CSE 6242 Activity 2

Vincent La (Georgia Tech ID - vla6)

September 24, 2017

The time complexity of each function is

1. `log_factorial`: $O(n)$
    1. We can see this because in this implementation, for each value of n, the recursive call simply evaluates at n-1, continuing until n = 1.
2. `sum_log_factorial`: $O(n^2)$
    - We can see this because `log_factorial` is $O(n)$ and this implementation of `sum_log_factorial` calls `log_factorial` for each value of n.
3. `fibonacci`: $O(2^n)$
    - We can see this if we draw the recursion tree. When calling `fibonacci(n)`, both `fibonacci(n)` and `fibonacci(n-1)` are called once. However, `fibonacci(n-2)` is called twice. `fibonacci(n-3)` is called four times, etc.

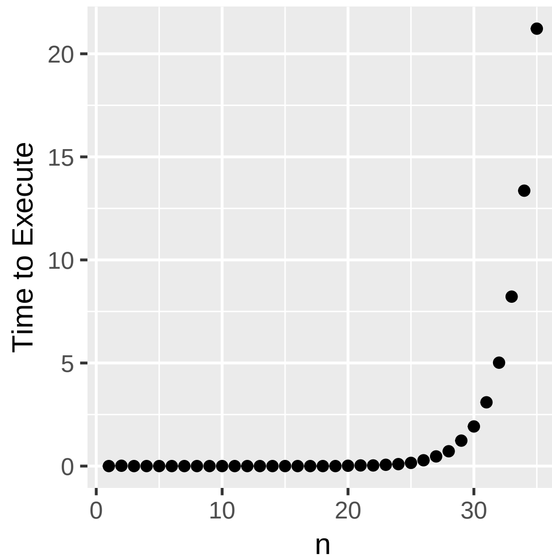
```r
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```r
# Plotting Fibonacci separately since run time takes the longest.
n = seq(1, 35, 1)
method = rep('fibonacci', length(n))
times = c()
for(i in n){
  times = c(times, system.time(fibonacci(i))[1])
}

df = data.frame(index=n, times=times, method=method)

ggplot(df, aes(index, times)) +
  geom_point() +
  xlab('n') +
  ylab('Time to Execute')
```

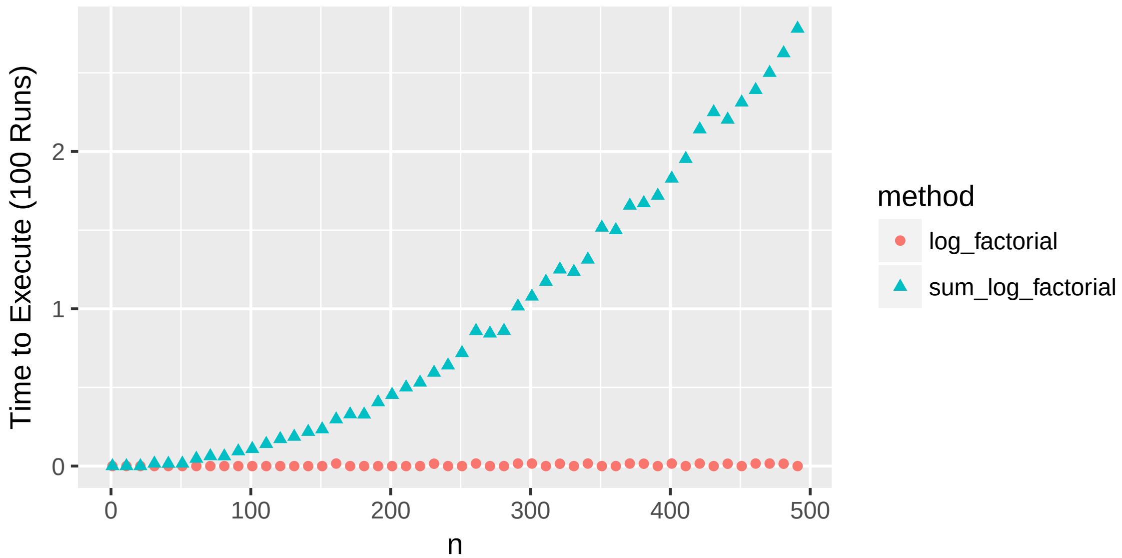



```r
# Testing
n = seq(1, 500, 10)
method = c(rep('log_factorial', length(n)), rep('sum_log_factorial', length(n)))
times = c()
for(i in n){
  times = c(times, system.time(replicate(25, log_factorial(i)))[1])
}

for(i in n){
  times = c(times, system.time(replicate(25, sum_log_factorial(i)))[1])
}

n = c(n, n)
df = data.frame(index=n, times=times, method=method)

ggplot(df, aes(index, times)) +
  geom_point(aes(color=method, shape=method, group=method)) +
  xlab('n') +
  ylab('Time to Execute (100 Runs)')
```


Time to Execute (100 Runs)
0
1
2
0
100
200
300
400
500
n
method
log_factorial
sum_log_factorial