1. Homework 1

Not Final

Posted: June/3/2021
Due: August/30/2021 24.00

The solutions for this homework are due August/30/2021 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Submission of your Homework

This course uses the myCourses submission functionality to allow students to electronically submit their work.

I suggest to submit often, meaning submit long before the deadline. We will only accept submitted code. You will receive 0 points for the hw, if your code is not submitted.

You have to submit all files for your solutions for all parts of a give home work in one zip file. The name of the zip file has to be: *your_login_name.gz*, or *your_login_name.gz your_login_name is the login name of one team member.*

1.2. Programming Standard

Please make sure that your code is compatible with the Coding Standard (https://www.cs.rit.edu/˜f2y-grd/java-coding-standard.html)

We will take points of, if your code does not follow this standard.

1.3. Teamwork

All work has to be submitted as a team of 2. You have to appear to the grading sessions on time. You have to select a grading slot during the first week. A schedule will be posted at the grad lab door at the beginning of the first week.

You will receive 0 points if you are late for your grading session.

The graders determine who answers the questions.

The signup sheet for your team can be found here: Need to get done (https://docs.google.com/spreadsheets)

1.4. Homework 1.1 (10 Points)

Objective: Compilation of a Java program, designing, implementing, and testing of an algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

All homework are submitted as a team of 2.

Definition: Prime Number

A number p is a prime number if p has the following properties:

• p must be a integer

• $p > 1$ and

• the factors of p are 1 and itself.

Definition: Nearly Perfect Number

A number *n* is a nearly perfect number, if the sum of its first *k* prime divisors adds up to *n*. An example would be: 77 is a nearly perfect number. (2 + 3 + 5 + 7 + 11 + 13 + 17 + 19)

A counter example would be: 6 (2 + 3)

The prime numbers between 1 and 77 are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 77

The following program prints out prime numbers.

```
class Prime {

  public static boolean isPrime(int n) {

        for ( int index = 2; index < n; index ++ ) {
                if ( n % index  == 0 )
                         return false;
        }

        return true;
  }
  public static void main( String args[] ) {
    for ( int index = 2; index <= 10; index ++ )
        if ( isPrime(index) )
                System.out.println(index + " " );
  }
}
```

```
Source Code: Src/21/Prime.java
```

Your Work:

Modify *Prime.java* in such a way that is prints out all nearly perfect numbers between *n* 2 and 1000 for the following properties:

```
% java NearlyPerfectNumber
7: true
5      is a nearly perfect number. (2 + 3)
10     is a nearly perfect number. (2 + 3 + 5)
17     is a nearly perfect number. (2 + 3 + 5 + 7)
28     is a nearly perfect number. (2 + 3 + 5 + 7 + 11)
 ...
963    is a nearly perfect number. (2 + 3 + 5 + 7 + 11 + 13 + 17 + 19 + 23 + 29 + 31 + 37
```

Your output does not have to be identical to mine, but similar.

Idea for a Solution:

- Design your algorithm on paper
- Run your algorithm by hand
- Only if you are sure it works, implement it

Requirements:

• You have to name your program NearlyPerfectNumber.java

• You can only use basic arithmetic operations, casting, boolean expressions, print statements, Math.sqrt, basic types and native arrays.

• You can not use any publicly available class or library which can determine if a number is a prime number.

• Your program has to compute the perfect numbers. In other words your program can not be something like:

```
class PerfectWrong {

  public static void main( String args[] ) {
        System.out.println("5       is a perfect number. (2 + 3)");
        System.out.println("10      is a perfect number. (2 + 3 + 5)");
        System.out.println("17      is a perfect number. (2 + 3 + 5 + 7)");
        System.out.println("28      is a perfect number. (2 + 3 + 5 + 7 + 11)");
        System.out.println("41      is a perfect number. (2 + 3 + 5 + 7 + 11 + 13)
        System.out.println("58      is a perfect number. (2 + 3 + 5 + 7 + 11 + 13
        System.out.println("77      is a perfect number. (2 + 3 + 5 + 7 + 11 + 13
  }
}
```

```
Source Code: Src/21/PerfectWrong.java
```

This solution would be graded with 0 points.

Submission:

Submit your files via myCourses.

1.5. Homework 1.2 (10 Points)

Objective: Creating and implementing an algorithm.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have a given set of buckets of water. Each bucket is completely full and the water in a bucket it measured in liters. You also know the volume of each bucket. You have one empty bucket of a known volume. You have to write a program which can find out if you can completely fill the empty bucket using all of the water of some buckets, and additionally using the maximum or minimum amount of buckets..

Explanation:

Examples for using the maximum amount of buckets:

Assume you have the following bucket volumes: 1l, 2l, 5l. You could answer the question with yes for a bucket of size 4l: can not be done

Assume you have the following bucket volumes: 1l, 2l, 5l. You could answer the question with yes for a bucket of size 3l = 1l + 2l.

Assume you have the following bucket volumes: 1l 2l, 3l 4l. You could answer the question with yes for a bucket of size 5l = 1l + 4l, or 2l + 3l.

Assume you have the following bucket volumes: 1l 1l, 2l, 3l 5l. You could answer the question with yes for a bucket of size 2l = 1l + 1l.

Examples for using the maximum amount of buckets:

Assume you have the following bucket volumes: 1l, 2l, 3l, 5l. You could answer the question with yes for a bucket of size 3l = 3l.

Assume you have the following bucket volumes: 1l 1l, 2l, 3l. You could answer the question with yes for a bucket of size 5l = 1l + 4l, or 2l + 3l.

Your Work:

Your work is to write a program which can find out if you can completely fill the empty bucket using all of the water of some buckets using the maximum number of buckets. If no commandline argument is supplied, then ypur program has to find the maximum amount of buckets, otherwise the minimum amount of buckets.

• You can use arrays

• You can hardcode all values in your program

• You can not use any classes besides the String class.

• Your program should only print out: if can can be done and if it can be done the volumes used. For example, if using the above volumes: 3l: yes - 1l + 2l.

This is a snippet of my code:

```
public class Water {
    static int[] myVolumes = { 1, 1, 2, 4, 5, 6 };
    static int soManyBuckets = myVolumes.length;
    static int[] bucketsToFill = { 1, 2, 3, 4, 6, 7, 8, 9 };
    static boolean largestAmountOfUsedBuckets = true;
    ...
    public static void main( String[] arguments ) {

        if ( arguments.length > 0 )
                largestAmountOfUsedBuckets = false;
```

My program produces the following output:

```
# java Water
Available full bucket volumes: 1l 1l 2l 4l 5l 6l
1l:          yes: 1l = 1l
2l:          yes: 2l = 1l + 1l
3l:          yes: 3l = 2l + 1l
4l:          yes: 4l = 2l + 1l + 1l
6l:          yes: 6l = 4l + 1l + 1l
7l:          yes: 7l = 4l + 2l + 1l
8l:          yes: 8l = 4l + 2l + 1l + 1l
9l:          yes: 9l = 5l + 2l + 1l + 1l
# java Water x
Available full bucket volumes: 1l 1l 2l 4l 5l 6l
1l:          yes: 1l = 1l
2l:          yes: 2l = 2l
3l:          yes: 3l = 2l + 1l
4l:          yes: 4l = 4l
6l:          yes: 6l = 6l
7l:          yes: 7l = 5l + 2l
8l:          yes: 8l = 6l + 2l
```

```
91:          yes: 91 = 51 + 41
```

Your output does not have to be identical to mine, but similar.

Idea for a Solution:

It might help to understand how to find all possible combinations of *n* items. For a moment assume the set is { 1, 2, 3 }.

```
combination(a, b)                                        =
combination(a, combination(b))
        combination(b) = { b, ∅ }
combination(a, { b, ∅ } )  ∪ { b, ∅ } )    =
    { a ∪ { b, ∅ } }  ∪ { b, ∅ }             =
    { ab, a, ∅ } ∪ { b, ∅ }                =
    { ab, a, ∅,  b }
```

This is the staring point of a recursive solution.

I suggest the following algorithm to solve this problem:

• Design your algorithm on paper

• Run your algorithm by hand

• Only if you are sure it works, implement it

Requirements:

• You have to name your program *Water.java*

• Your program must produce the correct output

Submission:

Submit your files via myCourses.

1.6. Homework 1.3 (10 Points)

Objective: Designing and implementing an algorithm and design a testing strategy.

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to implement an algorith to determine the maximum number of kings you can put on the not traditional nxn chess board, so such no king is not in check. The chess board has squares on it on which a king can not be place. You can only put a king on the squares white or black square. This might be helpful https://en.wikipedia.org/wiki/Rules_of_chess (https://en.wikipedia.org/wiki/Rules_of_chess)

Explanation:

The board used is made up of white, black, and '.' squares. You can only place a king on an white or black square. You can not place a king on blocked squares. The object is to place the maximum number of kings on he board. The board below shows an example.

```
 w b w b w b
 . w . w . w
 w . . . w b
 b . b . b w
 . . . . . b
 b w b w b w
```

- 'w' stands for white square.
- 'b' stands for black square.
- '.' for blocked square.

Your Work:

You should develop this program in stages. Start with the methods you can implement and test first. Develop a test scenario before you develop the method.

The chess board can be hard coded. The code creating my board looks like:

```
int MAX_ROWS           = 6;
int MAX_COLUMNS        = 6;
char theBoard[][]            = new char[1+MAX_ROWS][MAX_COLUMNS];

    public void initTheBoard() {
        soManyKings = 0;
        maxNofKings = 0;
        for (int row = 0; row < MAX_ROWS; row ++ )       {
                for (int column = 0; column < MAX_COLUMNS; column ++ )  {
                        theBoard[row][column] = whatColor(row, column);
                }
        }
    }
    public void createWall() {
        // theBoard[2][0] = NOT_LEGAL;
        theBoard[2][1] = NOT_LEGAL;
        theBoard[2][2] = NOT_LEGAL;
        theBoard[2][3] = NOT_LEGAL;
        theBoard[4][1] = NOT_LEGAL;
        theBoard[4][2] = NOT_LEGAL;
        theBoard[4][3] = NOT_LEGAL;
        theBoard[4][1] = NOT_LEGAL;
        theBoard[3][1] = NOT_LEGAL;
        theBoard[3][3] = NOT_LEGAL;
        for ( int index = 0; index < MAX_ROWS-1; index +=2 )    {
                theBoard[1][index] = NOT_LEGAL;
                theBoard[MAX_ROWS-2][index] = NOT_LEGAL;
        }
    }
```

My program produces the following output:

```
% java Board
maximum number of kings = 9
 K b K b K b
 . w . w . w
 K . . . K b
 b . K . b w
 . . . . . K
 K w K w b w
```

An incorrect solutionn would be, because the *K*.

```
 K b K b K b
 K w . w . w
 . . . . K b
 b . K . b w
 . . . . . K
 K w K w b w
```

Your output does not have to be identical to mine, but similar.

Requirements:

You have to name the file *Board.java*.

Idea for a Solution:

• Design your algorithm on paper

• Run your algorithm by hand

• Only if you are sure it works, implement it

Submission:

Submit your files via myCourses.