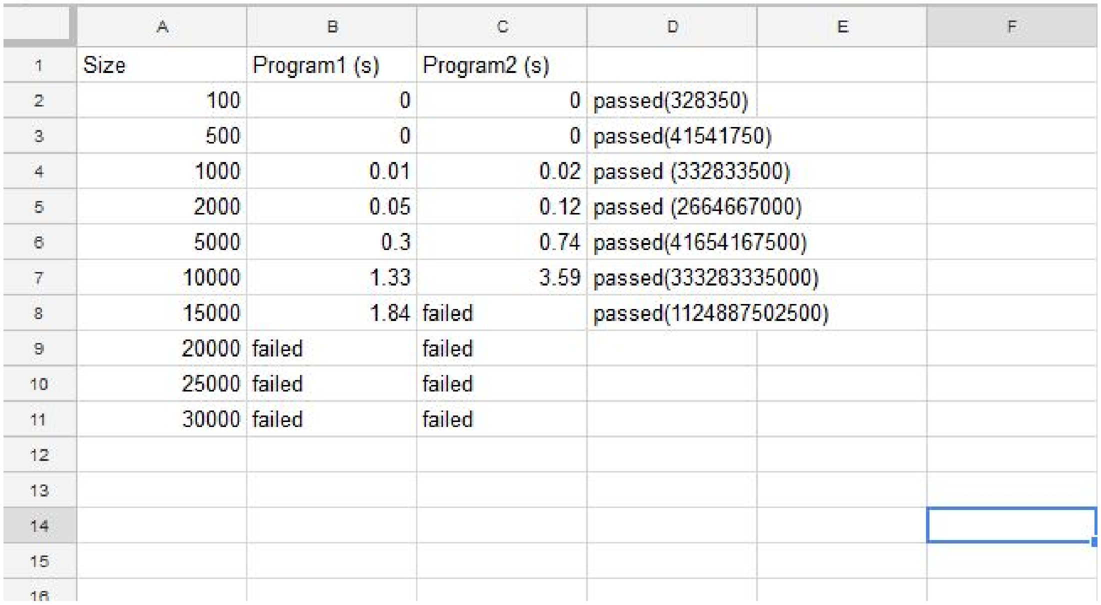
Name: Adrian Tran
SID: 861233198

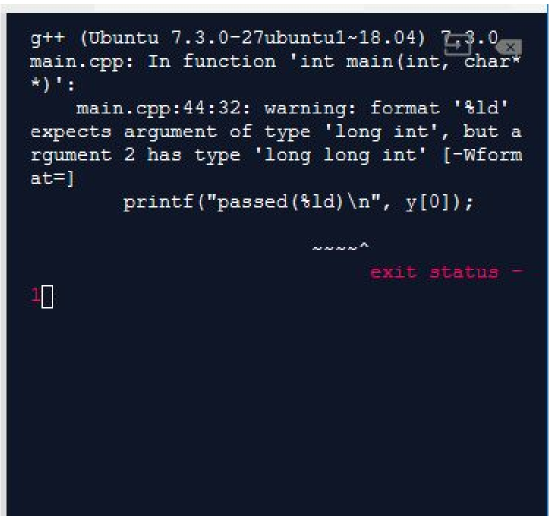
CS161L Lab03 Case Study

Data (runtime):



	A	B	C	D	E	F
1	Size	Program1 (s)	Program2 (s)			
2	100	0	0	passed(328350)		
3	500	0	0	passed(41541750)		
4	1000	0.01	0.02	passed (332833500)		
5	2000	0.05	0.12	passed (2664667000)		
6	5000	0.3	0.74	passed(41654167500)		
7	10000	1.33	3.59	passed(333283335000)		
8	15000	1.84	failed	passed(1124887502500)		
9	20000	failed	failed			
10	25000	failed	failed			
11	30000	failed	failed			
12						
13						
14						
15						
16						

However, it is worth noting that the failed entries did not actually output a "failed" message when ran in the IDE. They just exited, presumably because the code was taking too long. I tried using multiple IDE's to make sure the code runtime length was the cause of the problem. Most of the IDE's outputted an error message similar to this one:

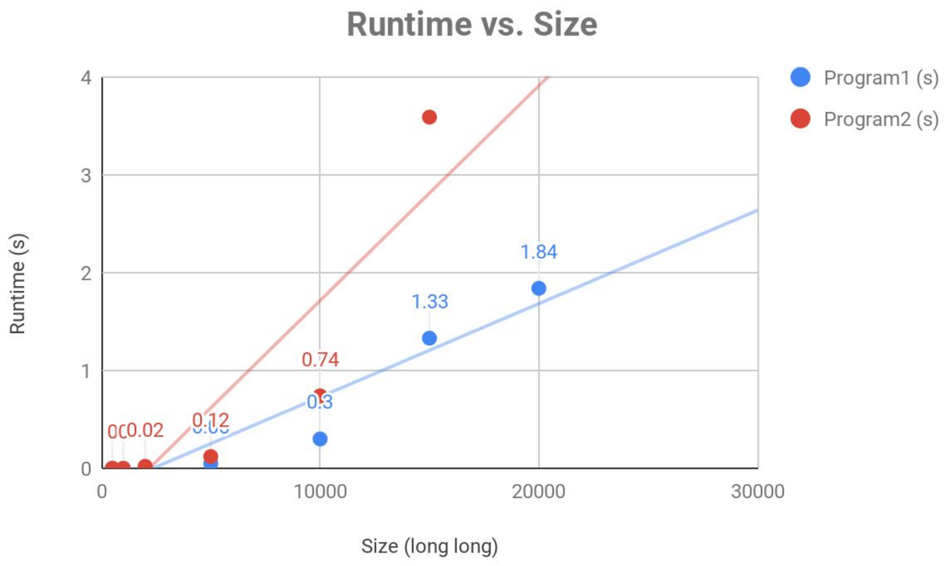


```
g++ (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
main.cpp: In function 'int main(int, char*
*)':
    main.cpp:44:32: warning: format '%ld'
expects argument of type 'long int', but a
rgument 2 has type 'long long int' [-Wform
at=]
        printf("passed(%ld)\n", y[0]);

                        ~~~~^
                              exit status -
1
```

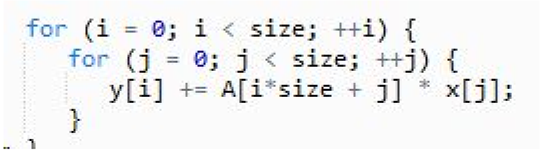
Where the program exited by itself without outputting or displaying a seg fault. One or two of the IDE's I used displayed a timeout error.

Graph of the data:



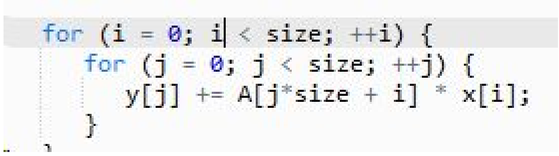
As we can see, program 2 has a much slower runtime than program 1, by a factor of approximately 2. I believe that the only difference in the code of the programs is in the matrix_vector_multiply function.

1) In program 1, we have the following main block of code:



```
for (i = 0; i < size; ++i) {
   for (j = 0; j < size; ++j) {
      y[i] += A[i*size + j] * x[j];
   }
```

2) In program 2, we have the following main block of code:



```
for (i = 0; i < size; ++i) {
   for (j = 0; j < size; ++j) {
      y[j] += A[j*size + i] * x[i];
   }
```

We can see that in program 2, we're working with a greater number of assignments. Namely, in Program 1, we jump to y[i] and assign "size" times. However, in Program 2, because we're using j as the index, we need to jump to y[j] and assign "size * size" times. Clearly, we can see that Program 2 has a larger CPI, which means that it would operate at a slower runtime, which is exactly what is shown in the graph.