Experiment 1

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the array");
       int numberOfElements = scanner.nextInt();
       int[] elements = new int[numberOfElements];
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for(int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elements[i] = element;
       }
       int count = 0;
       HashMap<Integer,Integer> map = new HashMap<>();
       for(int element : elements) {
           if (map.containsKey(element)) {
               count = map.get(element);
               map.put(element, count + 1);
           } else {
               map.put(element, 1);
           }
       }
       System.out.println();
       boolean areThereDuplicates = false;
       for(Map.Entry<Integer,Integer> entry : map.entrySet()) {
           if(entry.getValue() > 1) {
               System.out.print(entry.getKey() + " ");
               areThereDuplicates = true;
           }
       }
       if(!areThereDuplicates) {
           System.out.println("There are no duplicates present in the given
array.");
       }
   }
}
```

Experiment 2

```
import java.util.LinkedList;
import java.util.Scanner;

public class Main {

   public static int getNumber(LinkedList<Integer> list) {
       StringBuilder result = new StringBuilder();
       int index = 0,size = list.size();
       while(size != 0) {
           result.append(list.get(index).toString());
           size -= 1;
           index += 1;
       }
       return Integer.parseInt(result.toString());
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the linked
list");
       int numberOfElements = scanner.nextInt();
       LinkedList<Integer> elementList = new LinkedList<>();
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for(int i = 0; i < numberOfElements; i++) {
          int element = scanner.nextInt();
          elementList.add(element);
       }
       int result  = getNumber(elementList);
       System.out.println("\nThe number formed from the elements of the linked
list is " + result);
   }
}
```

Experiment 3

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

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

```java
   }

   public int getData() {
       return data;
   }

   public void setData(int data) {
       this.data = data;
   }

   public Node getNext() {
       return next;
   }

   public void setNext(Node next) {
       this.next = next;
   }
}


public class SinglyLinkedList {
   private Node head;
   private int size;

   public void add(int data) {
       if(head == null) {
           addFirst(data);
       }else {
           Node current = head;
           while(current.getNext() != null) {
               current = current.getNext();
           }
           Node newNode = new Node(data);
           current.setNext(newNode);
       }
   }

   private void addFirst(int data) {
       Node newNode = new Node(data);
       if (!isEmpty()) {
           newNode.setNext(head);
       }
       head = newNode;
       size++;
   }

   public Node removeFirst(){
       Node deletedNode = null;
       if(!isEmpty()){
           deletedNode = new Node(head.getData());
           if(size == 1){
               head = null;
           }
           else{
               head = head.getNext();
           }
           size--;
       }
       return deletedNode;
   }

   public Node removeLast(){
       Node deletedNode = null;
       if(!isEmpty()){
           Node current = head;
           Node prev = null;
           while(current.getNext() != null) {
               prev = current;
               current = current.getNext();
           }
           prev.setNext(null);
           deletedNode = current;
       }
       return deletedNode;
   }

   public void printLinkedList() {
       if(!isEmpty()) {
           Node current = head;
           while(current != null) {
               System.out.println(current.getData() + " ");
               current = current.getNext();
           }
           System.out.println();
       }else {
           System.out.println("Linked List is empty");
       }
   }

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

   public int getSize() {
       return size;
   }
}
```

Experiment 4

```java
public class Node {
   private int data;
   private Node next;

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

   public int getData() {
       return data;
   }

   public void setData(int data) {
       this.data = data;
   }

   public Node getNext() {
       return next;
   }

   public void setNext(Node next) {
       this.next = next;
   }
}


public class CircularLinkedList {
   private Node tail;
   private int size;

   public void addFirst(int element) {
       Node node = new Node(element);
       if (isEmpty()) {
           tail = node;
           node.setNext(node);
       } else {
           node.setNext(tail.getNext());
           tail.setNext(node);
       }
       size++;
   }

   public void addLast(int element) {
       Node node = new Node(element);
       if (isEmpty()) {
           addFirst(element);
       } else {
           node.setNext(tail.getNext());
           tail.setNext(node);
           tail = node;
           size++;
       }
   }

   public int removeFirst() {
       int removedElement = 0;
       if (!isEmpty()) {
           Node firstNode = tail.getNext();
           if (firstNode == tail) {
               tail = null;
           } else {
               tail.setNext(firstNode.getNext());
           }
           size--;
           removedElement = firstNode.getData();
       }


       return removedElement;
   }

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

   public int size() {
       return size;
   }

   public void printLinkedList() {
       if (!isEmpty()) {
           Node temp = tail.getNext();
           while (temp != tail) {
               System.out.print(temp.getData() + ", ");
               temp = temp.getNext();
           }
           System.out.println(temp.getData());
       }
   }
}
```

Experiment 5

```
public class MyStack {
   private final int MAX_CAPACITY;
   private int[] arr;
   int top;

   public MyStack(int MAX_CAPACITY) {
       this.MAX_CAPACITY = MAX_CAPACITY;
       arr = new int[MAX_CAPACITY];
       top = 0;
   }

   public void push(int element) {
       if(top != MAX_CAPACITY){
           arr[top] = element;
           top++;
       }
       else{
           System.out.println("Stack OverFlow");
       }
   }

   public int pop() {
       int topElement = 0;
       if(!isEmpty()){
           top--;
           topElement = arr[top];
       }else{
           System.out.println("Stack UnderFlow");
       }
       return topElement;

   }

   public int peek() {
       int topElement = 0;
       if(!isEmpty()){
           topElement = arr[top - 1];
       }
       else{
           System.out.println("Stack is Empty");
       }
       return topElement;
   }

   public boolean isEmpty() {
```

```java
        return top == 0;
    }

    public int size() {
        return top;
    }
    public void printStack(){
        for (int i = 0; i < top ; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Experiment 6

```java
public class Node {
    private int data;
    private Node next;

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

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }

    public Node getNext() {
        return next;
    }

    public void setNext(Node next) {
        this.next = next;
    }
}
public class MyStack {
```

```
private Node top;
private int size;

public MyStack() {
    top = null;
    size = 0;
}

public void push(int element) {
    Node node = new Node(element);
    node.setNext(top);
    top = node;
    size++;
}

public int pop() {
    int removedElement = 0;
    if (!isEmpty()) {
        removedElement = top.getData();
        top = top.getNext();
        size--;
    } else {
        System.out.println("Stack UnderFlow");
    }
    return removedElement;
}

public int peek() {
    int topElement = 0;
    if (!isEmpty()) {
        topElement = top.getData();
    } else {
        System.out.println("Stack is empty");
    }
    return topElement;
}

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

public int size() {
    return size;
}

public void printStack() {
    Node temp = top;
    while(temp != null) {
        System.out.println(temp.getData() + " ");
```

```
           temp = temp.getNext();
       }
       System.out.println();
   }
}
```

Experiment 7

```
public class MyQueue {
   private int[] arr;
   private int front;
   private int rear;
   private int size;

   public MyQueue(int[] arr){
       this.arr = arr;
       front = 0;
       rear = 0;
       size = 0;
   }

   public void enqueue(int element) {
       if(size != arr.length){
           if(rear == arr.length){
               rear = 0;
           }
           arr[rear] = element;
           rear++;
           size++;
       }
       else {
           System.out.println("queue overflow");
       }
   }

   public int dequeue() {
       int removedElement = 0;
       if(!isEmpty()){
           if(size != arr.length){
               if(front == arr.length){
                   front = 0;
               }
           }
           removedElement = arr[front];
           front ++;
           size--;
```

```
        }
        else{
            System.out.println("queue underflow");
        }

        return removedElement;
    }

    public int peek() {
        int frontElement = 0;
        if(!isEmpty()){
            frontElement = arr[front];
        }
        return frontElement;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int size() {
        return size;
    }

    public void printQueue(){
        if(rear <= front){
            for (int i = front; i < arr.length; i++) {
                System.out.print(arr[i] + "<--");
            }
            for (int i = 0; i < rear; i++) {
                System.out.print(arr[i] + "<--");
            }
        }
        else{
            for (int i = front; i < rear; i++) {
                System.out.print(arr[i] + "<--");
            }
        }
    }
}
```

Experiment 8

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

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

   public int getData() {
       return data;
   }

   public void setData(int data) {
       this.data = data;
   }

   public Node getNext() {
       return next;
   }

   public void setNext(Node next) {
       this.next = next;
   }
}


public class MyQueue {
   private Node front;
   private Node rear;
   private int size;
   public MyQueue(){
       front = null;
       rear = null;
       size = 0;
   }

   public void enqueue(int data) {
       Node node = new Node(data);
       if(!isEmpty()){
           rear.setNext(node);
           rear = node;
       }
       else{
           front = node;
           rear = node;
       }
```

```java
        size++;
    }

    public int dequeue() {
        int removedElement = 0;
        if(!isEmpty()){
            removedElement = front.getData();
            front = front.getNext();
            if(front == null){
                rear = null;
            }
            size--;
        }

        return removedElement;
    }

    public int peek() {
        int frontElement = 0;
        if(!isEmpty()){
            frontElement = front.getData();
        }
        return frontElement;
    }

    public boolean isEmpty() {
        return front == null;//size == o or rear == null
    }

    public int size() {
        return size;
    }

    public void printQueue(){
        Node temp = front;
        while (temp != null){
            System.out.print(temp.getData() + "<--");
            temp = temp.getNext();
        }
    }
}
```

Experiment 9

```
public class Node<E extends Comparable<E>> {
    private E data;
    private Node<E> left;
    private Node<E> right;

    public Node(E data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }

    public E getData() {
        return data;
    }

    public void setData(E data) {
        this.data = data;
    }

    public Node<E> getLeft() {
        return left;
    }

    public void setLeft(Node<E> left) {
        this.left = left;
    }

    public Node<E> getRight() {
        return right;
    }

    public void setRight(Node<E> right) {
        this.right = right;
    }

}
```

```
public class MyBinarySearchTree<E extends Comparable<E>> {
    private Node<E> root;

    public Node<E> getRoot() {
        return root;
    }
    public void insert(E data) {
        Node<E> node = new Node<>(data);
        if(isEmpty()) {
            root = node;
        }else {
            Node<E> temp = root;
            Node<E> parent = null;
            while(temp != null) {
                parent = temp;
                if(data.compareTo(temp.getData()) <= 0) {
                    temp = temp.getLeft();

                }else {
                    temp = temp.getRight();
                }
            }
            if(data.compareTo(parent.getData()) <= 0) {
                parent.setLeft(node);
            }else {
                parent.setRight(node);
            }
        }
    }
    public void preOrder(Node<E> node) {
        if(node == null) {
            return;
        }else {
            System.out.print(node.getData() + ",");
            preOrder(node.getLeft());
            preOrder(node.getRight());
        }
        System.out.println();
    }

    public void inOrder(Node<E> node) {
        if(node == null) {
            return;
        }else {
            inOrder(node.getLeft());
            System.out.print(node.getData() + ",");
            inOrder(node.getRight());
        }
        System.out.println();
    }

    public void postOrder(Node<E> node) {
        if(node == null) {
            return;
        }else {
            postOrder(node.getLeft());
            postOrder(node.getRight());
            System.out.print(node.getData() + ",");
        }
        System.out.println();
    }

    public boolean search(E searchElement) {
        boolean response = false;
        Node<E> temp = root;
        while(temp != null) {
            if(searchElement.compareTo(temp.getData()) == 0) {
                response = true;
                break;
            }else if(searchElement.compareTo(temp.getData()) < 0) {
                temp = temp.getLeft();
            }else {
                temp = temp.getRight();
            }
        }
        return response;
    }

    public boolean recursiveSearch(E searchElement) {
        Node<E> temp = root;
        return recursiveSearchHelper(root,searchElement);
    }

    private boolean recursiveSearchHelper(Node<E> root, E searchElement) {
        if(root == null) {
            return false;
        }else if(searchElement.compareTo(root.getData()) == 0) {
            return true;
        }
        if(searchElement.compareTo(root.getData()) < 0) {
            return recursiveSearchHelper(root.getLeft(),searchElement);
        }else {
            return recursiveSearchHelper(root.getRight(),searchElement);
        }
    }

    private boolean isEmpty() {
        if(root == null) {
            return true;
        }
        return false;
    }
}
```

Experiment 10

```java
import java.util.Scanner;

public class MyLinearSearch {

    public boolean linearSearch(int[] arr, int searchElement) {
        boolean response = false;
        for (int j : arr) {
            if (j == searchElement) {
                response = true;
                break;
            }
        }
        return response;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements present in the array");
        int numberOfElements = scanner.nextInt();
        int[] elements = new int[numberOfElements];
        System.out.println("\nNow enter the elements in a single line add space after entering each element.");
        for (int i = 0; i < numberOfElements; i++) {
            int element = scanner.nextInt();
            elements[i] = element;
        }
        System.out.println("Enter the number to be searched");
        int searchElement = scanner.nextInt();
        MyLinearSearch obj = new MyLinearSearch();
        boolean result = obj.linearSearch(elements, searchElement);
        if (result) {
            System.out.println("Number was found in the array");
        } else {
            System.out.println("Number was not found in the array.");
        }
    }
}
```

Experiment 11

```
import java.util.Scanner;

public class MyBinarySearch {

   public boolean binarySearch(int low, int high, int searchElement, int[]
elements) {
       if (low > high) {
           return false;
       }
       int mid = (low + high) / 2;
       if (elements[mid] == searchElement) {
           return true;
       } else if (elements[mid] < searchElement) {
           return binarySearch(mid + 1, high, searchElement, elements);
       } else {
           return binarySearch(low, mid - 1, searchElement, elements);
       }
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the array");
       int numberOfElements = scanner.nextInt();
       int[] elements = new int[numberOfElements];
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for (int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elements[i] = element;
       }
       System.out.println("Enter the number to be searched");
       int searchElement = scanner.nextInt();
       MyBinarySearch obj = new MyBinarySearch();
       boolean result = obj.binarySearch(0, elements.length - 1, searchElement,
elements);
       if (result) {
           System.out.println("Number was found in the array");
       } else {
           System.out.println("Number was not found in the array.");
       }
   }
}
```

Experiment 12

```
import java.util.Arrays;
import java.util.Scanner;

public class MySelectionSort {

   public void selectionSort(int[] arr) {
       int sortedIndex = arr.length;
       int maxElement;
       int maxElementIndex;
       for (int i = 0; i < arr.length; i++) {
           maxElement = arr[0];
           maxElementIndex = 0;
           for (int j = 0; j < sortedIndex; j++) {
               if (maxElement < arr[j]) {
                   maxElement = arr[j];
                   maxElementIndex = j;
               }
           }
           int temp = arr[maxElementIndex];
           sortedIndex--;
           arr[maxElementIndex] = arr[sortedIndex];
           arr[sortedIndex] = temp;
       }
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the array");
       int numberOfElements = scanner.nextInt();
       int[] elements = new int[numberOfElements];
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for (int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elements[i] = element;
       }
       System.out.println("Before Sorting");
       System.out.println(Arrays.toString(elements));
       MySelectionSort obj = new MySelectionSort();
       obj.selectionSort(elements);
       System.out.println("After Sorting");
       System.out.println(Arrays.toString(elements));
   }

}
```

Experiment 13

```java
import java.util.Arrays;
import java.util.Scanner;

public class MyInsertionSort {

   public void insertionSort(int[] arr){
       int unsortedIndex = 1;
       for (int i = unsortedIndex; i < arr.length ; i++) {
           for (int j = i; j > 0; j--) {
               if(arr[j] < arr[j-1]){
                   int temp = arr[j];
                   arr[j] = arr[j-1];
                   arr[j-1] = temp;
               }
               else{
                   break;
               }
           }
       }
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the array");
       int numberOfElements = scanner.nextInt();
       int[] elements = new int[numberOfElements];
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for (int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elements[i] = element;
       }
       System.out.println("Before Sorting");
       System.out.println(Arrays.toString(elements));
       MyInsertionSort obj = new MyInsertionSort();
       obj.insertionSort(elements);
       System.out.println("After Sorting");
       System.out.println(Arrays.toString(elements));

   }
}
```

Experiment 14

```java
import java.util.Arrays;
import java.util.Scanner;

public class MyQuickSort {

   public void quickSort(int[] arr, int lower, int upper){
       if(lower >= upper){
           return;
       }
       int pivotIndex = partition(arr, lower, upper);
       quickSort(arr, lower, pivotIndex -1);
       quickSort(arr, pivotIndex +1 , upper);

   }

   private int partition(int[] arr, int lower, int upper) {
       int pivot = arr[lower];
       int down = lower;
       int up = upper;
       while(down < up){
           while(down <= upper && arr[down] <= pivot){
               down = down + 1;
           }
           while(up >= lower && arr[up] > pivot){
               up = up - 1;
           }
           if(down < up){
               int temp = arr[down];
               arr[down] = arr[up];
               arr[up] = temp;
           }
       }
       arr[lower] = arr[up];
       arr[up] = pivot;
       return up;
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the array");
       int numberOfElements = scanner.nextInt();
       int[] elements = new int[numberOfElements];
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for (int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elements[i] = element;
       }
       System.out.println("Before Sorting");
       System.out.println(Arrays.toString(elements));
```

```
        MyQuickSort obj = new MyQuickSort();
        obj.quickSort(elements,0,elements.length - 1);
        System.out.println("After Sorting");
        System.out.println(Arrays.toString(elements));
    }
}
```

Experiment 15

```
import java.util.Arrays;
import java.util.Scanner;

public class MyMergeSort {
    void merge(int arr[], int l, int m, int r)
    {
        int size1 = m - l + 1;
        int size2 = r - m;

        int L[] = new int[size1];
        int R[] = new int[size2];
        for (int i = 0; i < size1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < size2; ++j)
            R[j] = arr[m + 1 + j];

        int index1 = 0, index2 = 0;

        int mergedIndex = l;
        while (index1 < size1 && index2 < size2) {
            if (L[index1] <= R[index2]) {
                arr[mergedIndex] = L[index1];
                index1++;
            }
            else {
                arr[mergedIndex] = R[index2];
                index2++;
            }
            mergedIndex++;
        }

        while (index1 < size1) {
            arr[mergedIndex] = L[index1];
            index1++;
            mergedIndex++;
        }
        while (index2 < size2) {
            arr[mergedIndex] = R[index2];
            index2++;
            mergedIndex++;
        }
    }

    void sort(int arr[], int low, int high)
    {
        if (low < high) {
            int m = low + (high - low)/2;
            sort(arr, low, m);
            sort(arr, m + 1, high);
            merge(arr, low, m, high);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements present in the array");
        int numberOfElements = scanner.nextInt();
        int[] elements = new int[numberOfElements];
        System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
        for (int i = 0; i < numberOfElements; i++) {
            int element = scanner.nextInt();
            elements[i] = element;
        }
        System.out.println("Before Sorting");
        System.out.println(Arrays.toString(elements));
        MyMergeSort obj = new MyMergeSort();
        obj.sort(elements,0,elements.length - 1);
        System.out.println("After Sorting");
        System.out.println(Arrays.toString(elements));
    }
}
```

Experiment 16

```
import java.util.LinkedList;
import java.util.Scanner;
```

```
public class MySort {

   void sortList(LinkedList list) {
       int[] count = {0,0,0};
       int size = list.size();
       for(int i = 0; i < size;i++) {
           count[(int) list.get(i)]++;
       }
       for(int i = 0; i < size; i++) {
           list.remove(0);
       }
       for(int i = 0; i < 3; i++) {
           while(count[i] != 0) {
               list.add(i);
               count[i]--;
           }
       }
   }

   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       System.out.println("Enter the number of elements present in the linked
list");
       int numberOfElements = scanner.nextInt();
       LinkedList<Integer> elementList = new LinkedList<>();
       System.out.println("\nNow enter the elements in a single line add space
after entering each element.");
       for(int i = 0; i < numberOfElements; i++) {
           int element = scanner.nextInt();
           elementList.add(element);
       }
       System.out.println("Before Sorting");
       System.out.println(elementList);
       MySort obj = new MySort();
       obj.sortList(elementList);
       System.out.println("After Sorting");
       System.out.println(elementList);
   }
}
```