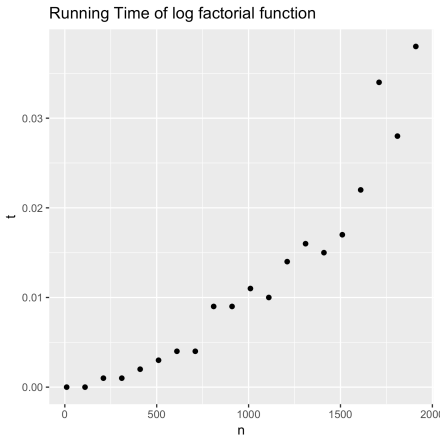

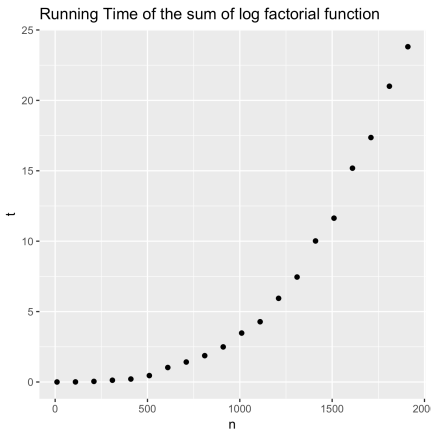

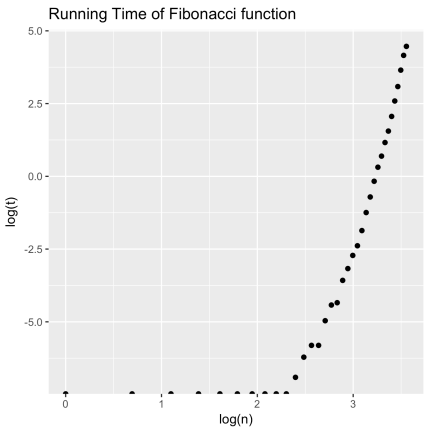

Log Factorial

Theoritical proof

In order to determine the growth rate of log factorial function, let's compute it :

$$\log(n!) = \log(n) + \log(n-1) + \log(n-2) + \cdots + \log(1) \leq nlog(n)$$

So the computationnal cost is 0(nlog(n)).

More points would be required to determine if the time complexity is 0(nlog(n)).

Sum of Log Factorial

Theoritical proof

In order to determine the growth rate of the sum of the log factorial function, let's compute it :

$\sum_{k=0}^{n} \log(n!) = \log(n!) + \log((n-1)!) + \log((n-2)!) + \cdots + \log(1!) \leq \log(n!) + \cdots + \log(n!)$

Hence the computational cost is 0(nlog(n !)) which is equal by using the previous expression to $0(n^2 \log(n))$.

Fibonacci

Theoritical proof

The sequence is models as F(n) = F(n-1) + F(n-2) thus the time function to calculate F(n) is the sum of the time to compute F(n-1) plus the one for F(n-2) plus the time to add them 0(1).

This scheme can be represented by a recursion tree with a depth of n, at each node there is two leafs hence we can induct that the time complexity is $0(2^n)$.

The log-log scale allows us to visualize a time complexity that could look like to $0(2^n)$ but as it not a straight line we could rather think it is an exponential cost : $0(e^n)$.

```
options(expressions = 50000) #Increase the number of nested recursions allowed
arr1 = c() # array for the log factorial
arr2 = c() # array for the sum of the log factorial
arr3 = c() # array for the fibonacci function

N1 = 2000 # Number of iterations for loop and R built in methods
a = seq(10,N1,100)
for (max in a){
  time_log_factorial = function(max){
    v1 = system.time(for (e in seq(1,max,100)) log_factorial(e))
    return(v1[1]) # access to user time
  }
  time_sum_log_factorial = function(max){
    v3 = system.time(for (e in seq(1,max,100)) sum_log_factorial(e))
    return(v3[1]) # access to user time
  }
  arr1 = c(arr1,time_log_factorial(max))
  arr3 = c(arr3, time_sum_log_factorial(max))
}
N2 = 35  # Number of iterations for Fibonacci function
a2 = seq(1,N2,1)
for (max in a2){
  time_fibonacci = function(max){
    v2 = system.time(for (e in seq(1,max)) fibonacci(e))
    return(v2[1]) # access to user time
  }
  arr2 = c(arr2,time_fibonacci(max))
}
```

```
d2 = data.frame(x = a2 ,y = arr2)

qplot(x = a2,
      y = arr2,
      data = d2,
      main = "Running Time of Fibonacci function",
      geom = "point",
      xlab = 'n',
      ylab = 't')
ggsave('plot_fibonacci.png',width = 5,height = 5)

qplot(x = log(a2),
      y = log(arr2),
      data = d2,
      main = "Running Time of Fibonacci function",
      geom = "point",
      xlab = 'log(n)',
      ylab = 'log(t)')
ggsave('plot_fibonacci_log.png',width = 5,height = 5)
```