Ricardo Deodutt
3Fork

There will be 2 codes attached to this project. In the first code attached, there is a revised version of my “rng2file.c” program. In my previous project (TwoSum Project), I had a program that generated numbers between -1000 to 1000. This program would print out a list of randomly generated numbers into a DAT file. On the first version, if I wanted to generate a different amount of numbers I would have to nano into the file and alter the line that says “#define LIMIT 1000”. This was not a tedious task to do but, eventually I found it very annoying because I had to compile the program every time I changed a number. My revised version which is called “ rng2file v2” now ask the user for how many numbers they would like to randomly generate. As you can see in figure 2, the program first ask the user "How many numbers do you want to generate?". The user would then enter any number they would like. The program will then output all the numbers into the shell and say the numbers were successfully generated in the range of -1000 to 1000. I created this code separately from the main code because it can be revised in another project as needed. This would also make the main program more efficient. For the main code, I could have done the same process and ask the user for a certain amount of numbers, but I felt this would alter the execution time. I felt more comfortable changing the value in the code and compiling it each time.

The other code attached is the main code called ”3ForkProject”. The reason why I call this project “3ForkProject” is because we are adding 3 numbers together and using the fork command. In this project the main objective was to add three numbers together and use the fork command. While doing these objectives, I made the outputs efficient and appealing to the

user. In the first project, I printed out all the computations the program did into the shell. For the new program, I made the program print the outputs to a file. This was necessary because if the code is printing out all the computations onto the user's shell, it would not be considered a fork command. When a fork command is being used, the user should be able to use the shell while the program runs in the background. In figure 13 below, I ran the program to compute 10,000 numbers. The program first finished the parent program. Once it finished, I was able to use the shell and type "ps" to check how long the child process was taking. If you take a look at the "3ForkProject.c" program attached, in line 33-40, it is very similar to line 23-30. While writing the outputs to a file, I faced some difficulties with larger quantities of numbers. The main problem was the output files would become extremely large. To fix this mistake, I added a limit to how many numbers would be outputted to the result file. In line 104-115 there is an if statement that finds inase the computation of 3 numbers is equal to the predetermined number. If that number is found, the program will increment the variable "w". If the variable "w" is lower than 101, then the program will print out the 3 numbers being added and say the predetermined sum has been found. You can see this in figure 15. Once the variable "w" has been incremented more than 100 times, the program stops outputting numbers into the results file.

In the main code, the program starts by initializing a bunch of values. Once again in line 13, I set a value for the CLOCKS_PER_SEC because when I was testing the program, windows and linux had different CLOCKS_PER_SEC values. In line 16 the start clock is started and in line 17, the child process is created. Essentially the main parent program has two if statements. In line 19, if the pid is equal to 0, the child code is ran. If the pid is not equals to 0, then the

program will run line 151. In line 151 the program will say the fork creation has failed and will terminate. If the pid is equal to 0, the child process is created and it will run the code inside the brackets. The parent process will skip to line 157 and print out how long it took to execute. If you take a look at figure 16, this graph represents the parent and child execution time. In the graph the parent process (which is the the line with circles on each data point) is a constant 0. The parent process always finishes at a number very close to 0 seconds. This proves that the parent process does not run the main code which adds all the numbers. The child process (which is the line with a star at each data point) is a slow increase in execution time based off the amount of numbers tested. The reason why the child process is longer is because it runs the main code which adds all the numbers. In figure 17, this chart shows all testing I have done to create a graph of average execution time. I ran each program 3 times to get a accurate average. While running the program each time, I kept the numbers randomized. This chart shows what I expected, which was the program takes longer to execute based off how much numbers are being processed.

Once the child runs the code, it will start by assigning the pre-determined value to be found which is 0. The program then opens up both the read and write file. In lines 27-30 and 37-40, these commands are placed incase the program cannot open the file. As you can see in figure 1, this picture shows an example of what the program does when it cannot open the file with random numbers in it. The parent program accurately finishes and the child program prints out the error message of the missing file and terminated the program.

In line 48-51 the program reads all the numbers in the BatchOfRNG file. BatchOfRNG is the file with a bunch of random numbers created by the “rng2file v2” program. In figure 3, this

this illustration shows 10 numbers generated by the program and written to a file called "BatchOfRNG". I made the output file very user friendly and printed out to the user the amount of numbers generated, just incase they have forgotten. Once the program reads the file, it will stores the numbers into an array called "numberArray". Next, in line 54-60 the program prints out the first 100 numbers in the array into the result file. As you can see in figure 14, this result file shows 10 numbers generated by the child process. Afterwards, the main part of the program will be remaining. In line 71-124, the program does the computation of adding the 3 numbers together. The program will then print out the first 100 pre-determined sums found. As you can see in figure 15, this is an image of the ending part of the result.dat file. This illustration shows how the program will print out to a file when a pre-determined sum has been found. In figure 14, the image does not show the pre-determined values found because there was no pre-determined values found by the program. In the same figure, you can see in line 18 the program states "There is a total of 0 pre-determined sums met". While I was writing this program, I tried to make everything efficient and user friendly.

After line 124, the program prints out a small stat in the results.dat file. You can see this small stat at the bottom of figure 14 and 15. The program prints out how many numbers were processed and the amount of sums found. In figure 15, this illustration shows that the program ran 10000 random numbers and found "62328717 pre-deteminred sums". That number was accurate based off my graphs and chart in figure 18 and 19. In figure 18, I recorded how many sums where found at each number. This illustration goes together with figure 19. In figure 19, this chart was recorded by myself. I ran each program 3 times and took the average number of each. I expected these numbers to be large based off the amount of numbers tested. After all

the numbers are computed, the read and write files (myFileREAD and myFileWRITE) are closed in line 132-133. In lines 135-146 the child process ends its clock and prints out some statistics to the shell. You can see the statistics printed out to the shell in figure 4, 5, 6, 7, 8, 9, 10, 11, and 12.

Overall when testing the program, I found some similarities. Figure 4 and 5 were very similar. These figures tested number 10 and 100. These numbers so fast that the child process ended at 0 seconds. When I tested 500, 1000, 2000, 3000, 4000, and 5000 numbers in figure 6, 7, 8, 9 10, and 11, I noticed most of the pre-determined sum doubled and the execution time doubled. I only noticed a major difference up into figure 12. In figure 12, I started to test a large amount of numbers. In figure 12. the program ran 10,000 numbers, found about 62 million pre-determined sums, and took me about 437 seconds to execute. I recorded all of these statistics in figure 17 and 19.

Overall this project was fun to do. I experimented a lot with the Linux operating system. Usually I would do any project on my windows device. I learned new commands on Linux. I faced some difficulties creating the program but I overcame them. Personally, I think my program will appeal to users more since I put more effort into that end.

Test Cases

Test case where the main program cannot open the file containing random numbers.

```
[rdeodutt@cpp ~]$ ./ThreeFOrk

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$ Could not open file with batch of random numbers.

[rdeodutt@cpp ~]$
```

Figure 1

Sample of how "rng2file v.2.c" works. This is how 10 numbers randomly generated looks like.

```
[rdeodutt@cpp ~]$ ./RNG
How many numbers do you want to generate? 10

Generating 10 random numbers....

267
331
444
936
197
101
-510
608
-904
-268

10 random numbers were successfully generated in [-1000 to 1000]
```

Figure 2

Sample of how the file "rng2file v.2.c" output file looks like. This file is named "BatchOfRNG.dat".

```
 GNU nano 2.0.9                                File: BatchOfRNG.dat

267
331
444
936
197
101
-510
608
-904
-268

There is a total number of 10 randomly generated numbers.
```

Figure 3

10 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
-------------------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 10.
Overall, there is a total of 0 pre-determined sums met.
The execution time of the child process was: 0.000000000000000 seconds.
-------------------------------------------------------------------------------------------------------------------

[rdeodutt@cpp ~]$
```

Figure 4

100 Numbers

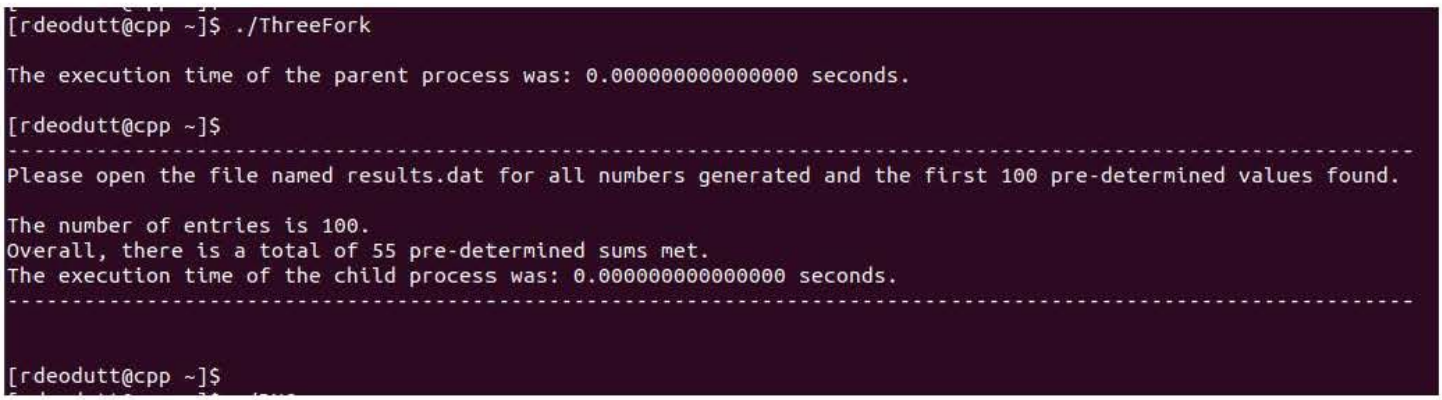

Figure 5

500 Numbers

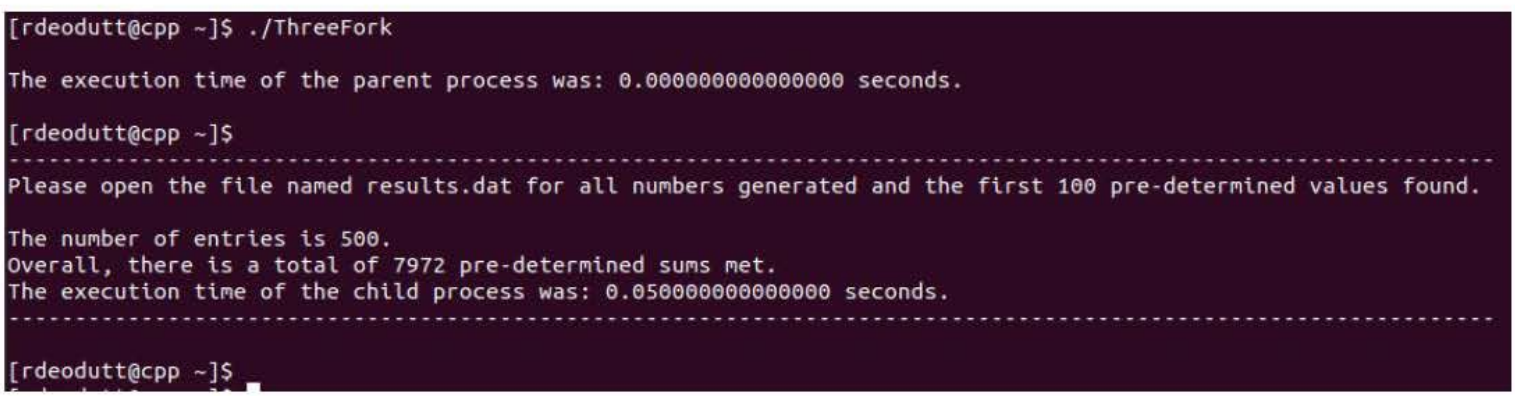

Figure 6

1,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
--------------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 1000.
Overall, there is a total of 61284 pre-determined sums met.
The execution time of the child process was: 0.440000000000000 seconds.
--------------------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 7

2,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
--------------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 2000.
Overall, there is a total of 505143 pre-determined sums met.
The execution time of the child process was: 3.570000000000000 seconds.
--------------------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 8

3,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
-----------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 3000.
Overall, there is a total of 1674898 pre-determined sums met.
The execution time of the child process was: 12.150000000000000 seconds.
-----------------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 9

4,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
-----------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 4000.
Overall, there is a total of 3997688 pre-determined sums met.
The execution time of the child process was: 28.620000000000001 seconds.
-----------------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 10

5,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
-----------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 5000.
Overall, there is a total of 7843987 pre-determined sums met.
The execution time of the child process was: 55.820000000000000 seconds.
-----------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 11

10,000 Numbers

```
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$
-----------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 10000.
Overall, there is a total of 62399611 pre-determined sums met.
The execution time of the child process was: 431.699999999999989 seconds.
-----------------------------------------------------------------------------------------------------


[rdeodutt@cpp ~]$
```

Figure 12

Checking "ps" of a program running in the background..

```
10000 random numbers were successfully generated in [-1000 to 1000]
[rdeodutt@cpp ~]$ ./ThreeFork

The execution time of the parent process was: 0.000000000000000 seconds.

[rdeodutt@cpp ~]$ ps
  PID TTY          TIME CMD
 7316 pts/2    00:00:00 bash
 7743 pts/2    00:02:26 ThreeFork
 7791 pts/2    00:00:00 ps
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$ ps
  PID TTY          TIME CMD
 7316 pts/2    00:00:00 bash
 7743 pts/2    00:02:30 ThreeFork
 7792 pts/2    00:00:00 ps
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$ ps
  PID TTY          TIME CMD
 7316 pts/2    00:00:00 bash
 7743 pts/2    00:04:07 ThreeFork
 7822 pts/2    00:00:00 ps
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$ ps
  PID TTY          TIME CMD
 7316 pts/2    00:00:00 bash
 7743 pts/2    00:06:17 ThreeFork
 7867 pts/2    00:00:00 ps
[rdeodutt@cpp ~]$
[rdeodutt@cpp ~]$
-----------------------------------------------------------------------------------------------------------
Please open the file named results.dat for all numbers generated and the first 100 pre-determined values found.

The number of entries is 10000.
Overall, there is a total of 62821452 pre-determined sums met.
The execution time of the child process was: 447.589999999999975 seconds.
-----------------------------------------------------------------------------------------------------------

[rdeodutt@cpp ~]$
```

Figure 13

This is how the Result.dat file looks like. If 10 numbers are processed.

results.dat

```
These are the first random numbers generated. (Up to 100)
547
263
-491
-516
457
648
472
380
459
-430


These are the first pre-determined values found. (Up to 100)


There is a total of 10 randomly generated numbers.
There is a total of 0 pre-determined sums met.
```

Figure 14

This is the ending of the result.dat file. 10,000 numbers were processed.

```
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.
898 + -406 + -492 = 0  ---------> The sum has met a pre-determined value.


There is a total of 10000 randomly generated numbers.
There is a total of 62328717 pre-determined sums met.
```

Figure 15

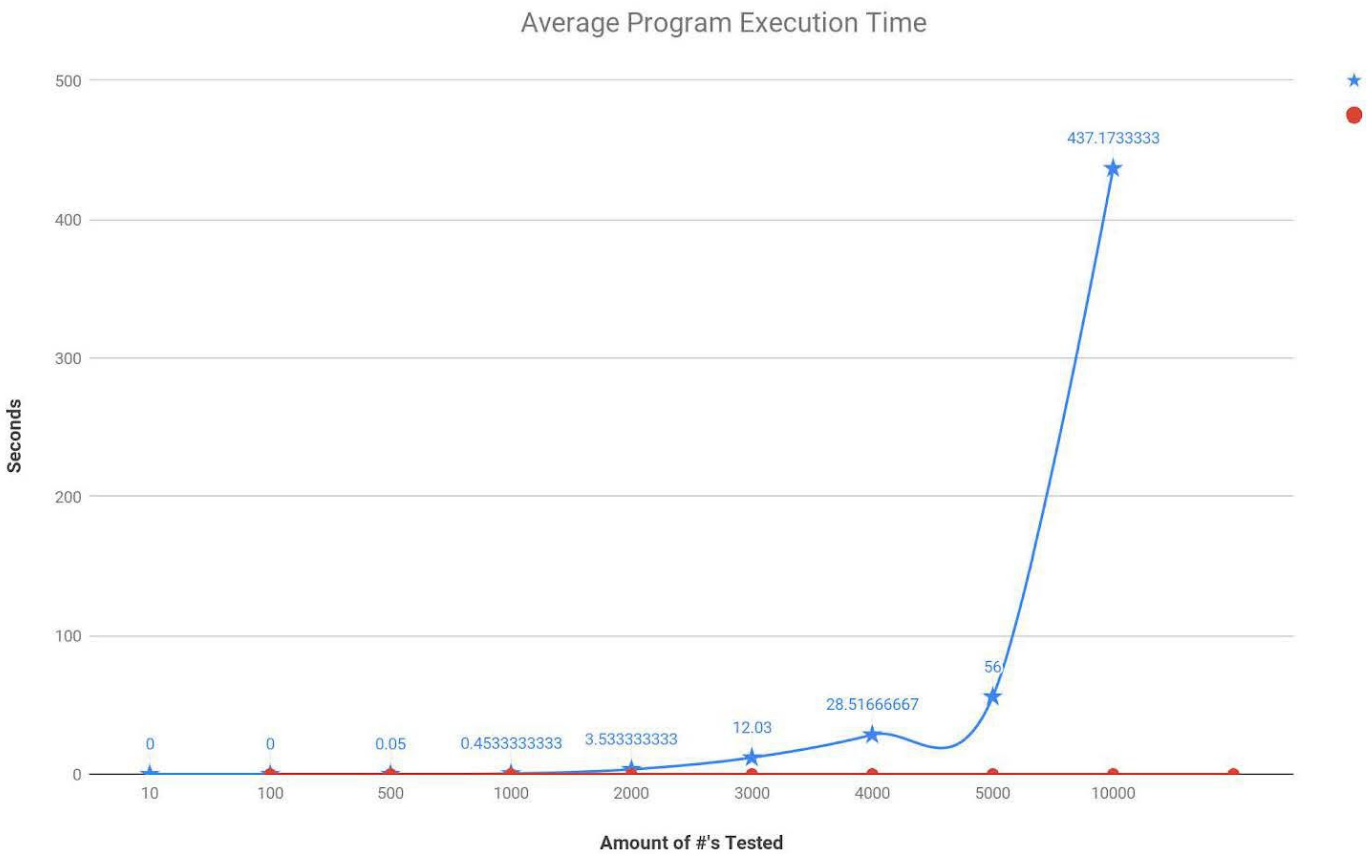

Figure 16

Raw Data from the graph in Figure 16

Execution Time(seconds)								
Amount of #s	1st Trial Parent	1st Trial Child	2nd trial Parent	2nd trial Child	3rd trial Parent	3rd trial Child	Average Parent	Average Child
10	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0
500	0	0.05	0	0.05	0	0.05	0	0.05
1000	0	0.44	0	0.48	0	0.44	0	0.4533333333
2000	0	3.57	0	3.5	0	3.53	0	3.533333333
3000	0	12.15	0	11.96	0	11.98	0	12.03
4000	0	28.62	0	28.58	0	28.35	0	28.51666667
5000	0	55.82	0	56.55	0	55.63	0	56
10000	0	447.59	0	431.7	0	432.23	0	437.1733333

Figure 17

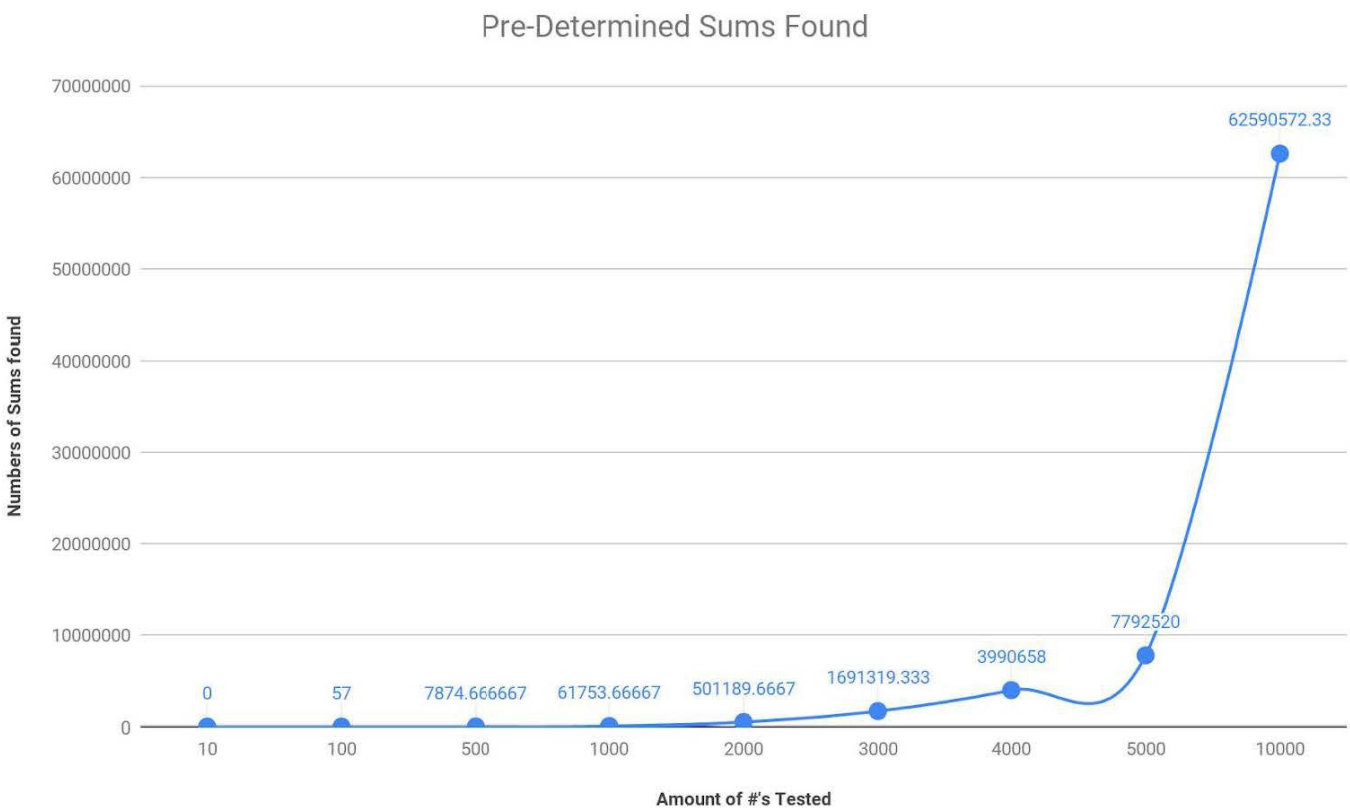

Figure 18

Raw Data from the graph in Figure 18

Sums Found					
Amount of #s		1st Trail	2nd Trail	3rd Trail	Average
10		0	0	0	0
100		58	55	58	57
500		8026	7626	7972	7874.666667
1000		61284	61039	62938	61753.66667
2000		505143	501887	496539	501189.6667
3000		1674898	1700493	1698567	1691319.333
4000		3997688	3969974	4004312	3990658
5000		7843987	7759399	7774174	7792520
10000		62821452	62399611	62550654	62590572.33

Figure 19