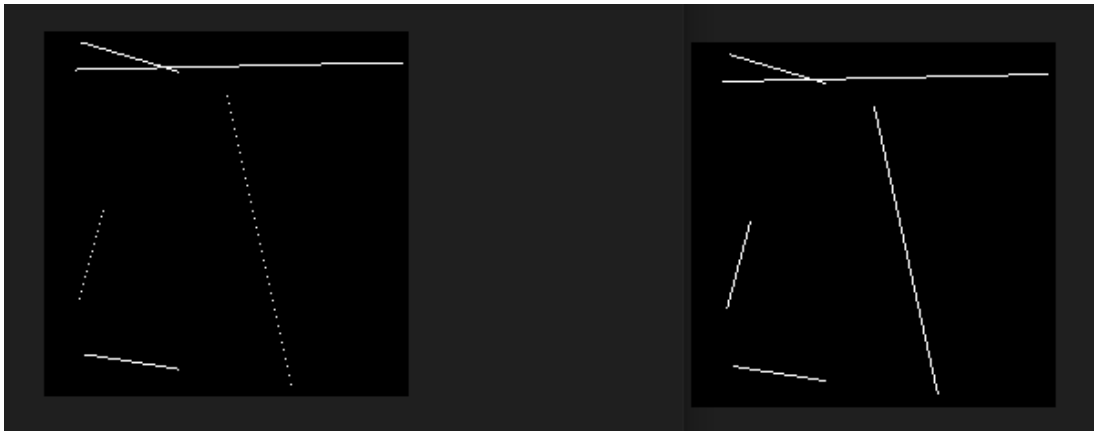
Program 1 and Program 2: Basic Line Drawing Algorithm and Bresenham Line Drawing Algorithm



Basic Drawing vs. Bresenham Drawing Algorithm

To determine the effectiveness of the line drawing algorithms, the first thing that needs to be tested is the speed at drawing random lines. This was done by only running the algorithm and not putting pixels on the screen. In the table below, we have the Algorithm on the left hand side, and the number of lines (N) drawn. The time for each is listed in the box. These times were averaged from multiple runs.

Random Line Drawing

	1 Line	10 Lines	100 Lines	1000 Lines	10000 Lines	100000 Lines
Basic	$4.08 * 10^{-5}$ s	$1.31 * 10^{-4}$ s	$1.05 * 10^{-3}$ s	$1.24 * 10^{-2}$ s	$1.09 * 10^{-1}$ s	1.07 s
Bresenham	$3.51 * 10^{-5}$ s	$1.65 * 10^{-4}$ s	$1.34 * 10^{-3}$ s	$1.56 * 10^{-2}$ s	$1.44 * 10^{-1}$ s	1.42 s

It is notable that the amount of time it took to draw was linear and that it had about the same amount of time to execute per line.

```
How Many Lines? (Integers only):10
Time total_basic: 0.00010470000000029067
Time total brz: 0.00013629999999975606
```

```
How Many Lines? (Integers only):10
Time total_basic: 9.769999999997836e-05
Time total brz: 0.00014980000000042182
```

```
How Many Lines? (Integers only):100
Time total_basic: 0.0010322000000002607
Time total brz: 0.001307799999999748
```

```
How Many Lines? (Integers only):100
Time total_basic: 0.0011366000000019305
Time total brz: 0.0013744999999980578
```

However, it was interesting to see that drawing one line often had issues of consistent time. This might be because the computations were drastically different depending on which direction, how long, or other variation. It seems that increasing amounts of lines were able to average out the wide variety of time cost to create a relatively linear line.

```
How Many Lines? (Integers only):1
Time total_basic: 1.0900000000035881e-05
Time total brz: 2.049999999997887e-05
```

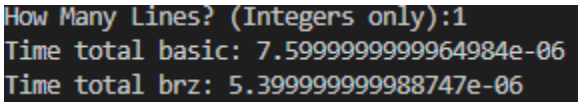
```
How Many Lines? (Integers only):1
Time total_basic: 0.00010380000000509426
Time total brz: 8.749999999935199e-05
```

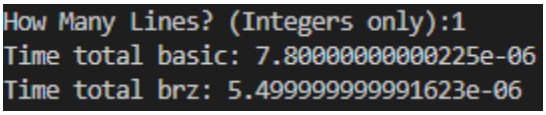
It was also discovered that the Bresenham algorithm for the random drawing was actually slower than the basic. This is likely because the bresenham algorithm implementation was not optimal and led to a longer run time. Another issue might have been the programming language. Because Python does not directly interact with the operating system, the cost saving of bit shifting is not as great because of the overhead cost. This means that the increase of operations outweighs the savings from bitshifting.

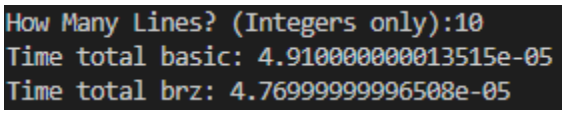
Short Line Drawing

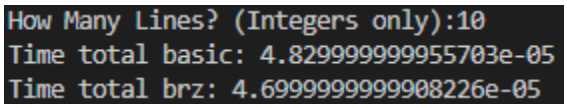
	1 Line	10 Lines	100 Lines	1000 Lines	10000 Lines	100000 Lines
Basic	$7.7 * 10^{-6}$ s	$4.85 * 10^{-5}$ s	$4.56 * 10^{-4}$ s	$4.61 * 10^{-3}$ s	$4.45 * 10^{-2}$ s	0.43 s
Bresenham	$5.4 * 10^{-6}$ s	$4.72 * 10^{-5}$ s	$4.49 * 10^{-4}$ s	$4.60* 10^{-3}$ s	$4.60 * 10^{-2}$ s	0.44 s

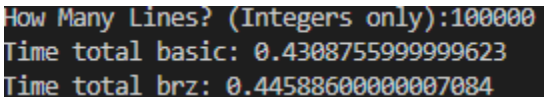
This test was done by generating the same line over and over again. It was a ‘short’ line, going from (0,0) to (25,20).







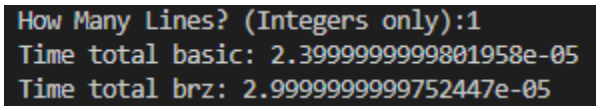


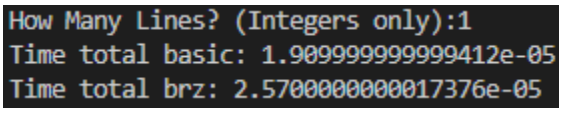


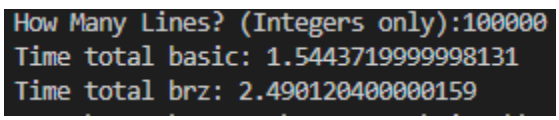
Long Line Drawing

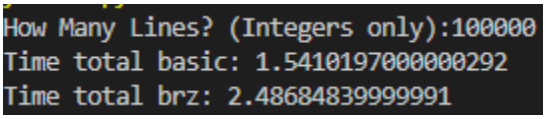
	1 Line	10 Lines	100 Lines	1000 Lines	10000 Lines	100000 Lines
Basic	$1.91 * 10^{-5}$ s	$1.68 * 10^{-4}$ s	$1.62 * 10^{-3}$ s	$1.73 * 10^{-2}$ s	$1.58 * 10^{-1}$ s	1.544 s
Bresenham	$2.65 * 10^{-5}$ s	$2.52 * 10^{-4}$ s	$2.55 * 10^{-3}$ s	$2.75 * 10^{-2}$ s	$2.52 * 10^{-1}$ s	2.48 s

This test was done by generating the same line over and over again. It was a ‘long’ line, going from (0,0) to (100,20).







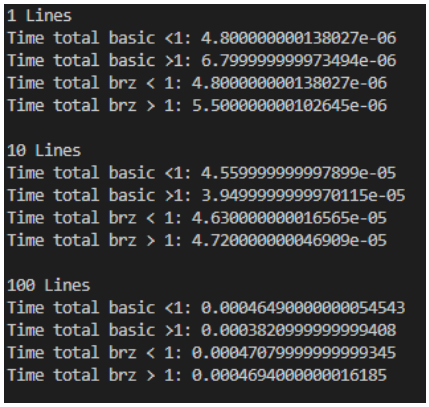


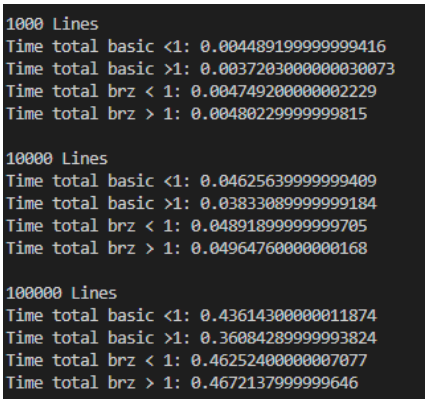
For the short drawing, it is noticeable that in the first 1000 or so lines, the bresenham was faster than the basic algorithm. Additionally, the basic draw seems to have a wider range of draw times than the Bresenham. However, the bresenham is slower once the line drawing count starts going up, around 10000 lines. The time between them is very close at 100,000 lines though.
For the long drawing, the bresenham is consistently slower. This likely means that the more computations or more loops causes the bresenham to be slower, while less computations or less loops causes the basic to be slower.

Different Slopes Times

	1 Line	10 Lines	100 Lines	1000 Lines	10000 Lines	100000 Lines
Basic - Less than 1	$4.85 * 10^{-6}$ s	$4.63 * 10^{-5}$ s	$4.61 * 10^{-4}$ s	$4.45 * 10^{-3}$ s	$4.63 * 10^{-2}$ s	0.436 s
Basic - Greater than 1	$6.8 * 10^{-6}$ s	$3.95 * 10^{-5}$ s	$3.73 * 10^{-4}$ s	$3.70 * 10^{-3}$ s	$4.59 * 10^{-2}$ s	0.390 s
Bresenham - Less than 1	$4.75 * 10^{-6}$ s	$4.63 * 10^{-5}$ s	$4.59 * 10^{-4}$ s	$4.76 * 10^{-3}$ s	$4.85 * 10^{-2}$ s	0.463 s
Bresenham - Greater than 1	$5.78 * 10^{-6}$ s	$4.82 * 10^{-5}$ s	$4.58 * 10^{-4}$ s	$4.79 * 10^{-3}$ s	$4.83 * 10^{-2}$ s	0.465 s

This is used to test out the different times between two different slopes, one of which is less than 1 and one greater than 1. Below is a sample run. It seems that the basic had a significant difference when slopes were greater than one, while the Bresenham started out with a bigger difference but closed the gap over time from more samples. It seems that the basic runs faster when the slopes are greater than one.





Conclusion

While the Bresenham line algorithm was supposed to be more time efficient in its computations, a combination of sub optimal implementation and potential python overhead caused the algorithm to only be faster than the basic algorithm when using low count, shorter lines. With the introduction of longer lines, both algorithms were slower at implementing them than smaller lines at all times. This is caused by a proportionately larger number of computations across a longer range of values. The same goes for the amount of lines; with an increasing amount of lines, the time it takes goes proportionally up. However, the relationship is close to linear, meaning for 1 additional line drawn, the time goes up the same each time. Lastly, the slopes seem to have a slight impact on the speed of the algorithms. Because basic always calculates the y, regardless of the size of the slope, this might have had a bigger difference. Bresenham switches to calculating y if the slope is greater than 1, x if less and shows very similar performance.