private Node deleteRec(Node root, int key) {
        if (root == null) {
            System.out.println("Element not present in the tree.");
            return root;
        }
        if (key < root.data)
            root.left = deleteRec(root.left, key);
        else if (key > root.data)
            root.right = deleteRec(root.right, key);
        else {
            if (root.left == null) {
                count--;
                return root.right;
            } else if (root.right == null) {
                count--;
                return root.left;
            }
            root.data = minValue(root.right);
            root.right = deleteRec(root.right, root.data);
        }
        return root;
    }
```

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

    public void findSmallest() {
        if (root == null) {
            System.out.println("The tree is empty.");
            return;
        }
        Node current = root;
        while (current.left != null)
            current = current.left;
        System.out.println("Smallest node is: " + current.data);
    }

    public void findLargest() {
        if (root == null) {
            System.out.println("The tree is empty.");
            return;
        }
        Node current = root;
        while (current.right != null)
            current = current.right;
        System.out.println("Largest node is: " + current.data);
```

```
        }

        public void countNodes() {
                System.out.println("Total number of nodes: " + count);
        }
}

public class BinarySearchTree {
        public static void main(String[] args) {
                BinarySearchTreeTemp  btree = new BinarySearchTreeTemp ();
                Scanner scanner = new Scanner(System.in);
                int choice, data;
                boolean continueInput = true;
                System.out.println("Binary Search Tree Operation");
                while (continueInput) {
                        System.out.print("Enter a value to insert: ");
                        data = scanner.nextInt();
                        btree.insert(data);
                        System.out.print("Press 0 to add new node ");
                        choice = scanner.nextInt();
                        if (choice != 0)
                                continueInput = false;
                }
                while (true) {
                        System.out.println("1. Pre-order Traversal");
                        System.out.println("2. In-order Traversal");
                        System.out.println("3. Post-order Traversal");
                        System.out.println("4. Search an Element");
                        System.out.println("5. Delete an Element");
```

```
System.out.println("6. Find Smallest Element");
System.out.println("7. Find Largest Element");
System.out.println("8. Count Nodes");
System.out.println("9. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();
switch (choice) {
case 1:
        System.out.print("Pre-order Traversal: ");
        btree.preorder();
        System.out.println();
        break;
case 2:
        System.out.print("In-order Traversal: ");
        btree.inorder();
        System.out.println();
        break;
case 3:
        System.out.print("Post-order Traversal: ");
        btree.postorder();
        System.out.println();
        break;
case 4:
        System.out.print("Enter search value: ");
        data = scanner.nextInt();
        Node result = btree.search(data);
        if (result != null)
                System.out.println("Element " + result.data + " found in the
tree.");
        else
```

```
                                System.out.println("Element " + data + " not found in the
tree.");
                                break;
                        case 5:
                                System.out.print("Enter the element you wish to delete: ");
                                data = scanner.nextInt();
                                btree.deleteKey(data);
                                break;
                        case 6:
                                btree.findSmallest();
                                break;
                        case 7:
                                btree.findLargest();
                                break;
                        case 8:
                                btree.countNodes();
                                break;
                        case 9:
                                System.out.println("Exiting...");
                                scanner.close();
                                System.exit(0);
                        default:
                                System.out.println("Invalid choice! Please try again.");
                        }
                }
        }
}
```

```
<terminated> BinarySearchTree (1) [Java A
Binary Search Tree Operation
Enter a value to insert: 24045
Press 0 to add new node 0
Enter a value to insert: 24010
Press 0 to add new node 0
Enter a value to insert: 24016
Press 0 to add new node 0
Enter a value to insert: 24036
Press 0 to add new node
```

```
1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal
4. Search an Element
5. Delete an Element
6. Find Smallest Element
7. Find Largest Element
8. Count Nodes
9. Exit
Enter your choice: 1
Pre-order Traversal: 24045 24010 24016 24036
```

```
1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal
4. Search an Element
5. Delete an Element
6. Find Smallest Element
7. Find Largest Element
8. Count Nodes
9. Exit
Enter your choice: 2
In-order Traversal: 24010 24016 24036 24045
```

```
1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal
4. Search an Element
5. Delete an Element
6. Find Smallest Element
7. Find Largest Element
8. Count Nodes
9. Exit
Enter your choice: 3
Post-order Traversal: 24036 24016 24010 24045
```

```
1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal
4. Search an Element
5. Delete an Element
6. Find Smallest Element
7. Find Largest Element
8. Count Nodes
9. Exit
Enter your choice: 4
Enter search value: 24045
```

```
1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal
4. Search an Element
5. Delete an Element
6. Find Smallest Element
7. Find Largest Element
8. Count Nodes
9. Exit
Enter your choice: 5
Enter the element you wish to delete: 24010
```

```
Enter your choice: 6
Smallest node is: 24016
```

```
Enter your choice: 7
Largest node is: 24045
```

```
Enter your choice: 8
Total number of nodes: 3
```

```
Enter your choice: 9
Exiting...
```

PRACTICAL 8: Implementing Heap with different operations.

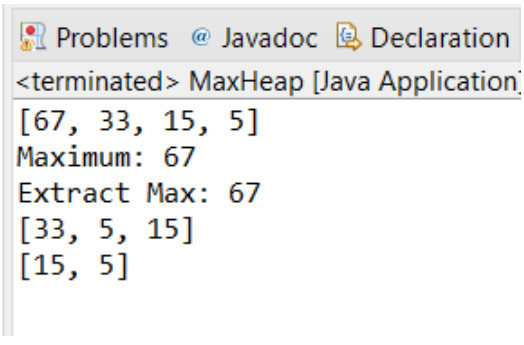
8.1: MaxHeap

```
package ds_java;

import java.util.ArrayList;

public class MaxHeap {
	private ArrayList<Integer> heap;

	public MaxHeap() {
		this.heap = new ArrayList<>();
	}

	private int parent(int index) {
		return (index - 1) / 2;
	}

	private int leftChild(int index) {
		return 2 * index + 1;
	}

	private int rightChild(int index) {
		return 2 * index + 2;
	}

	private void swap(int i, int j) {
		int temp = heap.get(i);
		heap.set(i, heap.get(j));
		heap.set(j, temp);
```

```
    }

    private void heapifyUp(int index) {
        int parent = parent(index);
        if (index > 0 && heap.get(index) > heap.get(parent)) {
            swap(index, parent);
            heapifyUp(parent);
        }
    }

    private void heapifyDown(int index) {
        int left = leftChild(index);
        int right = rightChild(index);
        int largest = index;
        if (left < heap.size() && heap.get(left) > heap.get(largest)) {
            largest = left;
        }
        if (right < heap.size() && heap.get(right) > heap.get(largest)) {
            largest = right;
        }
        if (largest != index) {
            swap(index, largest);
            heapifyDown(largest);
        }
    }

    public void insert(int key) {
        heap.add(key);
        heapifyUp(heap.size() - 1);
```

```
    }

    public int extractMax() {
        if (heap.isEmpty()) {
            throw new IllegalStateException("Heap is empty");
        }
        if (heap.size() == 1) {
            return heap.remove(0);
        }
        int max = heap.get(0);
        heap.set(0, heap.remove(heap.size() - 1));
        heapifyDown(0);
        return max;
    }

    public int getMax() {
        if (heap.isEmpty()) {
            throw new IllegalStateException("Heap is empty");
        }
        return heap.get(0);
    }

    public void delete(int key) {
        int index = heap.indexOf(key);
        if (index == -1) {
            throw new IllegalStateException("Key not found");
        }
        heap.set(index, heap.get(heap.size() - 1));
        heap.remove(heap.size() - 1);
```

```
            if (index < heap.size()) {
                heapifyDown(index);
                heapifyUp(index);
            }
        }

        public void printHeap() {
            System.out.println(heap);
        }

        public static void main(String[] args) {
            MaxHeap heap = new MaxHeap();
            heap.insert(5);
            heap.insert(15);
            heap.insert(67);
            heap.insert(33);
            heap.printHeap();
            System.out.println("Maximum: " + heap.getMax());
            System.out.println("Extract Max: " + heap.extractMax());
            heap.printHeap();
            heap.delete(33);
            heap.printHeap();
        }
}
```



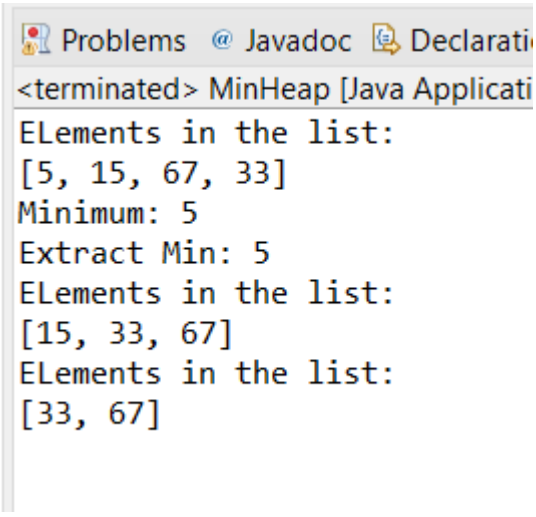
8.2: MinHeap

```
package ds_java;

import java.util.ArrayList;

public class MinHeap {
	private ArrayList<Integer> heap;

	public MinHeap() {
		this.heap = new ArrayList<>();
  }

	private int parent(int index) {
		return (index - 1) / 2;
	}

	private int leftChild(int index) {
		return 2 * index + 1;
	}

	private int rightChild(int index) {
		return 2 * index + 2;
	}

	private void swap(int i, int j) {
		int temp = heap.get(i);
		heap.set(i, heap.get(j));
		heap.set(j, temp);
	}
```

```
private void heapifyUp(int index) {
        int parent = parent(index);
        if (index > 0 && heap.get(index) < heap.get(parent)) {
                swap(index, parent);
                heapifyUp(parent);
        }
}

private void heapifyDown(int index) {
        int left = leftChild(index);
        int right = rightChild(index);
        int smallest = index;
        if (left < heap.size() && heap.get(left) < heap.get(smallest)) {
                smallest = left;
        }
        if (right < heap.size() && heap.get(right) < heap.get(smallest)) {
                smallest = right;
        }
        if (smallest != index) {
                swap(index, smallest);
                heapifyDown(smallest);
        }
}

public void insert(int key) {
        heap.add(key);
        heapifyUp(heap.size() - 1);
}
```

```
public int extractMin() {
        if (heap.isEmpty()) {
                throw new IllegalStateException("Heap is empty");
        }
        if (heap.size() == 1) {
                return heap.remove(0);
        }
        int min = heap.get(0);
        heap.set(0, heap.remove(heap.size() - 1));
        heapifyDown(0);
        return min;
}
public int getMin() {
        if (heap.isEmpty()) {
                throw new IllegalStateException("Heap is empty");
        }
        return heap.get(0);
}
public void delete(int key) {
        int index = heap.indexOf(key);
        if (index == -1) {
                throw new IllegalStateException("Key not found");
        }
        heap.set(index, heap.get(heap.size() - 1));
        heap.remove(heap.size() - 1);
        if (index < heap.size()) {
                heapifyDown(index);
                heapifyUp(index);
```

```
            }
        }
        public void printHeap() {
            System.out.println("ELements in the list:");
            System.out.println(heap);
        }
        public static void main(String[] args) {
            MinHeap heap = new MinHeap();
            heap.insert(15);
            heap.insert(5);
            heap.insert(67);
            heap.insert(33);
            heap.printHeap();
            System.out.println("Minimum: " + heap.getMin());
            System.out.println("Extract Min: " + heap.extractMin());
            heap.printHeap();
            heap.delete(15);
            heap.printHeap();
        }
}
```



PRACTICAL 9: **Create a Graph storage Structure using Adjacency Matrix**

```
package ds_java;

import java.util.*;

public class AdjacencyMatrixGraph {
	private int[][] adjMatrix;
	private Map<Integer, Integer> vertexMap; // Maps user-defined vertices to indices

	public AdjacencyMatrixGraph(int vertexCount) {
		adjMatrix = new int[vertexCount][vertexCount];
		vertexMap = new HashMap<>();
	}

	public void addVertex(int vertex, int index) {
		vertexMap.put(vertex, index); // Map user-defined vertex to zero-based index
	}

	public void addEdge(int from, int to) {
		Integer fromIndex = vertexMap.get(from);
		Integer toIndex = vertexMap.get(to);
		if (fromIndex == null || toIndex == null) {
			throw new IllegalArgumentException("Vertex not found: from = " + from + ",
to = " + to);
		}

		adjMatrix[fromIndex][toIndex] = 1;
		adjMatrix[toIndex][fromIndex] = 1; // For undirected graph
	}
```

```
    public void printAdjMatrix() {
        for (int[] row : adjMatrix) {
            for (int cell : row) {
                System.out.print(cell + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        int vertexCount = sc.nextInt();
        AdjacencyMatrixGraph graph = new AdjacencyMatrixGraph(vertexCount);
        System.out.println("Enter the vertices:");
        for (int i = 0; i < vertexCount; i++) {
            int vertex = sc.nextInt();
            graph.addVertex(vertex, i); // Map vertex to index
        }
        System.out.print("Enter the number of edges: ");
        int edgeCount = sc.nextInt();

        System.out.println("Enter the edges (from and to) as vertex values:");
        for (int i = 0; i < edgeCount; i++) {
            System.out.print("Enter from vertex: ");
            int from = sc.nextInt();
            System.out.print("Enter to vertex: ");
            int to = sc.nextInt();
            graph.addEdge(from, to);
```

```
            }
            System.out.println("Adjacency Matrix:");
            graph.printAdjMatrix();
        }

}
```

```
<terminated> Adjmatgraph [Java Application] C:\Users\aleen\.p2\pc
Enter the number of vertices: 4
Enter the vertices:
2
4
5
6
Enter the number of edges: 3
Enter the edges (from and to) as vertex values:
Enter from vertex: 2
Enter to vertex: 4
Enter from vertex: 4
Enter to vertex: 5
Enter from vertex: 5
Enter to vertex: 6
Adjacency Matrix:
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
```

PRACTICAL 10: Implementation of Graph traversal.

10.1: DFS (Depth-First Search)

```
package ds_java;
import java.util.*;
public class DepthFirstSearch {
	static class Stack {
		private final List elements;

		public Stack() {
			elements = new ArrayList();
		}

		public void push(Object element) {
			elements.add(element);
		}

		public Object pop() {
			if (isEmpty()) {
				throw new EmptyStackException();
			}
			return elements.remove(elements.size() - 1);
		}

		public boolean isEmpty() {
			return elements.isEmpty();
		}
	}
```

```
    public static void dfsUsingCustomStack(Map<Character, List<Character>> graph, char
startNode) {
        Set<Character> visited = new HashSet<>();
        Stack stack = new Stack();
        stack.push(startNode);
        System.out.println(startNode + " pushed to stack.");
        while (!stack.isEmpty()) {
            char currentNode = (char) stack.pop();
            if (!visited.contains(currentNode)) {
                System.out.print(currentNode + " ");
                visited.add(currentNode);
                List<Character> neighbors = graph.get(currentNode);
                if (neighbors != null) {
                    for (char neighbor : neighbors) {
                        if (!visited.contains(neighbor)) {
                            stack.push(neighbor);
                            System.out.println(neighbor + " pushed to
stack.");
                        }
                    }
                }
            }
        }

    public static void main(String[] args) {

        Map<Character, List<Character>> graph = new HashMap<>();
        graph.put('A', Arrays.asList('B', 'C'));
        graph.put('B', Arrays.asList('D', 'E'));
```

```
        graph.put('C', Arrays.asList('F'));
        graph.put('D', new ArrayList<>());
        graph.put('E', Arrays.asList('F'));
        graph.put('F', new ArrayList<>());
        System.out.println("DFS starting from node A:");
        dfsUsingCustomStack(graph, 'A');
    }
}
```

```
<terminated> Dfs [Java Application] C:\Users\aleen
DFS starting from node A:
A pushed to stack.
A B pushed to stack.
C pushed to stack.
C F pushed to stack.
F B D pushed to stack.
E pushed to stack.
E D
```

10.2: BFS (Breadth-First Search)

```
package ds_java;
import java.util.Queue;
import java.util.*;
public class BreadthFirstSearch {
   public static void bfsUsingQueue(Map<Character, List<Character>> graph, char startNode) {
      Set<Character> visited = new HashSet<>();
      Queue<Character> queue = new LinkedList<>();
      queue.add(startNode);
      visited.add(startNode);
      while (!queue.isEmpty()) {
         char node = queue.poll();
         System.out.print(node + " ");
```

```
            // Null check for neighbors
            List<Character> neighbors = graph.get(node);
            if (neighbors != null) {
                for (char neighbor : neighbors) {
                    if (!visited.contains(neighbor)) {
                        queue.add(neighbor);
                        visited.add(neighbor);
                    }
                }
            }
        }
    }
    public static void main(String[] args) {
        Map<Character, List<Character>> graph = new HashMap<>();
        graph.put('A', Arrays.asList('B', 'C'));
        graph.put('B', Arrays.asList('D', 'E'));
        graph.put('C', Arrays.asList('F'));
        graph.put('D', new ArrayList<>());
        graph.put('E', Arrays.asList('F'));
        graph.put('F', new ArrayList<>());
        System.out.println("BFS starting from node A:");
        bfsUsingQueue(graph, 'A');
    }
}
```

```
<terminated> Bfs [Java Application] C:\Users\aleen\.p2
BFS starting from node A:
A B C D E F
```

PRACTICAL 11: Create a minimum spanning tree using any method Kruskal's Algorithm or Prim's Algorithm.

11.1: Minimum Spanning Tree Using - Kruskal Algorithm

```
package ds_java;


import java.util.*;


class Edge implements Comparable<Edge> {
	int src, dest, weight;


	Edge(int src, int dest, int weight) {
		this.src = src;
		this.dest = dest;
		this.weight = weight;
	}


	public int compareTo(Edge other) {
		return this.weight - other.weight;
	}
}


public class KruskalAlgorithm {
	int V;
	List<Edge> edges = new ArrayList<>();


	KruskalAlgorithm(int V) {
		this.V = V;
	}
```

```
    void addEdge(int src, int dest, int weight) {
        edges.add(new Edge(src, dest, weight));
    }

    int findParent(int[] parent, int v) {
        if (parent[v] != v) {
            parent[v] = findParent(parent, parent[v]);
        }
        return parent[v];
    }

    void union(int[] parent, int[] rank, int u, int v) {
        int rootU = findParent(parent, u);
        int rootV = findParent(parent, v);
        if (rank[rootU] < rank[rootV]) {
            parent[rootU] = rootV;
        } else if (rank[rootU] > rank[rootV]) {
            parent[rootV] = rootU;
        } else {
            parent[rootV] = rootU;
            rank[rootU]++;
        }
    }

    void displayAllEdges() {
        System.out.println("All edges in the graph:");
        System.out.println("Edge \tWeight");
        for (Edge edge : edges) {
            System.out.println(edge.src + " - " + edge.dest + "\t" + edge.weight);
```

```
        }
    }

    void kruskalMST() {
        Collections.sort(edges);
        int[] parent = new int[V];
        int[] rank = new int[V];
        for (int i = 0; i < V; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
        System.out.println("\nMinimum Spanning Tree:");
        System.out.println("Edge \tWeight");
        int edgeCount = 0;
        for (Edge edge : edges) {
            if (edgeCount == V - 1)
                break;
            int rootU = findParent(parent, edge.src);
            int rootV = findParent(parent, edge.dest);
            if (rootU != rootV) {
                System.out.println(edge.src + " - " + edge.dest + "\t" + edge.weight);
                union(parent, rank, rootU, rootV);
                edgeCount++;
            }
        }
    }

    public static void main(String[] args) {
        KruskalAlgorithm graph = new KruskalAlgorithm(4);
```

```
        graph.addEdge(0, 1, 10);
        graph.addEdge(0, 2, 6);
        graph.addEdge(0, 3, 5);
        graph.addEdge(1, 3, 15);
        graph.addEdge(2, 3, 4);
        graph.displayAllEdges();
        graph.kruskalMST();
    }

}
```

```
<terminated> Kruskal [Java Application] C:\Users\aleer
All edges in the graph:
Edge    Weight
0 - 1   10
0 - 2   6
0 - 3   5
1 - 3   15
2 - 3   4

Minimum Spanning Tree:
Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10
```

11.2: Minimum Spanning Tree Using – Prim's Algorithm

```
package ds_java;

import java.util.Arrays;

public class PrimsAlgorithm {
    static int V = 5;
```

```
int minKey(int[] key, boolean[] mstSet) {
        int min = Integer.MAX_VALUE, minIndex = -1;
        for (int v = 0; v < V; v++)
                if (!mstSet[v] && key[v] < min) {
                        min = key[v];
                        minIndex = v;
                }
        return minIndex;
}

void printGraph(int[][] graph) {
        System.out.println("Graph (Adjacency Matrix):");
        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        System.out.print(graph[i][j] + "\t");
                }
                System.out.println();
        }
}

void printMST(int[] parent, int[][] graph) {
        System.out.println("\nMinimum Spanning Tree:");
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
                System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
}

void primMST(int[][] graph) {
        int[] parent = new int[V];
```

```
        int[] key = new int[V];
        boolean[] mstSet = new boolean[V];
        Arrays.fill(key, Integer.MAX_VALUE);
        key[0] = 0;
        parent[0] = -1;
        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;
            for (int v = 0; v < V; v++)
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }
        printMST(parent, graph);
    }

    public static void main(String[] args) {
        PrimsAlgorithm t = new PrimsAlgorithm();
        int[][] graph = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 }, { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 },
                { 0, 5, 7, 9, 0 } };

        t.printGraph(graph);
        t.primMST(graph);
    }

}
```

```
<terminated> Primsalgo [Java Application] C:\Users\aleen\.p2\p
Graph (Adjacency Matrix):
0       2       0       6       0
2       0       3       8       5
0       3       0       0       7
6       8       0       0       9
0       5       7       9       0

Minimum Spanning Tree:
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

PRACTICAL 12: **Group project (3 to 4 members) to be given to work on one application to a real world problem.**

Project Title: Student Exam Result Management System

Description:

This project helps manage student exam records. You can add student details, sort results by marks using a Binary Search Tree (BST), and search for a student by their roll number using Binary Search. The system also tracks failed students and allows undoing incorrect entries using a Stack.

Code:

```
package ds_java;


import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;
import java.util.Stack;


public class ExamResultManagementSystem {
	// Student class
	static class Student {
		int rollNumber;
		String name;
		int marks;

		public Student(int rollNumber, String name, int marks) {
			this.rollNumber = rollNumber;
			this.name = name;
			this.marks = marks;
		}
```

```
        @Override
        public String toString() {
                return "Roll No: " + rollNumber + ", Name: " + name + ", Marks: " + marks;
        }
}

// Node class for Binary Search Tree (BST)
static class Node {
        Student student;
        Node left, right;

        public Node(Student student) {
                this.student = student;
        }
}

// Binary Search Tree for sorting students by marks
static class StudentBST {
        Node root;

        public void insert(Student student) {
                root = insertRec(root, student);
        }

        private Node insertRec(Node root, Student student) {
                if (root == null) {
                        root = new Node(student);
                        return root;
```

```
                }
                if (student.marks < root.student.marks) {
                        root.left = insertRec(root.left, student);
                } else if (student.marks > root.student.marks) {
                        root.right = insertRec(root.right, student);
                }
                return root;
        }

        public void inOrder() {
                if (root == null) {
                        System.out.println("No students in the system.");
                } else {
                        System.out.println("Students sorted by marks:");
                        inOrderRec(root);
                }
        }

        private void inOrderRec(Node root) {
                if (root != null) {
                        inOrderRec(root.left);
                        System.out.println(root.student);
                        inOrderRec(root.right);
                }
        }
}

// Stack for undo functionality
static class UndoStack {
```

```
        Stack<Student> stack = new Stack<>();

        public void push(Student student) {
            stack.push(student);
            System.out.println("Added to undo stack: " + student);
        }

        public void undo() {
            if (stack.isEmpty()) {
                System.out.println("No actions to undo.");
            } else {
                System.out.println("Undo last action: " + stack.pop());
            }
        }
    }

    // Binary Search for students by roll number
    public static int binarySearch(List<Student> students, int rollNumber) {
        int left = 0, right = students.size() - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (students.get(mid).rollNumber == rollNumber) {
                return mid;
            } else if (students.get(mid).rollNumber < rollNumber) {
                left = mid + 1;
            } else {
                right = mid - 1;
```

```
                }
            }
            return -1; // Not found
        }

        // Main function
        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            StudentBST studentBST = new StudentBST();
            UndoStack undoStack = new UndoStack();
            List<Student> studentList = new ArrayList<>();

            while (true) {
                System.out.println("\nExam Result Management System:");
                System.out.println("1. Add Student Result");
                System.out.println("2. View All Results (Sorted by Marks)");
                System.out.println("3. Search Student by Roll Number");
                System.out.println("4. View Failed Students");
                System.out.println("5. Undo Last Entry");
                System.out.println("6. Exit");
                System.out.print("Choose an option: ");
                int choice = scanner.nextInt();
                scanner.nextLine(); // Consume newline

                switch (choice) {
                case 1:
                    System.out.print("Enter roll number: ");
                    int rollNumber = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
```

```
                System.out.print("Enter student name: ");
                String name = scanner.nextLine();
                System.out.print("Enter marks: ");
                int marks = scanner.nextInt();

                Student newStudent = new Student(rollNumber, name, marks);
                studentBST.insert(newStudent);
                undoStack.push(newStudent);
                studentList.add(newStudent);
                System.out.println("Student result added.");
                break;

            case 2:
                studentBST.inOrder();
                break;

            case 3:
                System.out.print("Enter roll number to search: ");
                int searchRoll = scanner.nextInt();
                studentList.sort(Comparator.comparingInt(s -> s.rollNumber));
                int index = binarySearch(studentList, searchRoll);
                if (index != -1) {
                    System.out.println("Student found: " + studentList.get(index));
                } else {
                    System.out.println("Student not found.");
                }
                break;

            case 4:
```

```
                        System.out.println("Failed Students (Marks < 40):");
                        for (Student student : studentList) {
                                if (student.marks < 40) {
                                        System.out.println(student);
                                }
                        }
                        break;

                case 5:
                        undoStack.undo();
                        break;

                case 6:
                        System.out.println("Exiting Exam Result Management System.
Goodbye!");
                        scanner.close();
                        return;

                default:
                        System.out.println("Invalid choice. Please try again.");
                }
        }
    }

}
```

```
Exam Result Management System:
1. Add Student Result
2. View All Results (Sorted by Marks)
3. Search Student by Roll Number
4. View Failed Students
5. Undo Last Entry
6. Exit
Choose an option: 1
Enter roll number: 24045
Enter student name: Abhishek
Enter marks: 92
Added to undo stack: Roll No: 24045, Name: Abhishek, Marks: 92
Student result added.

Exam Result Management System:
1. Add Student Result
2. View All Results (Sorted by Marks)
3. Search Student by Roll Number
4. View Failed Students
5. Undo Last Entry
6. Exit
Choose an option: 1
Enter roll number: 24010
Enter student name: Yash
Enter marks: 87
Added to undo stack: Roll No: 24010, Name: Yash, Marks: 87
Student result added.
```

```
Exam Result Management System:
1. Add Student Result
2. View All Results (Sorted by Marks)
3. Search Student by Roll Number
4. View Failed Students
5. Undo Last Entry
6. Exit
Choose an option: 2
Students sorted by marks:
Roll No: 24010, Name: Yash, Marks: 87
Roll No: 24045, Name: Abhishek, Marks: 92
Roll No: 24018, Name: Sakshi, Marks: 99

Exam Result Management System:
1. Add Student Result
2. View All Results (Sorted by Marks)
3. Search Student by Roll Number
4. View Failed Students
5. Undo Last Entry
6. Exit
Choose an option: 3
Enter roll number to search: 24018
Student found: Roll No: 24018, Name: Sakshi, Marks: 99
```

```
Choose an option: 4
Failed Students (Marks < 40):
```

```
Choose an option: 5
Undo last action: Roll No: 24018, Name: Sakshi, Marks: 99
```

```
Choose an option: 6
Exiting Exam Result Management System. Goodbye!
```