**Code:**

|  |
| --- |
| // Problem 1. Some Primes  //Find the 24th, 101st and 251st prime number.  // The program below first finds all prime numbers in the range [1...10 000 000]. Use the Sieve of Eratosthenes algorithm.  // and after that prints the required prime numbers only  using System;  using System.Collections.Generic;  using System.Linq;  class SomePrimes  {  static void Main()  {  /\* applying the Sieve of Eratosthenes means, that we will flag all not-prime numbers as not-prime  and whatever has remained outside the set of numbers flagged as not-prime, will be prime \*/  // declarations  // The HashSet<T> class is based on the model of mathematical sets and provides high-performance set operations  // similar to accessing the keys of the Dictionary<TKey, TValue> or Hashtable collections.  // In simple terms, the HashSet<T> class can be thought of as a Dictionary<TKey, TValue> collection without values.  HashSet<int> Primes = new HashSet<int>();  // we will use this bool array, to store the information about postions between 1 and 10 000 000  // at which the numbers are prime or not-prime  bool[] bigArr = new bool[10000000];  // we first set all positions in the bool array to true, all numbers will be considered prime unless proven otherwise  for (int i = 0; i < bigArr.Length; i++)  {  bigArr[i] = true;  }  // this for loop supplies the base numbers for the calculation of the not-prime numbers  for (int i = 2; i < Math.Sqrt(bigArr.Length); i++)  {  // we will do the below calculations only with values of i whose positions in the array are not yet marked as false  // or in other words, those values of i are note yet marked as not-prime  if (bigArr[i])  {  // and here the not prime numbers get calculated: j = i \* i (the square of i is definitely not prime),  // and et each step j increases with one more i; example: j = 2 \* 2; and increases with 2 at each step  // so we get: 2 \* 2 + 2 = 3 \* 2; +2 = 4 \* 2 and we calculate all multiples of i (2) within the given range  for (int j = i \* i; j < bigArr.Length; j = j + i)  {  bigArr[j] = false; // for all multiples of i; we mark their positions in the bool array as false, they are not prime  }  }  }  // if the position of a number into the bool array is still marked as true  // we add the number to the Primes list, as this number is obviously proven to be prime  for (int i = 0; i < bigArr.Length; i++)  {  if (bigArr[i])  {  Primes.Add(i);  }  }  // preparing for print and printing  //string primes = string.Join(", ", Primes);  //Console.WriteLine(primes);  // the elements of a list can be accessed by index; not possiible with HashSet however  // we have initially used HashSet though - because of the extesive calculations necessary,  //which would have failed if we used a lower performance type of collection  List<int> list = Primes.ToList();  Console.WriteLine("The 24th prime number is {0}", list[23]); // the 24th prime number  Console.WriteLine("The 101st prime number is {0}", list[101]); // the 101st prime number  Console.WriteLine("The 251st prime number is {0}\n", list[250]); // the 251st prime number  }  } |

**Output:**

|  |
| --- |
| The 24th prime number is 79  The 101st prime number is 541  The 251st prime number is 1579  Press any key to continue . . . |