```
#HomeWork1
#Name: Ajay Joshi
#Course: CSE 6242
#Git:903270434
#==========================================================================

#2.Implement a function that computes the log of a factorial value of an integer using a for
loop.
LoopLogFactorial <- function(a) {
 if (a < 1) return(Inf)
 if (a == 1) return(0)

 val <- 0
 for (i in 2:a) {
  val <- log(i) + val
  #print(paste("The factorial of",i,"is",val))
 }
 return(val)
}

#==========================================================================

#[Alternative]
#2. Implement a function that computes the log of a factorial value of an integer using a for
loop.
#Stirling's approximation gives an approximate value for the log of a factorial

#Stirling's approximation gets better as N gets higher
StirlingFactorial <- function(N) {
 if (N == 1) return(0)
 res <- (N+0.5) * log(N) - N + log(2* pi)/2;
 return(res)
}

#==========================================================================

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

options(expressions = 500000)
Recrursion.factorial <- function(N){
 #base case
 if (N < 1) return(Inf)
 if (N == 1) {
```

```
   #PrintIt(N,0)
   return(0)
  }
  val <- Recrursion.factorial(N-1)
  val <- sum(val,log(N))
  # PrintIt(N,val)
  return(val)
}

PrintIt <- function(N, factorialValue) {
  print(paste("The log factorial of",N,"is",factorialValue))
}

#=============================================================================

# 4.Using your two implementation of log factorial in 2 and 3 above,
#compute the sum of the log factorial of the integer 1,2,..N for various N values.

#Sum of factorial log using the implementation on question 2
Sum.LoopFactorial <- function(N){
  result <- 0
  summation <- 0
  for (i in 1:N) {
    #print(paste("The factorial of log(",i,"!) is",LoopLogFactorial(i)))
    result[i] <- LoopLogFactorial(i)
    summation <- sum(summation,result[i])
  }
  return(summation)
}


#Sum of factorial log using the implementation on question 3
Sum.RecrusionFactorial <- function(N){
  result <- 0
  summation <- 0
  for (i in 1:N) {
    #print(paste("The factorial of log(",i,"!) is",Recrursion.factorial(i)))
    result[i] <- Recrursion.factorial(i)
    summation <- sum(summation,result[i])
  }
  return(summation)
}

#=============================================================================
```

#5.compare the execution times of your two implementations for (4) with implementation based on the offical lfactorial(n).

```
# summation of Log(N!) from 1 to N is calculated using R inbuilt function.
Sum.LFacorial <- function(N) {
  a <- 0
  for (i in 1:N) {
    a <- sum(a,lfactorial(i))
  }
  return(a)
}

#-------------------------------------------------------------------------------

TestMethod <- function() {
  #assign the value of N
  N <- 3000

  #store execution times values in a vector
  ElapsedTimes <- 0
  ElapsedTimes[1:3] <- 0

  #---- system version ----
  # Start the clock!
  ptm <- proc.time()
  Sum.LFacorial(N)
  # Stop the clock
  SysTimeOfSumltFac<- proc.time() - ptm
  ElapsedTimes[1] <- SysTimeOfSumltFac[3]

  #---- for-loop version ----
  SysTimeOfLoopFac <- system.time(Sum.LoopFactorial(N))
  #Store the elapsed time in a vector
  ElapsedTimes[2] <- SysTimeOfLoopFac[3]

  #---- recursive version ----
  SysTimeOfRecFac <-system.time(Sum.RecrusionFactorial(N))
  ElapsedTimes[3] <- SysTimeOfRecFac[3]

  #----Function calls ----
  FactorialSums <- 0
  FactorialSums[1:3]

  FactorialSums[1] <- Sum.LFacorial(N)
```

```
 FactorialSums[2] <- Sum.LoopFactorial(N)
 FactorialSums[3] <- Sum.RecrusionFactorial(N)

 Names <- c("LFacorial", "Loop", "Recrusion")

 DataToBePlot <- data.frame(N=N,Names,ElapsedTimes, FactorialSums)
 require(ggplot2)

 qplot(ElapsedTimes, Names, data = DataToBePlot) + geom_point(color="red")  +
facet_wrap(~ElapsedTimes, nrow=3) + ggtitle("Elasped time by Names")

}
```

#What are the growth rate of the three implementations as N increases ?
-> when the value of N is from 0 to 10, the elapsed time to calculate log factorial using lfactorial was .0001 secs, which is longer than the other two. The loop and recrusion performed it in 0.000 second. As the value of N increases, recrusion takes the most amount of time, and least time is taken by lfactorial. In my oberservation, when the N is 3000, the recrusion takes 12.899 seconds, the loop takes 2.612 seconds, and lfactorial completed it in only 0.006 seconds. In this sitation, caluclating a log factorial using recrusion is expensive over the loop is because it uses more memory by filling up a stack.