**Please, upload your answers in Canvas: assignment PC LAB – April 7**

Name: Vu The Doan

Student nr: 12918687

PC2a – Looping

**1.** Write a script *beautyofmathstr* that produces the following output:

> source(' beautyofmathstr.R')

1 x 8 + 1 = 9

12 x 8 + 2 = 98

123 x 8 + 3 = 987

1234 x 8 + 4 = 9876

12345 x 8 + 5 = 98765

123456 x 8 + 6 = 987654

1234567 x 8 + 7 = 9876543

12345678 x 8 + 8 = 98765432

123456789 x 8 + 9 = 987654321

- Use strings only!

- Note: "123" is equal to paste("12", "3", sep="")

- You can use

cat(sprintf("%s x 8 + %d = %s\n", xstr, i, ystr))

to display your results (see also video *Additional Topics*). Here %s will be replaced by the string xstr, %d will be replaced by the integer i and the second %s represents ystr.

getwd()

setwd("D:/UvA/Year 1/Block 5/Programming and Numerical Analysis/Week 2")

rm(list = ls())

"1 x 8 + 1 = 9

12 x 8 + 2 = 98

123 x 8 + 3 = 987

1234 x 8 + 4 = 9876

12345 x 8 + 5 = 98765

123456 x 8 + 6 = 987654

1234567 x 8 + 7 = 9876543

12345678 x 8 + 8 = 98765432

123456789 x 8 + 9 = 987654321"

xstr <- ""

i <- 1

ystr <- ""

for(i in c(1:9)) {

xstr <- paste0(xstr, i)

ystr <- paste0(ystr, 10-i)

cat(sprintf("%s x 8 + %d = %s\n", xstr, i, ystr))

}

**2.** The reciprocal of the mathematical constant *e* can be approximated as follows:



Write a R-script that iterates the right-hand side of the formula for values of *n* until the difference between the exact value (using the build-in function exp(-1)) and its approximation for the first time is smaller than 0.0001. The script should display the exact value of , its approximation in 4 decimals and the value found for *n*.

Hint: use a while loop

Example output:

The build-in value of e^(-1) is 0.3679

The approximation is 0.3678

The value for n is ...

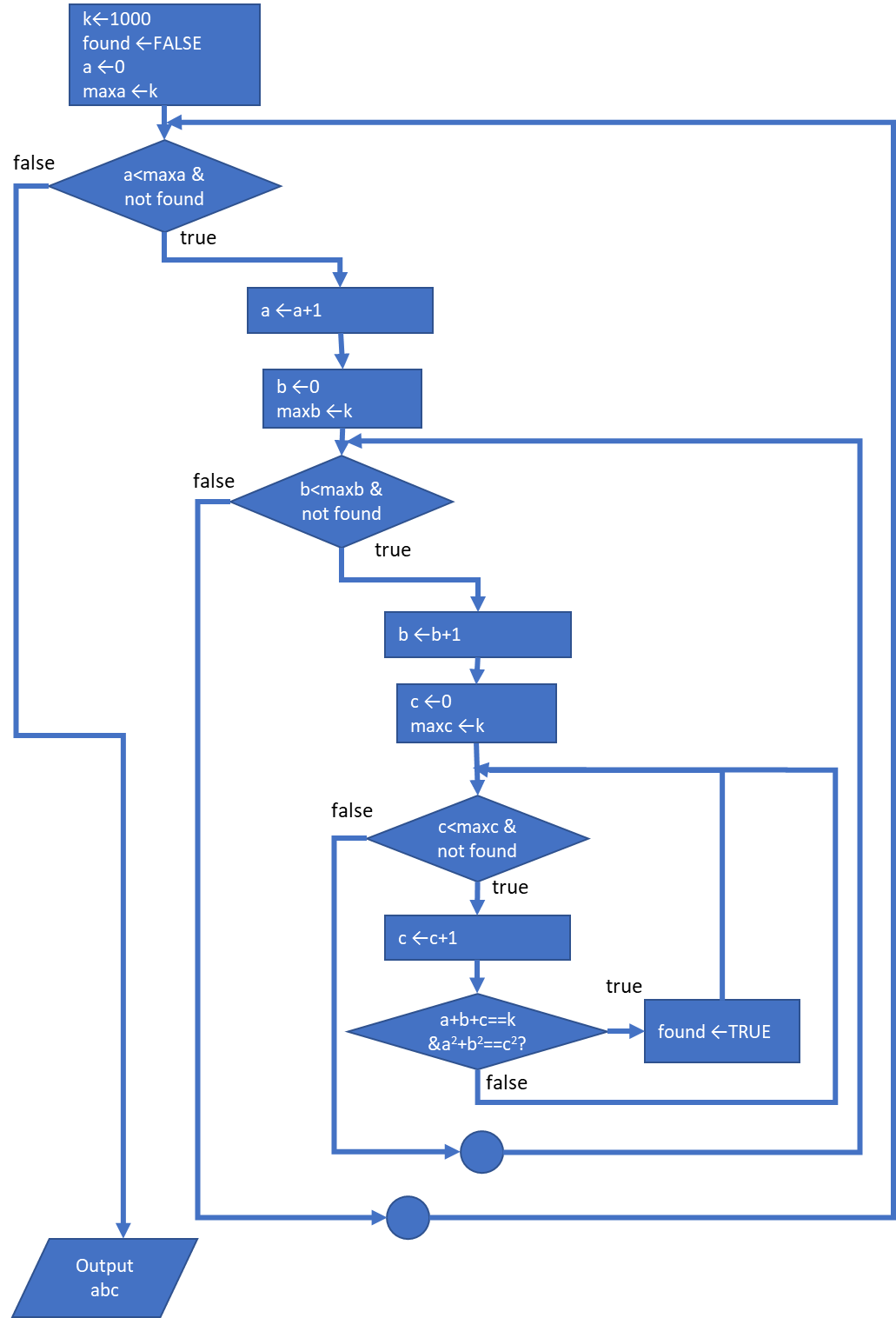
The value for n such that the difference is for the first time smaller than 0.0001 equals?

**Answer**:

3. A Pythagorean triplet is a set of three positive integers , for which, . For example, (9 + 16 = 25). There ***exists exactly one*** Pythagorean triplet for which . Find the product .

In this question, you will investigate 3 algorithms in increasing order of efficiency.

a. First, we use a naïve algorithm, called *Algorithm I*, with three while loops for finding as follows:



**b**. Let’s analyze the program somewhat more. First, since squares are always positive and , we conclude that and . Next, since we can switch the roles of and , we can without loss of generality assume Finally, since , we can derive the following bounds due to the ordering of , and :

(i)

(ii)

(iii)

Copy the source code of sub question (**a**) and modify the algorithm to incorporate restriction (i)-(iii). We call this *Algorithm II*. Note that the while loop for can be removed.

**c**. We can improve the algorithm as follows. Replacing by in the Pythagorean condition, we get

After simplifying, we get

Solving this equation for gives

.

Setting the denominator to , we can rewrite the term as

,

so that we obtain

Since is a positive integer, we know that

(i) [since

(ii) m>500 [since , should be smaller than ]

(iii) 500,000 should be a multiple of , ie. the remainder of 500,000 divided by should be zero.

*Pseudo code for Algorithm III*:

for m in 501 to 1000

if (500000 mod m = 0) then

a ←1000 – (500000 div m)

b ← 1000 – m

c ← 1000 – a – b

exit [for]

Code this algorithm in R.

d. To determine the efficiency, we compare the execution length. For, this you can use (you may need to install the library tictoc first):

library(tictoc)

tic()

# … code

#toc()

Fill in the following table:

|  |  |  |
| --- | --- | --- |
| Algorithm | Time | Relative efficiency to A |
| I |  | 1.0 |
| II |  |  |
| III |  |  |

Note that the execution time will be different for all due to differences in computer power! How much faster are algorithms II and III with respect to the naïve I?