1. What was most challenging about this deliverable?

There were two difficulties that took us a long time to solve, which were validating the time stamps and the hash of a block. Our original plan for validating the time stamps was to use the “at” method of the “Time” class, to convert time stamps in the float data type into “Time” objects, and then use operands to compare two time stamps. However, the plan didn’t work out because there seemed to be some slight trailing digit misses. For example, a time stamp of “1518893687.329767000” turned out to be “1518893687.329766988”, which caused errors when determing the correct order of blocks. We had to give up the idea of using the “Time” class to compare time stamps, and compare them in a rather naive way, which is to split the seconds and nanoseconds, then compare the two parts separately. The other difficulty was to validate the hashes of blocks, which technically shouldn’t be hard, but there was an issue with the newline characters and the extra spaces inside blocks, which caused the computed hash to be different from the given hash at the end of each block. If we really need to choose one, then the most challenging one would be the issue with comparing time stamps, the problem with validating hashes took us a long time to figure out, but a simple “strip” method would do the trick.

2. What kind of edge cases and failure modes did you consider?

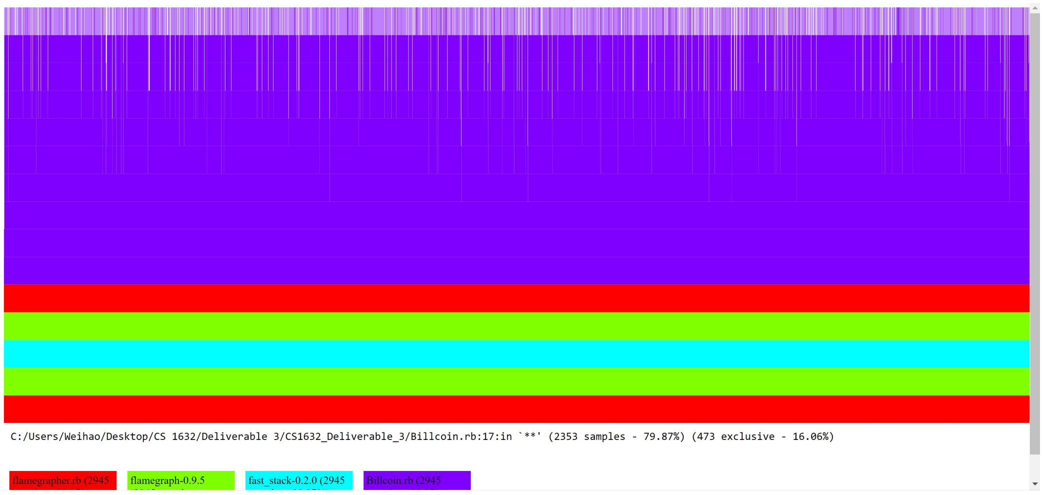
Whether the file from the given path exists or not, whether a block only has 5 elements separated by “|” character, whether the addresses are at least one character and less than six characters in length, and whether the addresses are alphabetic. The rest are the requirements specified by the deliverable documentations.

3. Using the flame graph, what methods were taking up the most CPU time?

The method “get_hash” took the most CPU time, since it had to use the “**” power operand.

4. Did you make any changes based on the flame graph or timing?

We didn’t, because the method that’s taking the most CPU time was the “get_hash” method, which has to compute the hash of a string according to a math formula, which cannot be reduced into a simpler form (based on Wolfram Alpha), so we can only leave it there.



```
C:/Users/Weihao/Desktop/CS 1632/Deliverable 3/CS1632_Deliverable_3/Billcoin.rb:17:in `**' (2353 samples - 79.87%) (473 exclusive - 16.06%)
```

Execution time for long.txt:

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 23
Milliseconds      : 473
Ticks             : 234734578
TotalDays         : 0.000271683539351852
TotalHours        : 0.00652040494444444
TotalMinutes      : 0.391224296666667
TotalSeconds      : 23.4734578
TotalMilliseconds : 23473.4578

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 23
Milliseconds      : 669
Ticks             : 236692370
TotalDays         : 0.000273949502314815
TotalHours        : 0.00657478805555556
TotalMinutes      : 0.394487283333333
TotalSeconds      : 23.669237
TotalMilliseconds : 23669.237

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 23
Milliseconds      : 699
Ticks             : 236993130
TotalDays         : 0.000274297604166667
TotalHours        : 0.0065831425
TotalMinutes      : 0.39498855
TotalSeconds      : 23.699313
TotalMilliseconds : 23699.313
```

MEAN (in miliseconds): 23614.002599999996

MEDIAN (in miliseconds): 23669.237