Codes

Simple Sieve & Incremental Sieve

```
import java.util.*;
class Main {
    public static void isPrime(int n){
        boolean p[]=new boolean[n+1];
        Arrays.fill(p,true);
        for(int i=2;i*i<=n;i++){
            if(p[i]==true){
                for(int j=i*i;j<=n;j+=i){
                    p[j]=false;
                }
            }
        }
        for(int i=2;i<=n;i++){
            if(p[i]==true){
                System.out.print(i+" ");
            }
        }
    }
    public static void main(String[] args) {
        //Scanner sc=new Scanner(System.in);
        //int n=sc.nextInt();
        isPrime(30);
    }
}
```

Segmented Sieve

```
import java.util.*;
class Main {
    public static void isPrime(int l, int r){
        boolean p[]=new boolean[r+1];
        Arrays.fill(p,true);
        for(int i=2;i*i<=r;i++){
            if(p[i]==true){
                for(int j=i*i;j<=r;j+=i){
                    p[j]=false;
                }
            }
        }
        for(int i=Math.max(l,2);i<=r;i++){
            if(p[i]==true){
                System.out.print(i+" ");
            }
        }
    }
    public static void main(String[] args) {
        isPrime(10,30);
    }
}
```

Simple & Segmented Sieve

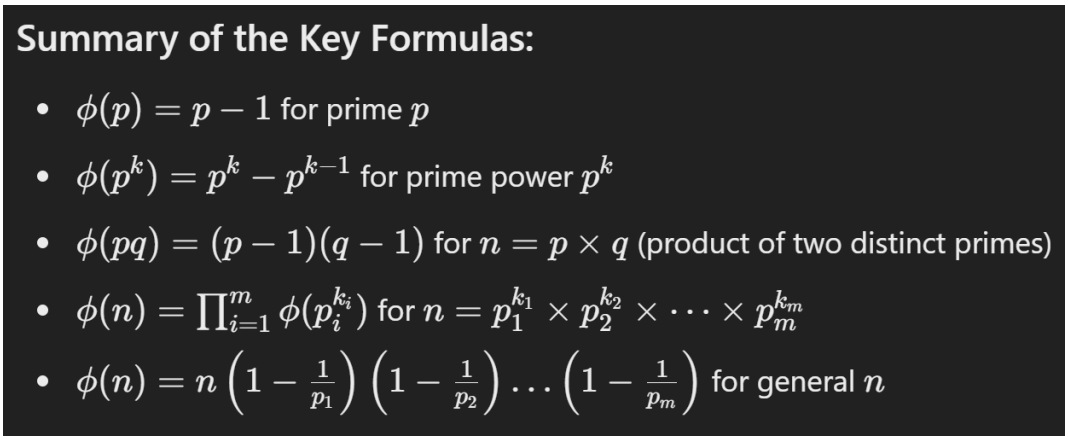
```
import java.util.Arrays;

public class UnifiedSieve {
    // Simple Sieve logic
    public static void simpleSieve(int n) {
        boolean[] isPrime = new boolean[n + 1];
        Arrays.fill(isPrime, true);
        for (int p = 2; p * p <= n; p++) {
            if (isPrime[p]) {
                for (int i = p * p; i <= n; i += p) {
                    isPrime[i] = false;
                }
            }
        }
        for (int i = 2; i <= n; i++) {
```

```
            if (isPrime[i]) {
                System.out.print(i + " ");
            }
        }
    }

    // Handle Segmented Sieve using the Simple Sieve approach
    public static void segmentedSieve(int L, int R) {
        boolean[] isPrime = new boolean[R + 1];
        Arrays.fill(isPrime, true);
        for (int p = 2; p * p <= R; p++) {
            if (isPrime[p]) {
                for (int i = p * p; i <= R; i += p) {
                    isPrime[i] = false;
                }
            }
        }
        for (int i = Math.max(L, 2); i <= R; i++) {
            if (isPrime[i]) {
                System.out.print(i + " ");
            }
        }
    }

    public static void main(String[] args) {
        // Example 1: Simple Sieve
        System.out.println("Primes up to 50:");
        simpleSieve(50);

        System.out.println("\n\nPrimes in range [10, 50]:");
        // Example 2: Segmented Sieve using Simple Sieve logic
        segmentedSieve(10, 50);

        System.out.println("\n\nPrimes up to 100 (incremental logic):");
        // Example 3: Incremental logic (simply reusing Simple Sieve)
        simpleSieve(100);
    }
}
```

Euler's Phi Algorithm

Euler's totient function (also called phi-function or totient function) takes a single positive integer n as input and outputs the number of integers present between 1 and n that are co-prime to n.
Note:
• 2 positive integers a and b are said to be co-prime if their greatest common factor/divisor is equal to 1, that is, gcd(a,b)=1

• 1 is considered co-prime to all numbers.



Properties

	Criteria of 'n'	Formula
	'n' is prime.	Φ(n) = (n-1)
Φ(n)	n = p x q 'p' and 'q' are primes.	Φ(n) = (p-1) × (q-1)
	n = a x b. Either 'a' or 'b' is composite. Both 'a' and 'b' are composite.	$\Phi(n)=n \times \left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\ldots$

```
import java.util.*;

class Main {
    // Function to calculate the greatest common divisor (GCD) using Euclidean algori
    public static int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }

    // Function to calculate Euler's Totient function
    public static int euler(int n) {
        int count = 1; // Initialize count to 1 because 1 is always coprime with n
        for (int i = 2; i < n; i++) {  // Loop from 2 to n-1
            if (gcd(i, n) == 1) {
                count++;
            }
        }
```

```
        return count;
    }

    public static void main(String[] args) {
        int ans = euler(121);  // Calculate Euler's Totient function for n = 121
        System.out.println(ans);  // Output the result
    }
}
```

Remainder Theorem

Yes, "**relatively prime**" and "**coprime**" are two terms that mean exactly the same thing in mathematics. They both refer to two numbers that share no common divisors other than 1.

Definition:

- **Relatively Prime** (or **Coprime**) numbers: Two integers a and b are relatively prime (or coprime) if their **greatest common divisor (GCD)** is 1. In other words, $\gcd(a, b) = 1$.

Examples:

- 8 and 15 are relatively prime because $\gcd(8, 15) = 1$. Their only common divisor is 1.
- 14 and 25 are coprime because $\gcd(14, 25) = 1$.

```
import java.io.*;
import java.util.*;

class Main {

    // Function to solve the system of congruences using brute force approach
    static int findMinX(int num[], int rem[], int k) {
        int x = 1;  // Start with the initial guess for x

        // Infinite loop until a valid solution is found
        while (true) {
            int j;

            // Loop through all congruences to check if current x satisfies all
            for (j = 0; j < k; j++) {
                // If x does not satisfy the j-th congruence, break and try the next
                if (x % num[j] != rem[j]) {
                    break;
                }
            }
```

```
            // If x satisfies all congruences (i.e., j == k), return x as the solutio
            if (j == k) {
                return x;
            }

            // If x does not satisfy all congruences, increment x and check again
            x++;
        }
    }

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

        // Input the number of congruence relations (size of the system)
        System.out.println("Enter the number of congruence relations: ");
        int size = sc.nextInt();

        // Arrays to store remainders (a) and moduli (m)
        int a[] = new int[size];
        int m[] = new int[size];

        // Input the values of a (remainders)
        System.out.println("Enter the values of a: ");
        for (int i = 0; i < size; i++) {
            a[i] = sc.nextInt(); // Read remainder for each congruence
        }

        // Input the values of m (moduli)
        System.out.println("Enter the values of m: ");
        for (int i = 0; i < size; i++) {
            m[i] = sc.nextInt(); // Read modulus for each congruence
        }

        // Call the function to find the solution x that satisfies all the congruence
        System.out.println("x is " + findMinX(m, a, size));
    }
}
```

Strobogrammatic Number

```
import java.util.*;

public class StrobogrammaticNumber {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number: ");
        String num = sc.nextLine();

        if (isStrobogrammatic(num)) {
            System.out.println(num + " is a strobogrammatic number");
        } else {
            System.out.println(num + " is not a strobogrammatic number");
        }

        sc.close();
    }

    public static boolean isStrobogrammatic(String num) {
        // Create a map to store valid strobogrammatic pairs
        Map<Character, Character> strobogrammaticDictionary = new HashMap<>();
        strobogrammaticDictionary.put('0', '0');
        strobogrammaticDictionary.put('1', '1');
        strobogrammaticDictionary.put('6', '9');
        strobogrammaticDictionary.put('8', '8');
        strobogrammaticDictionary.put('9', '6');

        int n = num.length();
        // Iterate from both ends of the string towards the center
        for (int i = 0, j = n - 1; i <= j; i++, j--) {
            char digitLeft = num.charAt(i);
            char digitRight = num.charAt(j);

            // Get the strobogrammatic pair of digitLeft, if it exists
            char mapping = strobogrammaticDictionary.getOrDefault(digitLeft, '-');

            // If the digit is not strobogrammatic or the pair doesn't match, return
            if (mapping == '-' || mapping != digitRight) {
                return false;
            }
        }
        return true;
    }
}
```

ALICE APPLE TREE

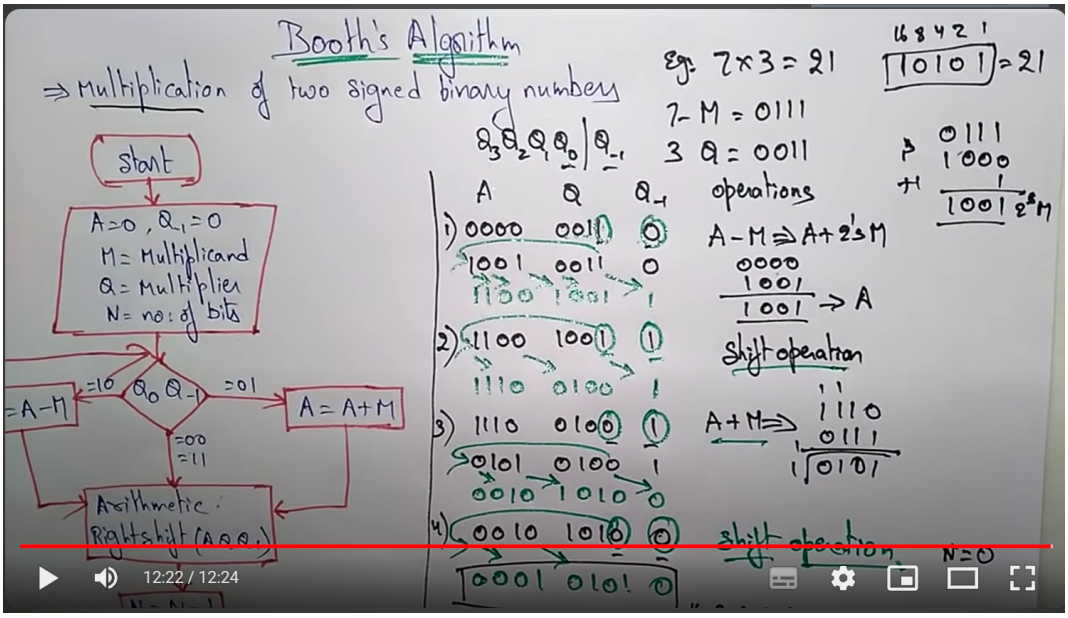
```
import java.util.*;
```

```
class Main {
    static int minApples(int M, int K, int S, int W, int E) {
        // If M apples can be collected entirely from South trees
        if (M <= S * K) {
            return M;
        }
        // If M apples can be collected from South, East, and West trees
        else if (M <= S * K + E + W) {
            return S * K + (M - S * K); // Add remaining apples from East and West
        }
        // If it's impossible to collect enough apples
        else {
            return -1;
        }
    }

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

        // Input collection with proper prompts on a single line
        System.out.print("Enter no of apples to be collected: ");
        int M = sc.nextInt();

        System.out.print("Enter no of apples in each tree: ");
        int K = sc.nextInt();

        System.out.print("Enter no of trees in North: ");
        int N = sc.nextInt(); // This is unused, can be removed from method signature

        System.out.print("Enter no of trees in South: ");
        int S = sc.nextInt();

        System.out.print("Enter no of trees in West: ");
        int W = sc.nextInt();

        System.out.print("Enter no of trees in East: ");
        int E = sc.nextInt();

        // Calculate the result using the minApples function
        int ans = minApples(M, K, S, W, E);

        // Output the result
        System.out.println("The minimum apples that can be collected: " + ans);

        sc.close();  // Close the scanner after use
    }
}
```

Binary Palindrome

```
import java.util.*;

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

        // Input the integer
        System.out.print("Enter an integer: ");
        int X = sc.nextInt();

        // Convert the integer to binary string
        String binary = Integer.toBinaryString(X);

        // Check if the binary string is a palindrome
        if (isPalindrome(binary)) {
            System.out.println("The binary representation is a palindrome.");
        } else {
            System.out.println("The binary representation is not a palindrome.");
        }

        sc.close();  // Close the scanner after use
    }

    // Helper function to check if a string is a palindrome
    public static boolean isPalindrome(String str) {
        // Use a for loop to compare characters from both ends of the string
        for (int left = 0, right = str.length() - 1; left < right; left++, right--) {
            if (str.charAt(left) != str.charAt(right)) {
                return false; // Not a palindrome
            }
        }
        return true; // It's a palindrome
    }
}
```

Booth's Algorithm




```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Create scanner object for input
        Scanner sc = new Scanner(System.in);
        // Read two integers
        int a = sc.nextInt();  // Multiplier
        int b = sc.nextInt();  // Multiplicand
        // Initialize product to 0
        int product = 0;

        // Get the number of bits in 'a' (length of binary string of 'a')
        int n = Integer.toBinaryString(a).length();

        // Perform binary multiplication using bit shifting
        for (int i = 0; i < n; i++) {
            // Extract the current least significant bit of 'a'
            int currentBit = (a & 1);

            // If the bit is 1, add 'b' to the product
            if (currentBit == 1) {
                product += b;
            }

            // Shift 'b' left by 1 (equivalent to multiplying by 2)
            b <<= 1;

            // Shift 'a' right by 1 (move to next bit)
```

```
            a >>= 1;
        }

        // Print the final product
        System.out.println("Result: " + product);
    }
}

/*
import java.util.Scanner;

public class Main {
    public static int boothAlgorithmMock(int a, int b) {
        // Simply return the product of a and b, mimicking the result of Booth's Algo
        return a * b;
    }

    public static void main(String[] args) {
        // Create scanner object for input
        Scanner sc = new Scanner(System.in);

        // Read two integers
        int a = sc.nextInt();  // Multiplier
        int b = sc.nextInt();  // Multiplicand

        // Call the mock Booth's algorithm method to get the product
        int result = boothAlgorithmMock(a, b);

        // Output the result
        System.out.println("Result: " + result);
    }
}

*/
```

Euclid's Algorithm

```
import java.util.*;

class Main {
    // Function to calculate the GCD using Euclidean algorithm
    public static int gcd(int a, int b) {
        // Base case: If a is 0, the GCD is b
        if (a == 0) {
            return b;
```

```
        }
        // Recursive case: Call the gcd function with b % a and a
        return gcd(b % a, a);
    }

    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner sc = new Scanner(System.in);
        // Read the first integer & second integer
        int a = sc.nextInt();
        int b = sc.nextInt();

        // Call the gcd function to find the greatest common divisor
        int res = gcd(a, b);
        System.out.println(res);
        sc.close();
    }
}
```

Karatsuba Algorithm

```
import java.util.Scanner;

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

        // Taking input from the user
        System.out.print("Enter the first number: ");
        int a = sc.nextInt();  // Read first number

        System.out.print("Enter the second number: ");
        int b = sc.nextInt();  // Read second number

        // Direct multiplication
        int result = a * b;

        // Output the multiplication result
        System.out.println("Multiplication result: " + result);

        sc.close();  // Close the scanner after use
    }
}

/* Implenting the same using Long Datatype
```

```
import java.util.Scanner;

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

        // Taking input from the user (using long to support larger numbers)
        System.out.print("Enter the first number: ");
        long a = sc.nextLong();  // Read first number

        System.out.print("Enter the second number: ");
        long b = sc.nextLong();  // Read second number

        // Direct multiplication using long
        long result = a * b;

        // Output the multiplication result
        System.out.println("Multiplication result: " + result);

        sc.close();  // Close the scanner after use
    }
}

*/
```

Longest Sequence of 1 After Flipping a Bit

```
import java.util.*;

class Main {
    public static int maxOne(int[] a, int k) {
        int wstart = 0;  // Start of the sliding window
        int count = 0;   // Count of zeros within the window
        int max1 = 0;    // Maximum length of the subarray found

        // Iterate over the array with 'wend' (end of the window)
        for (int wend = 0; wend < a.length; wend++) {
            // If we encounter a 0, increment the replacement count
            if (a[wend] == 0) {
                count++;
            }

            // If the number of replacements exceeds 'k', shrink the window from the
            while (count > k) {
                if (a[wstart] == 0) {
```

```
                    count--; // Decrease the count of zeros
                }
                wstart++;  // Shrink the window from the left
            }

            // Update the maximum length of the subarray after processing the window
            max1 = Math.max(max1, wend - wstart + 1);
        }

        return max1;  // Return the length of the longest subarray
    }

    public static void main(String[] args) {
        int[] a = new int[]{1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0};
        int k = 1;  // Number of zeros allowed to replace
        int res = maxOne(a, k);  // Call the function to get the result
        System.out.println(res);  // Print the result
    }
}
```

Swap Two Nibbles in a Byte

```
import java.util.Scanner;

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

        // Take input from the user
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        // Swap the nibbles
        int swapnum = ((num & 0x0F) << 4 | (num & 0xF0) >> 4);

        // Print the result before and after swapping the nibbles
        System.out.println("Before swapping the nibble: " + num);
        System.out.println("After swapping the nibble: " + swapnum);

        // Close the scanner
        scanner.close();
    }
}
```

Block Swap Algorithm

```
//Right Rotate
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int k = 2;  // Number of positions to rotate
        int n = arr.length;
        // If k is greater than the length of the array, use modulo to avoid redundan
        k = k % n;
        // Create a new array to hold the rotated result
        int[] result = new int[n];

        // Use System.arraycopy to perform the block swap
        System.arraycopy(arr, n - k, result, 0, k);  // Copy last k elements to the f
        System.arraycopy(arr, 0, result, k, n - k);  // Copy first n-k elements to th

        // Print the result
        System.out.println(Arrays.toString(result));
    }
}
/* Can use for loop to print elements
        Use based on the requied output
                    for(int i=0;i<n;i++){
                        System.out.print(res[i]+" ");
                    }
*/

//Left Rotate
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int k = 2;
        int n = arr.length;
        k = k % n;
        int[] result = new int[n];
        System.arraycopy(arr, k, result, 0, n - k);
        System.arraycopy(arr, 0, result, n - k, k);
        System.out.println(Arrays.toString(result));
    }
}
```

Max Product Subarray

```
import java.util.*;
public class Main
{
    static int maxProductSubArray(int arr[]) {
        int result = Integer.MIN_VALUE;
        for(int i=0;i<arr.length-1;i++)
            for(int j=i+1;j<arr.length;j++) {
                int prod = 1;
                for(int k=i;k<=j;k++)
                    prod *= arr[k];
                result = Math.max(result,prod);
            }
       return result;
    }
    public static void main(String[] args) {
        int nums[] = {1,2,-3,0,-4,-5};
        int answer = maxProductSubArray(nums);
        System.out.print("The maximum product subarray is: "+answer);
    }
}

/* Better Code using 2 Loops
import java.util.*;
public class Main
{
    static int maxProductSubArray(int arr[]) {
        int result = arr[0];
        for(int i=0;i<arr.length-1;i++) {
            int p = arr[i];
            for(int j=i+1;j<arr.length;j++) {
                result = Math.max(result,p);
                p *= arr[j];
            }
            result = Math.max(result,p);
        }
       return result;
    }
    public static void main(String[] args) {
        int nums[] = {1,2,-3,0,-4,-5};
        int answer = maxProductSubArray(nums);
        System.out.print("The maximum product subarray is: "+answer);
    }
}
*/
```

Maximum Sum of Hourglass in Matrix

*The total number of hourglasses in a matrix is equal to (R-2) * (C-2), where R refers to the number of Rows and C refers to the number of Columns.*

```
import java.util.Scanner;

class Main {

    // Function to calculate the maximum hourglass sum
    static int findMaxSum(int[][] mat, int row, int col) {

        // Check if matrix size is less than 3x3, in which hourglass cannot be formed
        if (row < 3 || col < 3) {
            System.out.println("Not possible to calculate hourglass sum for given mat
            return Integer.MIN_VALUE; // Return minimum value if matrix is too small
        }

        // Initialize max_sum to the smallest possible integer
        int max_sum = Integer.MIN_VALUE;

        // Traverse the matrix to calculate hourglass sums
        for (int i = 0; i < row - 2; i++) {
            for (int j = 0; j < col - 2; j++) {
                // Calculate the sum of the current hourglass
                int sum = (mat[i][j] + mat[i][j + 1] + mat[i][j + 2]) +
                          (mat[i + 1][j + 1]) +
                          (mat[i + 2][j] + mat[i + 2][j + 1] + mat[i + 2][j + 2]);

                // Update max_sum with the maximum value found
                max_sum = Math.max(max_sum, sum);
            }
        }

        return max_sum; // Return the maximum hourglass sum
    }

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

        // Read the number of rows and columns
        int row = input.nextInt();
        int col = input.nextInt();

        // Initialize the matrix
        int[][] mat = new int[row][col];
```

```
        // Read the matrix elements
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                mat[i][j] = input.nextInt();
            }
        }

        // Find and print the maximum hourglass sum
        int res = findMaxSum(mat, row, col);
        if (res != Integer.MIN_VALUE) {
            System.out.println(res);
        }

        input.close(); // Close the scanner
    }
}
```

Max Equilibrium Sum

```
import java.util.*;

class Main {
    public static int findMaxSum(int[] arr, int n) {
        int res = Integer.MIN_VALUE;
        boolean found = false;

        // Iterate through each element of the array
        for (int i = 0; i < n; i++) {
            // Calculate prefix sum for elements from 0 to i (inclusive)
            int presum = 0;
            for (int j = 0; j <= i; j++) {
                presum += arr[j];
            }

            // Calculate suffix sum for elements from i to n-1 (inclusive)
            int suffsum = 0;
            for (int j = i; j < n; j++) {
                suffsum += arr[j];
            }

            // Check if prefix and suffix sums are equal
            if (presum == suffsum) {
                found = true; // Indicate at least one equilibrium point found
                res = Math.max(res, presum);  // Update the maximum sum
```

```
            }
        }

        // Return result if found, otherwise return -1 for no equilibrium point
        return found ? res : -1;
    }

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

        // Input the size of the array
        System.out.println("Enter size of the array");
        int n = s.nextInt();

        // Input the elements of the array
        int[] arr = new int[n];
        System.out.println("Enter elements of the array");
        for (int i = 0; i < n; i++) {
            arr[i] = s.nextInt();
        }

        // Output the result of findMaxSum function
        int result = findMaxSum(arr, n);
        if (result == -1) {
            System.out.println("No equilibrium point found");
        } else {
            System.out.println("Maximum equilibrium sum: " + result);
        }
    }
}


/*import java.util.*;
class Main {
    static int findMaxSum(int[] arr, int n) {
        int res = Integer.MIN_VALUE;

        // Iterate over each element in the array
        for (int i = 0; i < n; i++) {
            // Calculate the prefix sum
            int prefix_sum = arr[i];
            for (int j = 0; j < i; j++) {
                prefix_sum += arr[j];
            }

            // Calculate the suffix sum
            int suffix_sum = arr[i];
            for (int j = n - 1; j > i; j--) {
```

```
                suffix_sum += arr[j];
            }

            // If prefix sum equals suffix sum, update the result
            if (prefix_sum == suffix_sum) {
                res = Math.max(res, prefix_sum);
            }
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter size of the array");
        int n = s.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter elements of the array");
        for (int i = 0; i < n; i++) {
            arr[i] = s.nextInt();
        }
        System.out.println(findMaxSum(arr, n));
    }
}
*/
```

Leaders in Array

```
import java.util.Arrays;

class Main {
    public static int[] printLeadersBruteForce(int[] arr, int n) {
        int[] leaders = new int[n];  // Create a temporary array to hold leaders
        int index = 0;  // Initialize index to 0

        for (int i = 0; i < n; i++) {
            boolean leader = true;
            // Checking if arr[i] is greater than all elements on its right
            for (int j = i + 1; j < n; j++) {
                if (arr[j] > arr[i]) {
                    leader = false;
                    break;
                }
            }

            // If it is a leader, append it to the leaders array
```

```
            if (leader) {
                leaders[index] = arr[i];
                index++;
            }
        }

        // Now, you might want to trim the array to the correct size
        return Arrays.copyOf(leaders, index);  // Trim the array to the correct size
    }

    public static void main(String[] args) {
        int[] arr = {10, 22, 12, 3, 0, 6};
        int n = arr.length;
        int[] result = printLeadersBruteForce(arr, n);

        // Print the result (leaders)
        System.out.println(Arrays.toString(result));  // Output: [22, 12, 6]
    }
}
```

Majority Element

```
import java.util.*;

public class Main {

    public static int majorityElement(int[] v) {
        // Size of the given array:
        int n = v.length;
        for (int i = 0; i < n; i++) {
            // Selected element is v[i]
            int cnt = 0;
            for (int j = 0; j < n; j++) {
                // Counting the frequency of v[i]
                if (v[j] == v[i]) {
                    cnt++;
                }
            }

            // Check if frequency is greater than n/2
            if (cnt > (n / 2))
                return v[i];
        }
```

```
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {2, 2, 1, 1, 1, 2, 2};
        int ans = majorityElement(arr);
        System.out.println("The majority element is: " + ans);
    }
}
```

Quick, Selection Sort

```
import java.util.*;

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

        // Read the input array size and array elements
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

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

        // Use Arrays.sort to sort the array
        Arrays.sort(arr);

        // Print the sorted array
        System.out.println("Sorted array: " + Arrays.toString(arr));

        sc.close();  // Close the scanner after use
    }
}
```

Josephus Trap

```
import java.util.*;
```

```
class Main {
    static int josephus(int n, int k) {
        if (n == 1)
            return 1;
        else
            return (josephus(n - 1, k) + k - 1) % n + 1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int k = sc.nextInt();
        System.out.println(josephus(n, k));
    }
}
```

Weighted Substring

Move Hyphen to Beginning

Manacher's Algorithm

Sorted Unique Permutation

Maneuvering

Combination

Theoretical Topics in Syllabus:

Warnsdorff’s Algorithm
Hamiltonian Cycle
Kruskal's Algorithm
Activity Selection Problem
Graph Coloring
Huffman Coding

Maze Solving

N Queens

Maze and N Queens won't come for coding