PC2a – Looping

1.

```
leftnr = ""
rightnr=""
for(i in 1:9){
  leftnr = paste(leftnr, as.character(i),sep="")
  rightnr = paste(rightnr, as.character(10-i),sep="")
  cat(sprintf('%s x 8 + %d = %s\n', leftnr, i, rightnr))
}
```

2.

```
# Approximates 1/e as (1-1/n)^n, and determines
# the value of n required for accuracy to 4 dec. places

actual <- exp(-1)
diff <- 1  # initialize difference between actual and approximation
n <- 0     # counter

while(diff >= 0.0001){
  n <- n+1
  approx <- (1-1/n)^n;
  diff <- abs(actual-approx);
}

cat(sprintf('The build-in value of e^(-1) is %.4f\n',actual))
cat(sprintf('The approximation is %.4f\n',approx))
cat(sprintf('The value for n is %d\n',n))
```

The value for n such that the difference is for the first time smaller than 0.0001 equals?
Answer: **1840**

3.a. First, we use a naïve algorithm, called *Algorithm I*, with three `while` loops for finding (a, b, c) as follows:

```
library(tictoc)
tic()
k<-1000
found<-FALSE
a<-0
maxa<-k
while( (a<maxa) & !found ){
  a<-a+1
  b<-0
  maxb<-k
  while( (b<maxb) & !found ){
    b<-b+1
    c<-0
    maxc<-k
    while( (c<maxc) & !found ){
      c<-c+1
      if( (a+b+c==k) & (a^2+b^2==c^2) ) {
        found<-TRUE
      }
    }
  }
}
cat("Solution:",a*b*c,"\n")
toc()
```

b. Let's analyze the program somewhat more. First, since squares are always positive and $c^2 = a^2 + b^2$, we conclude that $c > a$ and $c > b$. Next, since we can switch the roles of a and b $(3^2 + 4^2 = 4^2 + 3^2)$, we can without loss of generality assume $0 < a < b < c$. Finally, since $a + b + c = 1000$, we can derive the following bounds due to the ordering of a, b and c:

(i) $1 \leq a < k/3$

(ii) $b > a$

(iii) $c = 1000 - a - b$

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 c can be removed.

```
tic()
found<-FALSE
a<-0
maxa<-k %/%3
while( (a<maxa) & !found ){
  a<-a+1
  b<-a
  maxb<-(k-a) %/% 2
  while( (b<maxb) & !found ){
    b<-b+1
    c<-k-a-b
    if( (a+b+c==k) & (a^2+b^2==c^2) ) {
      found<-TRUE
    }
  }
}
```

```
cat("Solution:",a*b*c,"\n")
toc()
```

c.

Code this algorithm in R.

```
tic()
for( m in 501:1000 ){
  if( 500000 %% m==0){
    a<-1000-(500000 %/% m)
    b<-1000-m
    c<-1000-a-b
    break
  }
}
cat("Solution:",a*b*c,"\n")
toc()
```

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	58.64	1.0
II	0.09	58.64/0.09=651
III	0.05	58.64/0.05=1,173

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?
We see that Algorithm II is 651 times faster than Algorithm I, while Algorithm III is more than 1173 times faster.