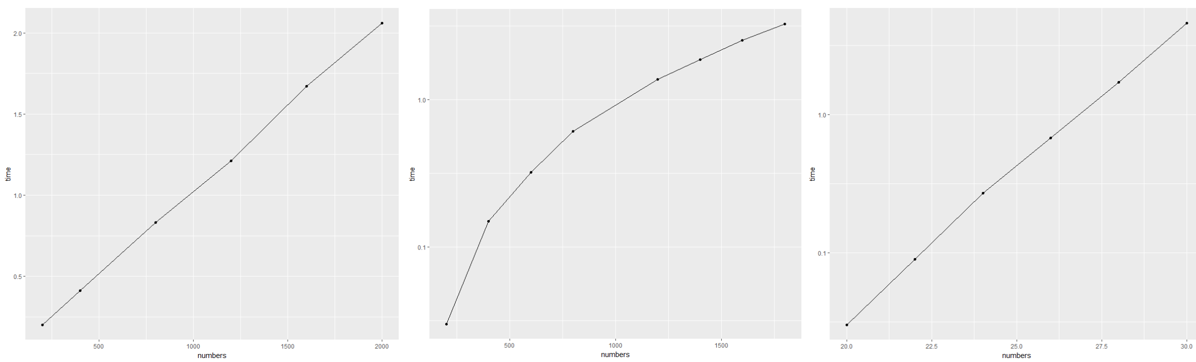
X. Sheldon Gu (xgu60)

log_factorial: **O(N).** Linear on normal scale plot. (Figure 1 left).

sum_log_factorial: **O(N$^{\alpha}$).** Non-linear on y-log10-scale plot (Figure 1 middle).

fibonacci: **O(α^N).** Linear on y-log10-scale plot (Figure 1 right).

Figure 1: Plots of running time versus numbers. **Left:** log_factorial, x, y normal scale. **Middle:** sum_log_factorial, x normal scale, y log10 scale. **Right:** Fibonacci, x normal scale, y log10 scale.



- To plot Figure 1 left, I added a constant time in log_factorial function. “return (constant.time() + log(n) + log_factorial(n - 1))”.

Figure 2: Snip of my code.

```r
  return (sum)
}

fibonacci <- function(n) {
  # Return nth Fibonacci number
  if (n <= 1)
    return (n)
  return (fibonacci(n - 1) + fibonacci(n - 2))
}

constant.time <- function(){
  count = 0
  for(i in seq(1:50)){
    for(j in seq(1:50)){
      count = count + 1
    }
  }
  return (0)
}

running.time <- function(function.name, num.seq){
  nums = c()
  time = c()
  for(n in num.seq){
    nums = append(nums, n)
    time = append(time, system.time(a <- function.name(n))[[1]])
  }
  time.dataframe = data.frame(numbers = nums, time = time)
  return (time.dataframe)
}

options(expressions=500000)
#rt = running.time(log_factorial, c( 200, 400, 800,1200,1600, 2000))
#rt = running.time(sum_log_factorial, c( 200, 400, 600, 800,1200, 1400, 1600,
rt = running.time(fibonacci, c(20, 22, 24, 26, 28, 30))

p <- ggplot(data = rt, aes(numbers, time)) +
  geom_point() +
  geom_line() +
  scale_y_log10()

print (p)
```