#HW1

#Author: Daniel Rozen, drozen3

```
# #2.Implement a function that computes the log of the factorial value of an
# integer using a for loop. Note that implementing it using log(A)+log(B)+
#  · · · avoids overflow while implementing it as log(A · B · · · · ) creates an
# overflow early on.

myFun2 = function(x) {
 if (is.numeric(x)) { # check if integer
 sum=0
 for (i in 1:(x)){
  sum=sum+log(i)
 }
 return(sum)
 }
}

# 3. Implement a function that computes the log of the factorial value of an
# integer using recursion.

myFun3= function(x) {
 if (is.numeric(x)) { # check if integer

  if (x==1) {
   return(log(1))
  }

  currentSum = log(x) + myFun3(x-1)
```

```
    return(currentSum)

  }
}

# 4. Using your two implementations of log-factorial in (2) and (3) above, compute
# the sum of the log-factorials of the integers 1, 2, . . . ,N for various N
# values.

# using for loop (probably simplest) for the sum in both cases

sumFun2= function(x) {
  if (is.numeric(x)) { # check if integer
    sum=0
    for (i in 1:(x)){
      sum=sum+myFun2(i)
    }
    return(sum)
  }
}

sumFun3= function(x) {
  if (is.numeric(x)) { # check if integer
    sum=0
    for (i in 1:(x)){
      sum=sum+myFun3(i)
    }
    return(sum)
  }
```

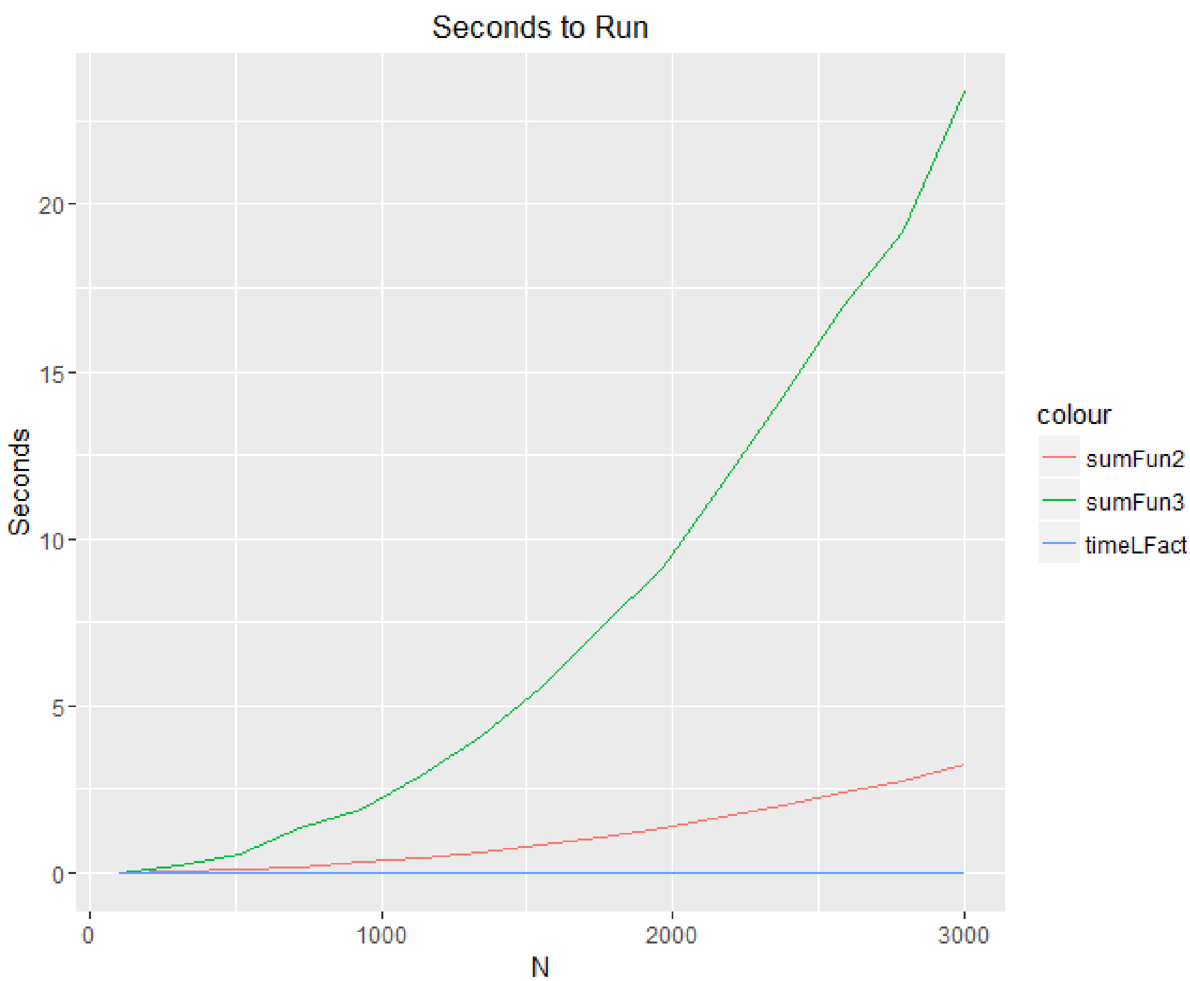
```
}
```

```
# 5. Compare the execution times of your two implementations for (4) with an
# implementation based on the official R function lfactorial(n). You
# may use the function system.time() to measure execution time.



# an implementation based on the official R function lfactorial(n)
sumLFact= function(x) {
 if (is.numeric(x)) { # check if integer
   sum=0
   for (i in 1:(x)){
     sum=sum+lfactorial(i)
   }
   return(sum)
 }
}

#What
# are the growth rates of the three implementations as N increases? Use the
# command options(expressions=500000) to increase the number of
# nested recursions allowed. Compare the timing of the recursion implementation
# as much as possible, and continue beyond that for the other two
# implementations.

options(expressions=500000)

#create chart
```

```r
N = seq(100, 3000, length = 15)
time2 = c()
time3 = c()
timeLFact = c()

for (n in N) {
  time2= c(time2, system.time(sumFun2(n))[3]) # use elapsed time
  time3= c(time3, system.time(sumFun3(n))[3])
  timeLFact= c(timeLFact, system.time(sumLFact(n))[3])
}

#print dataframe

df=data.frame(N=N, time2=time2, time3=time3, timeLFact=timeLFact)
df

# plot run time as a function of array size for R and
# .C implementations

library(ggplot2)

ggplot(df, aes(N)) +
  geom_line(aes(y = time2, color = "sumFun2")) +
  geom_line(aes(y = time3, color = "sumFun3")) +
  geom_line(aes(y = timeLFact, color = "timeLFact")) +
        xlab("N") +
        ylab("Seconds") +
        ggtitle("Seconds to Run")
```

<u>Table of Execution times (in seconds) of Implementations vs N</u>

	N	time2	time3	timeLFact
1	100.0000	0.00	0.02	0.00
2	307.1429	0.03	0.20	0.00
3	514.2857	0.09	0.54	0.00
4	721.4286	0.19	1.34	0.00
5	928.5714	0.32	1.88	0.00
6	1135.7143	0.46	2.92	0.00
7	1342.8571	0.63	4.12	0.00
8	1550.0000	0.86	5.57	0.00
9	1757.1429	1.06	7.34	0.00
10	1964.2857	1.35	9.12	0.00
11	2171.4286	1.67	11.69	0.01
12	2378.5714	2.04	14.25	0.00
13	2585.7143	2.43	16.92	0.00
14	2792.8571	2.75	19.15	0.00
15	3000.0000	3.26	23.36	0.00



Regarding the growth rates of the three implementations as N increases: as we see from the above table and graph, sumFun3 seems to grow exponentially, sumFun2 also seems to but to a lesser extent, and sumLFact take negligible time and it's growth rate seems to be unobservable.