Ariel University, School of Computer Science

Introduction to Computer Science, 2026A

Ex0 - Foundation of problem-solving

Abstract:

In this first assignment, we will start a discussion on a few fundamental problems of computer science, particularly a variant of the Goldbach Conjecture. The main task in this assignment is to define a detailed algorithm (using free text and pseudocode) for efficiently testing the following version of the Goldbach Conjecture. Given a natural (even) number n, you are **required** to design the following functions: the design should be 100% correct and as efficient as possible.

Done! Class solution can be found here:

https://github.com/benmoshe/Intro2CS_ArielU_2026A/tree/main/src/assignments/Ex0/sol

a) **isPrime(n)** → boolean: return true if and only if n is a prime number.
b) **getPrimePair(s, n)** → integer: given a positive integer s and an even positive number n, find the smallest **prime** number (p1>=s) such that p2 = p1+n is also a prime number.
c) **getClosestPrimePair(s,n)** → integer: given a positive integer s and an even positive number n, find the smallest **prime** number (p3) such that:
    i) p3 >= s
    ii) p4 = p3 + n is a prime number.
    iii) There are no prime numbers in the range of [p3+1, p4-1].
d) **getMthClosestPrimePair(m, n)** → integer: given a non-negative integer m and an even positive number n, find the prime number (p5) such that
    i) p6 = p5 + n is a prime number.
    ii) There are no prime numbers in the range of [p5+1, p6-1].
    iii) There are exactly **m** ranges of **closestPrimePairs** before (below) p5.

To Do:

This assignment is divided into 3 main parts - solve them in order!

Part 1 (start in class 1):

1. Understand the problem and read the related work.
2. Test the following “toy example”,

   Example 1:

   Enter a natural even number in range of [2,100] : 2

   a) First prime pair of range = 2 : [3, 5]

   b) Prime pair above 1024 with range of 2 : [1031, 1033]

   c) The Prime closest pair of range 2 : [3, 5]

   d) The 5'th closest prime pair of range 2 : [41, 43]

   e) The program runtime is: 0.014 seconds
3. Run the given solution Ex0 (very slow) solution, by opening a terminal and running the following command: java -jar Ex0_sol.jar-out.jar (you may also run the .jar file from your IDE).

```
 java -jar Ex0_sol.jar-out.jar
Enter a natural even number in range of [2,100] : 8
a) First prime pair of range = 8 : [3, 11]
b) Prime pair above 1099511627776 with range of 8  : [1099511628323, 1099511628331]
c) The Prime closest pair of range 8 : [89, 97]
d) The 32'th closest prime pair of range 8 : [2531, 2539]
e) The program runtime is: 2.993 seconds
```

Part 2 (start after class 1):

4. Open a Java project with all 3 .java files of Ex0, you should only edit Ex0.java.
5. Write in your own words the required functionality for each of the four functions - your writing should be in English - in the documentation of Ex0.java.
6. Write a related pseudo-code for all the requirements four functions (you are mostly welcome to use the isPrime example below - but improve it). Your documentation should be above each related function (in Ex0.java).

Part 3 (start in class 2):

7. Implement your code (once you complete part 2).
8. Test your solution against the suggested JUnit Test class.
9. Test your runtime solution on input 98 and report your results in this Form. You can see the reported results here.

Notes:

1. The IsPrime pseudo-code that is given to you below can be improved significantly!!
2. The Ex0 jar file (given to you) is running in “slow motion” - make sure your solution runs faster than it.
3. In case n is not an even integer >= 2, the program should print an error and exit. You may assume that n <= 100
4. You can discuss this assignment with anyone in class, but when writing the solution, do it **yourself**! Please go over this document, which covers the School’s honesty policy.
5. The suggested algorithm should be as efficient as possible. But make sure it is 100% correct!
6. Write your pseudocode as simply as possible - based on simple commands of “if”, “while”, ”variables”, and “return” (see example below).
7. The key to a good implementation (and design) is a proper division into logical functions.
8. Submission guidelines: Your solution should be written as a single Java file named Ex0.java. Make sure you write your ID at the top of the page. English submissions are required. Your solution should be submitted to Moodle according to the instructions, as presented to you in the TA sessions (Deadline TBD@23:59).

Appendix:

Example: for writing pseudocode for the is-Prime Algorithm:

This algorithm gets a natural number (n) and tests if n is a prime number (or not). The algorithm works by testing all the natural numbers in the range [2,n-1] if any of them divides n, if so, n is not a prime number.

```
1.      Input(n >1)              // assuming n is a natural number n>0
2.      ans =  true              // ans is the variable representing “is n a prime”
3.      i = 2                    // init value of I is 2 (the divider)
4.      while (i<n ) {           // loop over all number 2<=i<n
5.          t = n%i              // t is the value of n mod i.
6.          if (t=0) {           // if t is zero, n is NOT a prime number (I divide it)
7.              ans = false      //change the value of ans to false (not prime).
            } // if               // the end of the “if” block
8.          i = i+1              // increment i by 1.
        } // while                // the end of the “while” block
9.      return (ans)             // returns the answer of the algorithm
```

Example for a relatively efficient implementation of Ex0 (run on MAC M4, java 25):
Enter a natural even number in range of [2,100] : 100
a) First prime pair of range = 100 : [3, 103]
b) Prime pair above 1099511627776 with range of 100 : [1099511627791, 1099511627891]
c) The Prime closest pair of range 100 : [396733, 396833]
d) The 32'th closest prime pair of range 100 : [9413533, 9413633]
e) The program runtime is: 0.333 seconds

Grades:
Here is a preliminary list of grades and remarks.
In case you want to appeal, make sure to go over this document and write a detailed explanation to this email: BodekMavo2026@gmail.com.
Keep in mind that any appeal will cause a full recheck of your original work - your grade will change accordingly. The deadline for appealing on Ex0 is 27.11

Here is a nice student solution:

```java
package assignments.Ex0;

/**
* This class is the basis for Ex0 (your first assignment),
* The definition of the Ex0 can be found here:
https://docs.google.com/document/d/1UtngN203ttQKf5ackCnXs4UnbAROZWHr/
edit?usp=sharing&ouid=113711744349547563645&rtpof=true&sd=true
* You are asked to complete the functions below and may add
additional functions if needed.
*/
public class Ex0 {
   public final static long ID = 203712104;  // Do update your ID here

   private static final int MAX_LIMIT = 20000000;
   private static boolean[] isPrimeArray;
   private static java.util.ArrayList<Long> primesList;

   /**
    * This function checks if n is a prime number.
    * Notes:
    * i) I've optimized the algo by using Sieve of Eratosthenes: O(1)
lookup for n <= 20M, and O(sqrt(n)) for larger n
    * ii) Computes all prime numbers up to MAX_LIMIT on class init
    *
    * @param n (Integer) - represented as long
    * @return true if and only if there is no integer (p) within the
range of [2,n) which divides n.
    *
    */
   public static boolean isPrime(long n) {
       if (n < 2) {
           return false;
       }

       if (n <= MAX_LIMIT) {
           return isPrimeArray[(int) n];
       }

       // Fall back to less efficient algo
       if (n == 2) {
           return true;
       }
       if (n % 2 == 0) {
           return false;
```

```
        }

        for (long p = 3; p * p <= n; p += 2) {
            if (n % p == 0) {
                return false;
            }
        }
        return true;
    }

    /**
     * This function finds the first prime integer (p1) >= start, for
which p2=p1+n is also a prime number.
     *
     * @param start - a starting value from which p1 should be
searched for.
     * @param n     - a positive (even) integer value.
     * @return the first prime number p1 such that: i) p1>=start, ii)
p1+n is a prime number.
     * in case a wrong value is given to the function
     * (n<0 or n is an odd number) the function should return -1.
     *
     */
    public static long getPrimePair(long start, long n) {
        long ans = -1;
        if (n >= 2 && n % 2 == 0) {
            /// Add your code below ///
            long current = start;
            while (true) {
                /** Check if current is prime */
                if (isPrime(current)) {
                    /** Check if current + n is also prime*/
                    if (isPrime(current + n)) {
                        /** Found a valid prime pair*/
                        ans = current;
                        break;
                    }
                }
                /** Move to next candidate*/
                current++;
            }
        }
        return ans;
    }

    /**
     * This function compute the first prime number p1 for which:
     * i) p1 >= start (p1 is a prime number)
     * ii) p1+n==p2 ia a prime number.
     * iii) there are no prime numbers in the (p1,p2) range.
     * <p>
     * I've optimized this function by using pre-computed primes list
     *
```

```java
    * @param start a positive integer which is the lower bound of p1.
    * @param n     - a positive even integer.
    * @return a prime number p1>=start that the following prime
number is p1+n.
    */
   public static long getClosestPrimePair(long start, long n) {
       /// Add your code below ///
       /** didn't implement binary search by myself, the lecturer
said we can use which tool that we know.*/
       int index = java.util.Collections.binarySearch(primesList,
start);

       if (index < 0) {
           // start is not prime, get insertion point
           index = -(index) - 1;
       }
       // Now index points to first prime > start
       while (index < primesList.size() - 1) {
           long p1 = primesList.get(index);
           long p2 = primesList.get(index + 1);  // Next prime in
sequence

           // Check if gap equals n
           if (p2 - p1 == n) {
               return p1;
           }

           index++;
       }

       // Should not reach here if MAX_LIMIT is sufficient
       return -1;
   }

   /**
    * This function compute the m'th positive integer p1 for which:
    * i) p1 is a prime number.
    * ii) p1+n==p2 ia a prime number.
    * iii) there are no prime numbers in the (p1,p2) range.
    *
    * @param m a none negative integer.
    * @param n - a positive even integer.
    * @return a prime number p1>=start that the following prime
number is p1+n.
    *
    */
   public static long getMthClosestPrimePair(int m, long n) {
       if (m < 0 | n < 0 | n % 2 != 0) {
           System.err.println("Invalid input: got m=" + m + ", n=" + n
+ "  |  m should be >=0 & n should be a positive even integer ");
           return -1;
       }
       /// Add your code below ///
```

```
        // Find the m'th closest prime pair
        // Use getClosestPrimePair repeatedly, starting from where the
previous search ended
        long current = 2;    // Start from the first prime

        for (int i = 0; i <= m; i++) {
            // Find the next closest prime pair starting from current
            current = getClosestPrimePair(current, n);

            // If this is not the last iteration, move to the next
candidate
            // to search for the next closest pair
            if (i < m) {
                current++;
            }
        }

        return current;

    }

    /// //////// Private Functions - you are welcome to add additional
(private) functions below.

    static {
        sieveOfEratosthenes(MAX_LIMIT);
    }

    /**
     * Sieve of Eratosthenes: Pre-computes all primes nums up to
geiven number
     *
     * @param limit - the upper bound for prime generation
     */
    private static void sieveOfEratosthenes(int limit) {
        isPrimeArray = new boolean[limit + 1];
        java.util.Arrays.fill(isPrimeArray, true);
        isPrimeArray[0] = isPrimeArray[1] = false;  // 0 and 1 are not
prime

        // Sieve algorithm
        for (int p = 2; p * p <= limit; p++) {
            if (isPrimeArray[p]) {
                // Mark all multiples of p as not prime
                for (int i = p * p; i <= limit; i += p) {
                    isPrimeArray[i] = false;
                }
            }
        }

        // Build list of primes for another purposes (sequnce access)
        primesList = new java.util.ArrayList<>();
```

```
        for (int i = 2; i <= limit; i++) {
            if (isPrimeArray[i]) {
                primesList.add((long) i);
            }
        }
    }
}
```