1ST AUGUST ASSIGNMENT

Q1. Write a program to sort an array in descending order using bubble sort. Input Array {3,5,1,6,0} Output Array: {6, 5, 3, 1, 0}

```java
// Java program to implement Bubble Sort in descending order

import java.util.Arrays;

public class BubbleSortDescending {
    public static void bubbleSortDescending(int[] arr) {
        int n = arr.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;

            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] < arr[j + 1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }

            // If no two elements were swapped in the inner loop, the array is already sorted
            if (!swapped)
                break;
        }
    }
```

```
    public static void main(String[] args) {
        int[] arr = {3, 5, 1, 6, 0};
        System.out.println("Input Array: " + Arrays.toString(arr));
        bubbleSortDescending(arr);
        System.out.println("Output Array: " + Arrays.toString(arr));
    }
}
```

Q2. WAP to sort an array in descending order using selection sort. Input Array {3,5,1,6,0} Output Array: {6, 5, 3, 1, 0}

```java
// Java program to implement Selection Sort in descending order

import java.util.Arrays;

public class SelectionSortDescending {
    public static void selectionSortDescending(int[] arr) {
        int n = arr.length;

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

            for (int j = i + 1; j < n; j++) {
                if (arr[j] > arr[maxIndex]) {
                    maxIndex = j;
                }
            }

            // Swap arr[i] and arr[maxIndex]
            int temp = arr[i];
```

```
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {3, 5, 1, 6, 0};
        System.out.println("Input Array: " + Arrays.toString(arr));
        selectionSortDescending(arr);
        System.out.println("Output Array: " + Arrays.toString(arr));
    }
}
```

Q3. WAP to sort an array in decreasing order using insertion sort. Input Array {3,5,1,6,0} Output Array: {6, 5, 3, 1, 0}

```java
// Java program to implement Insertion Sort in descending order

import java.util.Arrays;

public class InsertionSortDescending {
    public static void insertionSortDescending(int[] arr) {
        int n = arr.length;

        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;

            // Move elements that are greater than the key one position ahead
            while (j >= 0 && arr[j] < key) {
                arr[j + 1] = arr[j];
```

```
            j--;
        }

        arr[j + 1] = key;
    }
  }

  public static void main(String[] args) {
    int[] arr = {3, 5, 1, 6, 0};
    System.out.println("Input Array: " + Arrays.toString(arr));
    insertionSortDescending(arr);
    System.out.println("Output Array: " + Arrays.toString(arr));
  }
}
```

Q4. Find out how many passes would be required to sort the following array in decreasing order using bubble sort. Input Array {3, 5, 1, 6, 0}

```java
// Java program to find the number of passes in Bubble Sort in descending order

public class BubbleSortPassesDescending {
  public static int bubbleSortPassesDescending(int[] arr) {
    int n = arr.length;
    int passes = 0;
    boolean swapped;

    for (int i = 0; i < n - 1; i++) {
      swapped = false;

      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] < arr[j + 1]) {
```

```
                // Swap arr[j] and arr[j+1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        // If no two elements were swapped in the inner loop, the array is already sorted
        if (!swapped)
            break;

        passes++;
    }

    return passes;
  }

  public static void main(String[] args) {
    int[] arr = {3, 5, 1, 6, 0};
    System.out.println("Input Array: " + Arrays.toString(arr));
    int passes = bubbleSortPassesDescending(arr);
    System.out.println("Number of passes: " + passes);
  }
}
```

Q5. Find out the number of iterations to sort the array in descending order using selection sort. Input Array {3, 5, 1, 6, 0}

```java
// Java program to find the number of iterations in Selection Sort in descending order
```

```
public class SelectionSortIterationsDescending {
    public static int selectionSortIterationsDescending(int[] arr) {
        int n = arr.length;
        int iterations = 0;

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

            for (int j = i + 1; j < n; j++) {
                if (arr[j] > arr[maxIndex]) {
                    maxIndex = j;
                }
                iterations++;
            }

            // Swap arr[i] and arr[maxIndex]
            int temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }

        return iterations;
    }

    public static void main(String[] args) {
        int[] arr = {3, 5, 1, 6, 0};
        System.out.println("Input Array: " + Arrays.toString(arr));
        int iterations = selectionSortIterationsDescending(arr);
        System.out.println("Number of iterations: " + iterations);
    }
}
```

Q1: Given a number, print its binary representation. [easy] Input 1: number = 5 Output 1: 101 Input 2: number = 10 Output 2: 1010

```java
// Java program to print the binary representation of a number

public class BinaryRepresentation {
    public static void printBinary(int number) {
        StringBuilder binary = new StringBuilder();

        while (number > 0) {
            binary.insert(0, number % 2);
            number

 = number / 2;
        }

        System.out.println("Binary Representation: " + binary.toString());
    }

    public static void main(String[] args) {
        int number1 = 5;
        int number2 = 10;

        System.out.println("Input 1: number = " + number1);
        printBinary(number1);

        System.out.println("Input 2: number = " + number2);
        printBinary(number2);
    }
}
```

Q2: Given a number 'n', predict whether it is a power of two or not. [medium] Input 1: n = 15 Output 1: False Input 2: n = 32 Output 2: True

```java
// Java program to check if a number is a power of two

public class PowerOfTwo {
    public static boolean isPowerOfTwo(int n) {
        return (n > 0) && ((n & (n - 1)) == 0);
    }

    public static void main(String[] args) {
        int number1 = 15;
        int number2 = 32;

        System.out.println("Input 1: n = " + number1);
        System.out.println("Output 1: " + isPowerOfTwo(number1));

        System.out.println("Input 2: n = " + number2);
        System.out.println("Output 2: " + isPowerOfTwo(number2));
    }
}
```

Q3. Problem 1: Given a number. Using bit manipulation, check whether it is odd or even. Input 8, Even 3, False

```java
// Java program to check if a number is odd or even using bit manipulation

public class OddEvenCheck {
    public static boolean isEven(int num) {
```

```
        return (num & 1) == 0;
    }

    public static void main(String[] args) {
        int number1 = 8;
        int number2 = 3;

        System.out.println("Input 1: " + number1 + ", " + (isEven(number1) ? "Even" : "Odd"));
        System.out.println("Input 2: " + number2 + ", " + (isEven(number2) ? "Even" : "Odd"));
    }
}
```

Q4. Given a number, count the number of set bits in that number without using extra space. Note: bit '1' is also known as a set bit.

```java
// Java program to count the number of set bits in a number

public class CountSetBits {
    public static int countSetBits(int num) {
        int count = 0;

        while (num > 0) {
            count += num & 1;
            num >>= 1;
        }

        return count;
    }

    public static void main(String[] args) {
        int number = 15;
```

```
        System.out.println("Number of set bits in " + number + ": " + countSetBits(number));
    }
}
```

Q5. Given an integer array, duplicates are present in it in a way that all duplicates appear an even number of times except one which appears an odd number of times. Find that odd appearing element in linear time and without using any extra memory. For example, Input: arr [] = [4, 3, 6, 2, 6, 4, 2, 3, 4, 3, 3] Output: The odd occurring element is 4.

```java
// Java program to find the odd occurring element in an array without using extra memory

public class OddOccurringElement {
    public static int findOddOccurringElement(int[] arr) {
        int result = 0;

        for (int num : arr) {
            result ^= num;
        }

        return result;
    }

    public static void main(String[] args) {
        int[] arr = {4, 3, 6, 2, 6, 4, 2, 3, 4, 3, 3};
        System.out.print("The odd occurring element is: " + findOddOccurringElement(arr));
    }
}
```

Q1. Given an array. Find the number X in the array. If the element is present, return the index of the element, else print "Element not found in array". Input the size of the array, the array from the user, and the element X from the user. Use Linear Search to find the element.

```java
// Java program to perform Linear Search in an array

import java.util.Scanner;

public class LinearSearch {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements you want to add: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.print("Enter the element to be searched in the array: ");
        int target = scanner.nextInt();
```

```
        int index = linearSearch(arr, target);

        if (index != -1) {
            System.out.println("Element found at index: " + index);
        } else {
            System.out.println("Element not found in the array.");
        }

        scanner.close();
    }
}
```

Q2. Given an array and an integer "target", return the last occurrence of "target" in the array. If the target is not present, return -1.

```java
// Java program to find the last occurrence of an element in an array

public class LastOccurrence {
    public static int lastOccurrence(int[] arr, int target) {
        int n = arr.length;
        for (int i = n - 1; i >= 0; i--) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {1112, 3, 4, 4, 5, 6, 6, 6, 6};
```

```
        int target1 = 4;
        int target2 = 15;

        System.out.println("Input 1: arr = [1112, 3, 4, 4, 5, 6, 6, 6, 6], target = 4");
        System.out.println("Output 1: " + lastOccurrence(arr, target1));

        System.out.println("Input 2: arr = [2, 2, 2, 6, 6, 18, 29, 30, 30, 30], target = 15");
        System.out.println("Output 2: " + lastOccurrence(arr, target2));
    }
}
```

Q3. Given a

sorted binary array, efficiently count the total number of 1's in it.

```java
// Java program to count the total number of 1's in a sorted binary array

public class CountOnesInBinaryArray {
    public static int countOnes(int[] arr) {
        int left = 0;
        int right = arr.length - 1;
        int lastOneIndex = -1;

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

            if (arr[mid] == 1) {
                lastOneIndex = mid;
                left = mid + 1;
            } else {
```

```
            right = mid - 1;
        }
    }

    if (lastOneIndex == -1) {
        return 0;
    }

    return lastOneIndex + 1;
  }

  public static void main(String[] args) {
    int[] arr1 = {0, 0, 0, 0, 1, 1, 1, 1, 1};
    int[] arr2 = {0, 0, 1, 1, 1};

    System.out.println("Input 1: arr = [0, 0, 0, 0, 1, 1, 1, 1, 1]");
    System.out.println("Output 1: " + countOnes(arr1));

    System.out.println("Input 2: arr = [0, 0, 1, 1, 1]");
    System.out.println("Output 2: " + countOnes(arr2));
  }
}
```

Q4. Given a sorted integer array containing duplicates, count occurrences of a given number. If the element is not found in the array, report that as well.

```java
// Java program to count occurrences of a given number in a sorted array

public class CountOccurrences {
  public static int countOccurrences(int[] nums, int target) {
    int leftIndex = findLeftmostIndex(nums, target);
```

```
    int rightIndex = findRightmostIndex(nums, target);

    if (leftIndex == -1) {
        return 0;
    }

    return rightIndex - leftIndex + 1;
}

private static int findLeftmostIndex(int[] nums, int target) {
    int left = 0;
    int right = nums.length - 1;
    int index = -1;

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

        if (nums[mid] >= target) {
            right = mid - 1;
            if (nums[mid] == target) {
                index = mid;
            }
        } else {
            left = mid + 1;
        }
    }

    return index;
}

private static int findRightmostIndex(int[] nums, int target) {
    int left = 0;
```

```
    int right = nums.length - 1;
    int index = -1;

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

        if (nums[mid] <= target) {
            left = mid + 1;
            if (nums[mid] == target) {
                index = mid;
            }
        } else {
            right = mid - 1;
        }
    }

    return index;
}

public static void main(String[] args) {
    int[] nums = {2, 5, 5, 5, 6, 6, 8, 9, 9, 9};
    int target1 = 5;
    int target2 = 6;
    int target3 = 7;

    System.out.println("Input: nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9], target = 5");
    System.out.println("Output: Target 5 occurs " + countOccurrences(nums, target1) + " times");

    System.out.println("Input: nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9], target = 6");
    System.out.println("Output: Target 6 occurs " + countOccurrences(nums, target2) + " times");

    System.out.println("Input: nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9], target = 7");
```

```
        System.out.println("Output: Target 7 occurs " + countOccurrences(nums, target3) + " times");
    }
}
```

Q5: Given a positive integer num, return true if num is a perfect square or false otherwise. A perfect square is an integer that is the square of an integer. In other words, it is the product of some integer with itself.

```java
// Java program to check if a number is a perfect square

public class PerfectSquare {
    public static boolean isPerfectSquare(int num) {
        if (num < 0) {
            return false;
        }

        long left = 0;
        long right = num;

        while (left <= right) {
            long mid = left + (right - left) / 2;
            long square = mid * mid;

            if (square == num) {
                return true;
            } else if (square < num) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
```

```
        return false;
    }

    public static void main(String[] args) {
        int num1 = 16;
        int num2 = 14;

        System.out.println("Input 1: num = 16");
        System.out.println("Output 1: " + isPerfectSquare(num1));

        System.out.println("Input 2: num = 14");
        System.out.println("Output 2: " + isPerfectSquare(num2));
    }
}
```