Neel Parikh npari3@uic.edu Homework 3 CS 341

```
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ g++ -g main1.cpp -o main1
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ time ./main1

real    0m0.009s
user    0m0.000s
sys     0m0.016s
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ ghc ./hsknatural.hs
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ time ./hsknatural

real    0m0.013s
user    0m0.000s
sys     0m0.000s
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ ghc ./hskaccumlator.hs
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ time ./hskaccumlator

real    0m0.012s
user    0m0.000s
sys     0m0.016s
```

```
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ ghci hskaccumlator.hs
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( hskaccumlator.hs, interpreted )
Ok, one module loaded.
*Main> :set +s
*Main> main
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
(0.02 secs, 229,792 bytes)
*Main>
Leaving GHCi.
neelparikh0830@Neel-PC:/mnt/c/Users/neelp/Desktop/infint$ ghci hsknatural.hs
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( hsknatural.hs, interpreted )
Ok, one module loaded.
*Main> :set +s
*Main> main
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
(0.06 secs, 227,256 bytes)
*Main>
Leaving GHCi.
```

Program	Time in Seconds	Time in Milliseconds
C++	0.009s	9ms
Haskell naturally	0.013s	13ms
Haskell accumulator	0.012s	12ms

Haskell naturally inter.	0.06s	60ms
Haskell accumulator inter.	0.02s	20ms

For both accumulator and naturally an interpreter takes very less time to analyze the source code but however, the overall time for runtime or execution the process is much slower. The compiler takes a lot of time to analyze the source code but the overall time for the runtime or execution process is much faster. But, When you look at accumulator and the naturally occurring function we can say that accumulator is way more faster than the naturally function because in that function doesn't have to create 100 stacks for the function due to it having in a single loop and naturally it might have to create 100 stacks because it's repeatedly calling itself and that might create new stack everytime. In imperative languages, the speed is related to how smart the programmer is thinking and coding in different way and how he is trying to optimize the code and runtime in c/c++ and you will get the fastest output or runtime.