Problems to solve

1. Recursive program for prime number

```
import java.util.*;

class GFG {

    // Returns true if n is prime, else
    // return false.
    // i is current divisor to check.
    static boolean isPrime(int n, int i)
    {

        // Base cases
        if (n <= 2)
            return (n == 2) ? true : false;
        if (n % i == 0)
            return false;
        if (i * i > n)
            return true;

        // Check for next divisor
        return isPrime(n, i + 1);
    }

    // Driver program to test above function
    public static void main(String[] args)
    {

        int n = 15;

        if (isPrime(n, 2))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

2. Prime factors of a big number

Given a number N, print all the prime factors and their powers.

```
void factorize(long long n)
{
    int count = 0;

    // count the number of times 2 divides
    while (!(n % 2)) {
        n >>= 1; // equivalent to n=n/2;
        count++;
    }

    // if 2 divides it
    if (count)
        cout << 2 << "  " << count << endl;

    // check for all the possible numbers that can
```

```
    // divide it
    for (long long i = 3; i <= sqrt(n); i += 2) {
        count = 0;
        while (n % i == 0) {
            count++;
            n = n / i;
        }
        if (count)
            cout << i << "  " << count << endl;
    }

    // if n at the end is a prime number.
    if (n > 2)
        cout << n << "  " << 1 << endl;
}

// driver program to test the above function
int main()
{
    long long n = 1000000000000000000;
    factorize(n);
    return 0;
}
```

3. Check if a number is divisible by all prime divisors of another number

```
class Divisible
{
    public static int gcd(int a, int b) {
      return b == 0 ? a : gcd(b, a % b); }

    // Returns true if all prime factors
    // of y divide x.
    static boolean isDivisible(int x, int y)
    {
        if (y == 1)
            return true;

        int z = gcd(x, y);

        if (z == 1)
            return false;

        return isDivisible(x, y / z);
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        int x = 18, y = 12;
        if (isDivisible(x, y))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

4. Prime numbers after prime P with sum S

Given three numbers sum S, prime P and N, find all N prime numbers after prime P such that their sum is equal to S.

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

class GFG
{
    // vector to store prime
    // and N primes whose sum
    // equals given S
    static ArrayList<Integer> set =
                    new ArrayList<Integer>();
    static ArrayList<Integer> prime =
                    new ArrayList<Integer>();

    // function to check
    // prime number
    static boolean isPrime(int x)
    {
        // square root of x
        int sqroot = (int)Math.sqrt(x);

        // since 1 is not
        // prime number
        if (x == 1)
            return false;

        // if any factor is
        // found return false
        for (int i = 2;
                 i <= sqroot; i++)
            if (x % i == 0)
                return false;

        // no factor found
        return true;
    }

    // function to display N
    // primes whose sum equals S
    static void display()
    {
        int length = set.size();
        for (int i = 0;
                 i < length; i++)
            System.out.print(
                    set.get(i) + " ");
        System.out.println();
    }

    // function to evaluate
    // all possible N primes
    // whose sum equals S
    static void primeSum(int total, int N,
                         int S, int index)
    {
        // if total equals S
        // And total is reached
```

```
    // using N primes
    if (total == S &&
        set.size() == N)
    {
        // display the N primes
        display();
        return;
    }

    // if total is greater
    // than S or if index
    // has reached last
    // element
    if (total > S ||
        index == prime.size())
        return;

    // add prime.get(index)
    // to set vector
    set.add(prime.get(index));

    // include the (index)th
    // prime to total
    primeSum(total + prime.get(index),
                     N, S, index + 1);

    // remove element
    // from set vector
    set.remove(set.size() - 1);

    // exclude (index)th prime
    primeSum(total, N,
             S, index + 1);
}

// function to generate
// all primes
static void allPrime(int N,
                     int S, int P)
{
    // all primes less
    // than S itself
    for (int i = P + 1;
             i <= S ; i++)
    {
        // if i is prime add
        // it to prime vector
        if (isPrime(i))
            prime.add(i);
    }

    // if primes are
    // less than N
    if (prime.size() < N)
        return;
    primeSum(0, N, S, 0);
}

// Driver Code
```

```
    public static void main(String args[])
    {
        int S = 54, N = 2, P = 3;
        allPrime(N, S, P);
    }
}
```

5. Find the highest occurring digit in prime numbers in a range

```
class GFG {

    // Sieve of Eratosthenes
    static void sieve(boolean prime[], int n) {

        for (int p = 2; p * p <= n; p++) {
            if (prime[p] == false)
                for (int i = p * 2; i <= n; i += p)
                    prime[i] = true;
        }
    }

    // Returns maximum occurring digits in primes
    // from l to r.
    static int maxDigitInPrimes(int L, int R) {

        boolean prime[] = new boolean[R + 1];
        Arrays.fill(prime, false);

        // Finding the prime number up to R.
        sieve(prime, R);

        // Initialse frequency of all digit to 0.
        int freq[] = new int[10];
        int val;

        // For all number between L to R, check if
        // prime or not. If prime, incrementing
        // the frequency of digits present in the
        // prime number.
        for (int i = L; i <= R; i++) {

            if (!prime[i]) {
                int p = i; // If i is prime

                while (p > 0) {
                freq[p % 10]++;
                p /= 10;
                }
            }
        }

        // Finding digit with highest frequency.
        int max = freq[0], ans = 0;

        for (int j = 1; j < 10; j++) {
            if (max <= freq[j]) {
                max = freq[j];
                ans = j;
```

```
            }
        }

        return ans;
    }

    // Driver code
    public static void main(String[] args) {
        int L = 1, R = 20;
        System.out.println(maxDigitInPrimes(L, R));
    }
}
```

6. Check if a number is Full Prime

```
class Prime{

    // function to check digits
    public static boolean checkDigits(int n)
    {
        // check all digits are prime or not
        while (n > 0) {
            int dig = n % 10;

            // check if digits are prime or not
            if (dig != 2 && dig != 3 &&
                dig != 5 && dig != 7)
                return false;

            n /= 10;
        }

        return true;
    }

    // To check if n is prime or not
    public static boolean prime(int n)
    {
        if (n == 1)
            return false;

        // check for all factors
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0)
                return false;
        }

        return true;
    }

    // To check if n is Full Prime
    public static boolean isFullPrime(int n)
    {
        // The order is important here for
        // efficiency
        return (checkDigits(n) && prime(n));
    }
```

```
    // driver code
    public static void main(String[] args)
    {
        int n = 53;
        if (isFullPrime(n))
            System.out.print( "Yes" );
        else
            System.out.print( "No");
    }
}
```

7. Insert minimum number in array so that sum of array becomes prime

```
class GFG
{
    // function to check if a
    // number is prime or not
    static boolean isPrime(int n)
        {
            // Corner case
            if (n <= 1)
                return false;

            // Check from 2 to n - 1
            for (int i = 2; i < n; i++)
                if (n % i == 0)
                    return false;

            return true;
        }

    // Find prime number
    // greater than a number
    static int findPrime(int n)
        {
            int num = n + 1;

            // find prime greater than n
            while (num > 0)
                {

                    // check if num is prime
                    if (isPrime(num))
                        return num;

                    // increment num
                    num = num + 1;
                }
            return 0;
        }

    // To find number to be added
    // so sum of array is prime
    static int minNumber(int arr[], int n)
        {
            int sum = 0;

            // To find sum of array elements
```

```
            for (int i = 0; i < n; i++)
                sum += arr[i];

            // if sum is already prime
            // return 0
            if (isPrime(sum))
                return 0;

            // To find prime number
            // greater than sum
            int num = findPrime(sum);

            // Return difference of
            // sum and num
            return num - sum;
        }

    // Driver Code
    public static void main(String[]args)
        {
            int arr[] = { 2, 4, 6, 8, 12 };
            int n = arr.length;
            System.out.println(minNumber(arr, n));
        }
}
```

8. Find the prime numbers which can written as sum of most consecutive primes

9. Find two prime numbers with given sum

```
class GFG
{
    // Generate all prime numbers less than n.
    static boolean SieveOfEratosthenes(int n, boolean isPrime[])
    {
        // Initialize all entries of boolean
        // array as true. A value in isPrime[i]
        // will finally be false if i is Not a
        // prime, else true bool isPrime[n+1];
        isPrime[0] = isPrime[1] = false;
        for (int i = 2; i <= n; i++)
            isPrime[i] = true;

        for (int p = 2; p * p <= n; p++)
        {
            // If isPrime[p] is not changed,
            // then it is a prime
            if (isPrime[p] == true)
            {
                // Update all multiples of p
                for (int i = p * 2; i <= n; i += p)
                    isPrime[i] = false;
            }
        }
        return false;
    }

    // Prints a prime pair with given sum
    static void findPrimePair(int n)
```

```
    {
        // Generating primes using Sieve
        boolean isPrime[]=new boolean[n + 1];
        SieveOfEratosthenes(n, isPrime);

        // Traversing all numbers to find first
        // pair
        for (int i = 0; i < n; i++)
        {
            if (isPrime[i] && isPrime[n - i])
            {
                System.out.print(i + " " + (n - i));
                return;
            }
        }
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 74;
        findPrimePair(n);
    }
}
```

10. Twisted Prime Number

```
class GFG
{
    static int reverse(int n)
    {
        int rev = 0, r;
        while (n > 0)
        {
            r = n % 10;
            rev = rev * 10 + r;
            n /= 10;
        }
        return rev;
    }
    static boolean isPrime(int n)
    {
        // Corner cases
        if (n <= 1)
            return false;
        if (n <= 3)
            return true;

        // This is checked so that we can skip
        // middle five numbers in below loop
        if (n % 2 == 0 || n % 3 == 0)
            return false;

        for (int i = 5; i * i <= n; i = i + 6)
            if (n % i == 0 || n % (i + 2) == 0)
                return false;

        return true;
    }
```

```
    // function to check Twisted Prime Number
    static boolean checkTwistedPrime(int n)
    {
        if (isPrime(n) == false)
            return false;

        return isPrime(reverse(n));
    }

    // Driver Code
    public static void main(String args[])
    throws IOException
    {
        // Printing Twisted Prime Numbers upto 200
        System.out.println("First few Twisted Prime" +
        " Numbers are :- n");
        for (int i = 2; i <= 200; i++)
            if (checkTwistedPrime(i))
                System.out.print(i + " ");
    }
}
```

11. Almost Prime Numbers

```
class GFG {

    // A function to count all prime factors
    // of a given number
    static int countPrimeFactors(int n)
    {

        int count = 0;

        // Count the number of 2s that divide n
        while (n % 2 == 0) {

            n = n / 2;
            count++;
        }

        // n must be odd at this point. So we
        // can skip one element (Note i = i +2)
        for (int i = 3; i <= Math.sqrt(n);
                                  i = i + 2) {

            // While i divides n, count i and
            // divide n
            while (n % i == 0) {

                n = n / i;
                count++;
            }
        }

        // This condition is to handle the case
        // whien n is a prime number greater
        // than 2
```

```
        if (n > 2)
            count++;

        return (count);
    }

    // A function to print the first n numbers
    // that are k-almost primes.
    static void printKAlmostPrimes(int k, int n)
    {

        for (int i = 1, num = 2; i <= n; num++) {

            // Print this number if it is k-prime
            if (countPrimeFactors(num) == k) {

                System.out.print(num + " ");

                // Increment count of k-primes
                // printed so far
                i++;
            }
        }

        return;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 10, k = 2;
        System.out.println("First " + n + " "
             + k + "-almost prime numbers : ");

        printKAlmostPrimes(k, n);
    }
}
```

12. Check if a number can be written as a sum of ‘k’ prime numbers

13. Special prime numbers

```
vector<int> primes;

// Generating all the prime numbers
// from 2 to n.
void SieveofEratosthenes(int n)
{
    bool visited[n];
    for (int i = 2; i <= n + 1; i++)
        if (!visited[i]) {
            for (int j = i * i; j <= n + 1; j += i)
                visited[j] = true;
            primes.push_back(i);
        }
}

bool specialPrimeNumbers(int n, int k)
```

```
{
    SieveofEratosthenes(n);
    int count = 0;
    for (int i = 0; i < primes.size(); i++) {
        for (int j = 0; j < i - 1; j++) {

            // If a prime number is Special prime
            // number, then we increments the
            // value of k.
            if (primes[j] + primes[j + 1] + 1
                == primes[i]) {
                count++;
                break;
            }
        }

        // If at least k Special prime numbers
        // are present, then we return 1.
        // else we return 0 from outside of
        // the outer loop.
        if (count == k)
            return true;
    }
    return false;
}

// Driver function
int main()
{
    int n = 27, k = 2;
    if (specialPrimeNumbers(n, k))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
    return 0;
}
```

14. Twin Prime Numbers between 1 and n

```
class GFG {

    static void printTwinPrime(int n)
    {
        // Create a boolean array "prime[0..n]"
        // and initialize all entries it as
        // true. A value in prime[i] will
        // finally be false if i is Not a
        // prime, else true.
        boolean prime[] = new boolean[n + 1];

        for (int i = 0; i <= n; i++)
            prime[i] = true;

        for (int p = 2; p * p <= n; p++) {

            // If prime[p] is not changed,
            // then it is a prime
            if (prime[p] == true) {
```

```
                // Update all multiples of p
                for (int i = p * 2; i <= n; i += p)
                    prime[i] = false;
            }
        }

        // to check for twin prime numbers
        // display th twin prime
        for (int i = 2; i <= n - 2; i++) {

            if (prime[i] == true &&
                prime[i + 2] == true)

                // Display the result
                System.out.print(" (" + i + ", " +
                                 (i + 2) + ")");
        }
    }

    // Driver Program to test above function
    public static void main(String args[])
    {
        int n = 25;
        printTwinPrime(n);
    }
}
```

15. K-Primes (Numbers with k prime factors) in a range

```
class GFG {

    static void printKPFNums(int A, int B, int K)
    {
        // Count prime factors of all numbers
        // till B.
        boolean prime[] = new boolean[B+1];
        Arrays.fill(prime,true);
        int p_factors[] = new int[B+1];
        Arrays.fill(p_factors,0);

        for (int p = 2; p <= B; p++)
            if (p_factors[p] == 0)
                for (int i = p; i <= B; i += p)
                    p_factors[i]++;

        // Print all numbers with k prime factors
        for (int i = A; i <= B; i++)
            if (p_factors[i] == K)
                System.out.print( i + " ");
    }
    // Driver code
    public static void main(String args[])
    {
        int A = 14, B = 18, K = 2;
        printKPFNums(A, B, K);
    }
}
```