Algorithm 1

Description

Algorithm one follows a generally greedy approach to finding all the bombs. We improved this approach by using statistical analysis which propagates forward as the algorithm builds intuition of the board. The algorithm follows this methodology: there are two initial boards. One board contains ‘label values’ which represent the number of bombs surrounding a spot on the board. The other board contains ‘probability values’ which represent how likely we believe that spot is to have a bomb. These probability values do not actually add to 1, rather they represent a system more akin to particle filtering. In the beginning, we visit the initial square, which is guaranteed to have the highest possible label value, and we update the probability values surrounding the initial square.

Whenever we uncover a square, there are 3 possibilities we can follow to update the probability values surrounding that square. First, if we uncover a square and that square’s value is 0, then there are no bombs surrounding that square. Therefore, we do not want our algorithm to check these squares which do not have bombs. Hence, we reduce the probability values of the squares surrounding the 0 square to -100. Second, consider whether we uncover a square that has a typical label value. In this case, we simply want to append this label value to the probability value of the squares surrounding the square we uncovered. We do this because if the label value of the square we uncovered is high, there is a higher chance that there is a bomb in the surrounding squares. Finally, if the uncovered square is a bomb, it is less likely that there are more bombs surrounding the uncovered square. Hence, we decrease the probability value of surrounding squares.

Our main algorithm loop continues until all the bombs have been found. Within this loop, we consider the current state of our probability table and find the positions which have the highest probability value (most likely to have a bomb). If we have several positions that are tied, we determine which position is the closest to the center and mine that square. We choose to mine the square closest to the center because mines closer to the center have capability to uncover more information. When we mine a square, we update that position with its corresponding label value in the label table and we update values in the probability table using the approach we outlined above. Again, our loop completes when we find all of the bombs in the table.

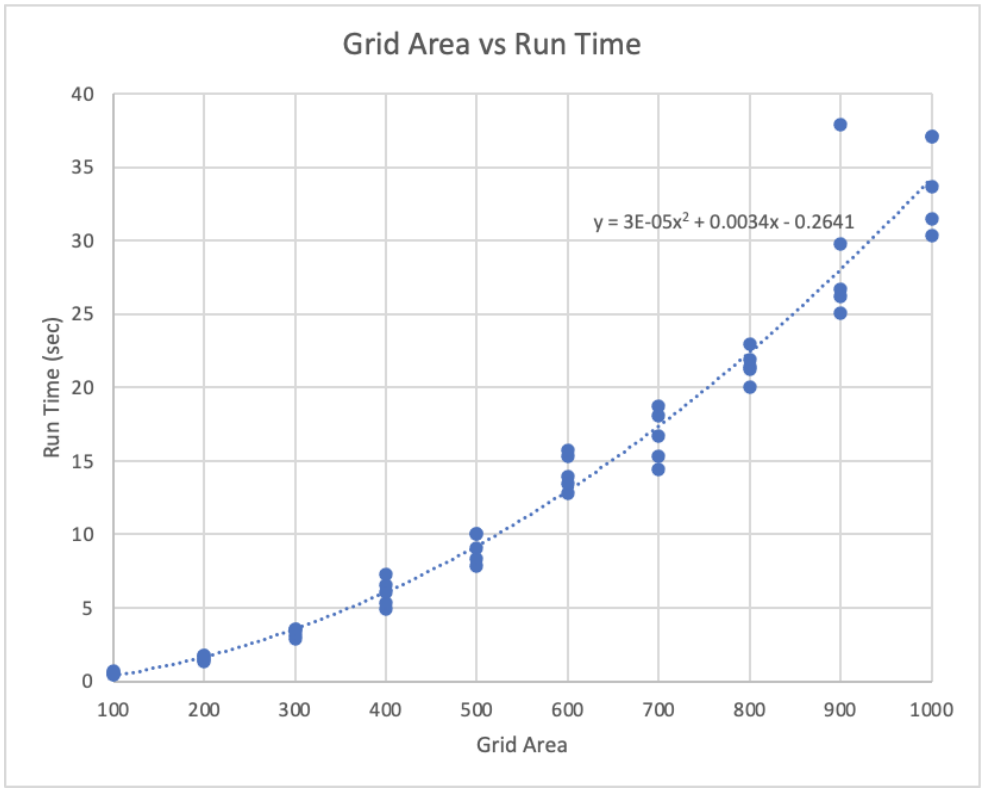
Justification

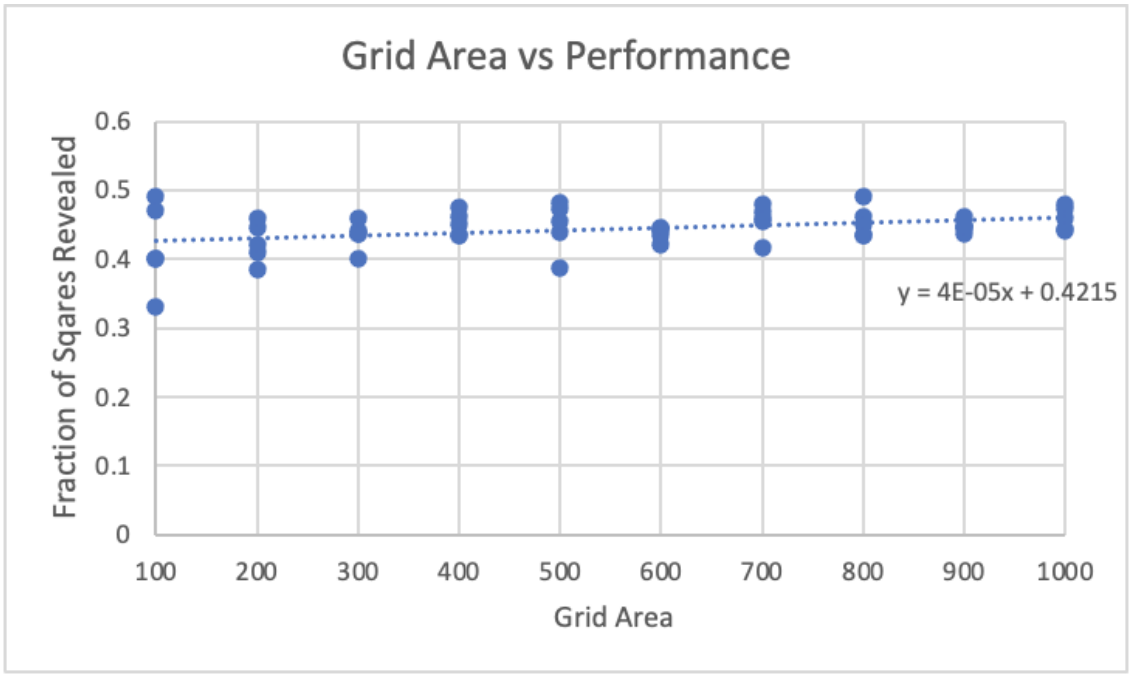
Although we improve performance through probability values, we still follow a generally greedy approach with this algorithm. The main loop of our algorithm continues until all bombs have been mined. After we mine a square, we do not reconsider that square. Following this logic, we are guaranteed to eventually mine every bomb in the table. This guarantees the correctness of our algorithm because every bomb will be mined. Our performance improvements do not change the correctness of this process.

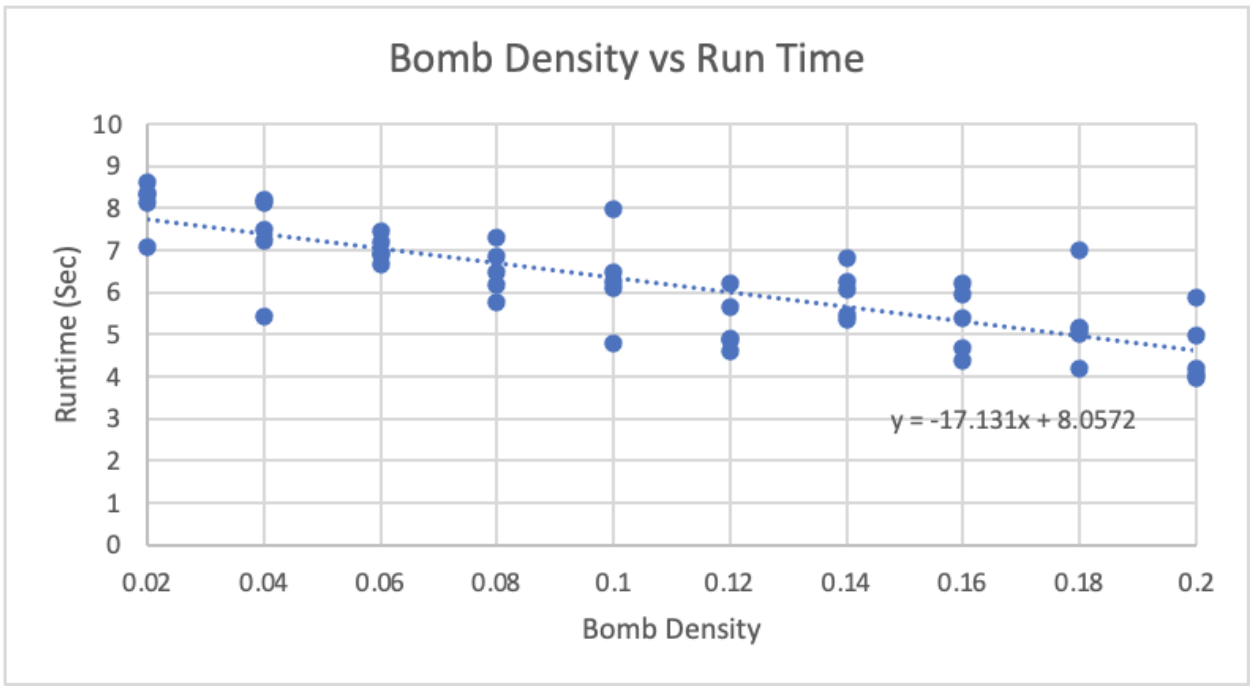
Run Time : O(n^2) where n = height * width

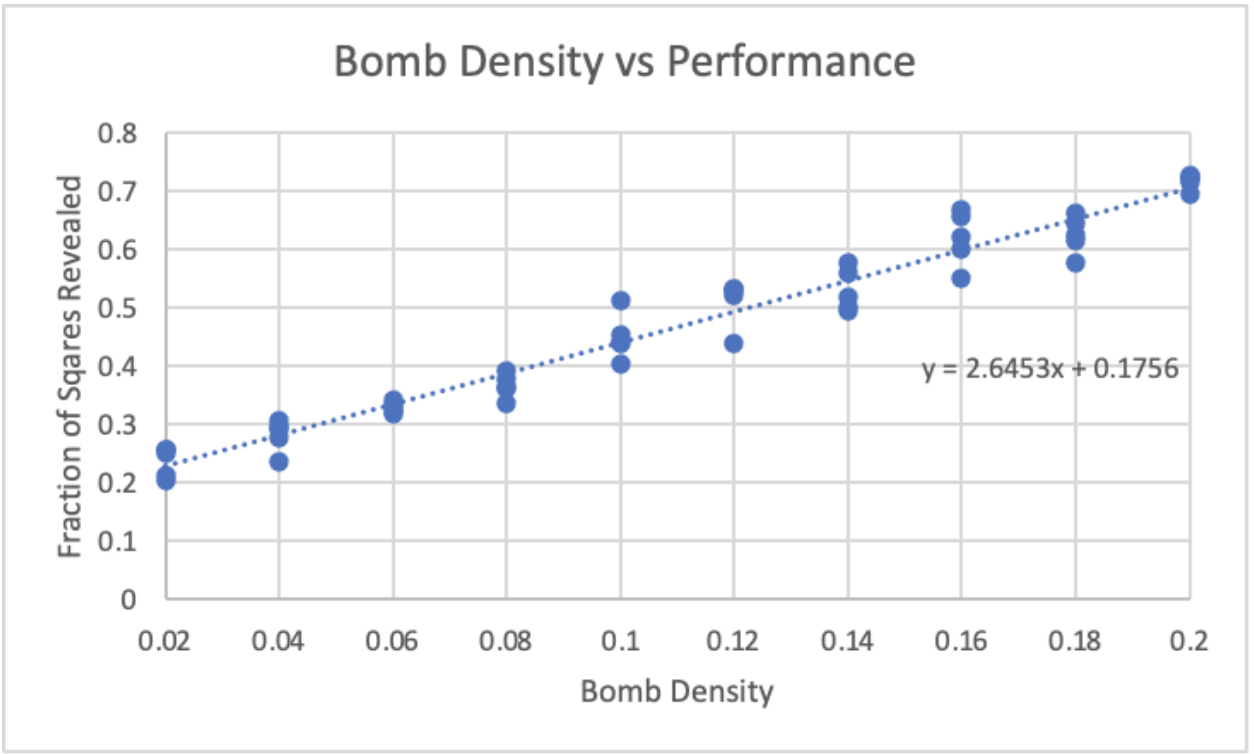
The worst case algorithm will occur in the case that all squares are 0 (the lowest bomb density). This is because if a square is 0, then we cannot build any intuition from these blocks. We would have to mine every square, which would be n. Then, for every square mined, our

algorithm would have to loop through every value in the probability table in order to find the max probability values. This process of finding the max probability value is an O(n) process. Overall then, this would be an O(n^2) algorithm.






Bomb Density vs Run Time
Runtime (Sec)
10
9
8
7
6
5
4
3
2
1
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
y = -17.131x + 8.0572


Bomb Density vs Performance
Fraction of Sqares Revealed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
y = 2.6453x + 0.1756

Statistics

Here you can see the statistics we compiled for our first algorithm. Most of the discussion pertaining to these statistics will be had when we finish our discussion of the second algorithm, so we can compare them. Furthermore, note the fact that the best fit line for grid area x runtime exhibits a n^2 curve. This further proves that this algorithm is a polynomial time algorithm. Another interesting result is that as bomb density increases, the run time decreases. We will discuss this later.

Algorithm 2

Description

For this algorithm, we wanted to really leverage our runtime while keeping performance consistent. As a result, we had to make a simple, elegant solution that would still achieve our intended purpose: minimizing the amount of squares we open to eventually open all bombs. We began by approaching the problem similar to a divide and conquer approach, although in the end applying more of a minimal constraint satisfaction approach. We partition the grid into smaller 3 x 3 grids where possible (we later discuss our solution to dealing with edge cases). We then mine the middle square and look at the value of bombs around that node. This value now acts as our constraint. We then randomly mine around this middle square until the number of bombs that we find equals the constraint. If we mine the bomb in the middle, then we simply mine all squares in our 3 x 3 grid. This is because we wouldn't be able to quickly derive any other intuition from elsewhere, although as we soon see, this does not dramatically affect the performance of the algorithm. We continue this process to mine all bombs in the original grid.

However, we are not able to evenly partition the original graph if either the `height(original grid) % 3 == 1` or `width(original grid) % 3 == 1.` In terms of edge cases, the grid sizes provided cannot be divided in a way which leaves more than a single column or row which has not been checked. In these cases, we simply create a new 3 x 3 grids which encapsulate either the row or column we have not checked. In the case that both the column and row have not been checked, we also check the bottom right most square.

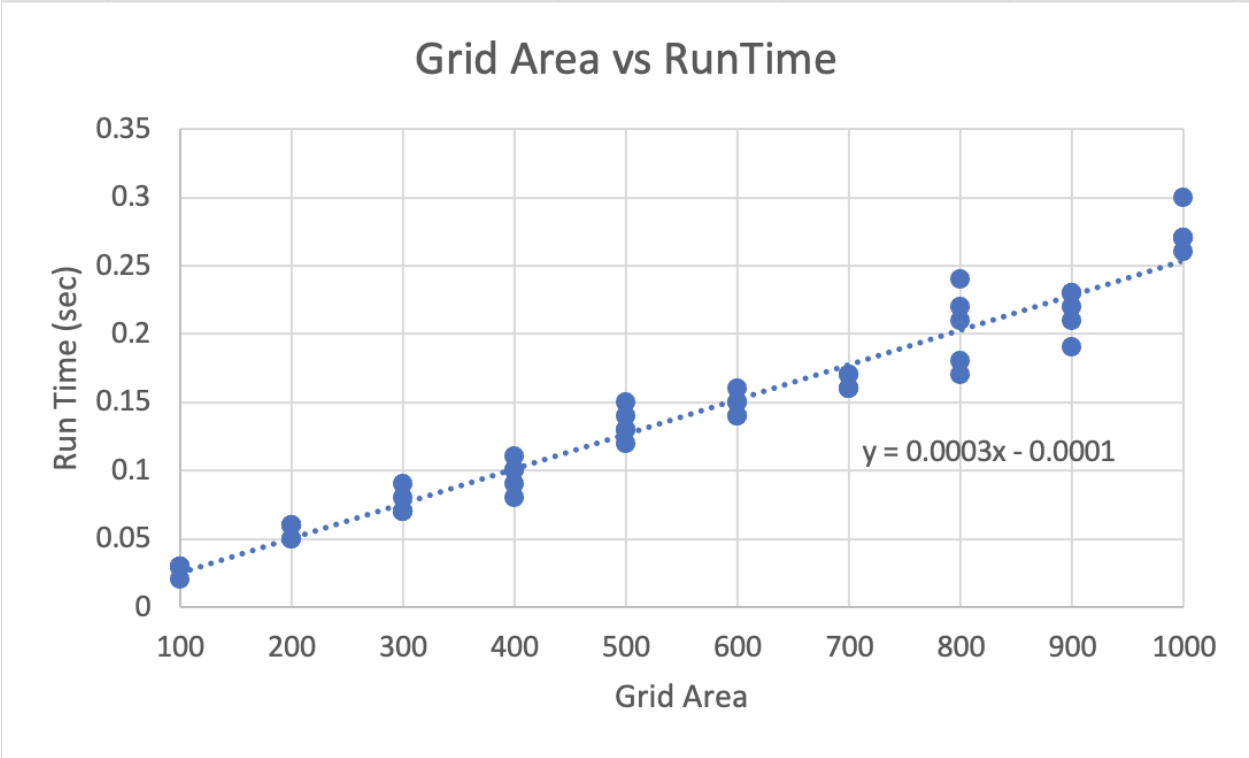
Justification

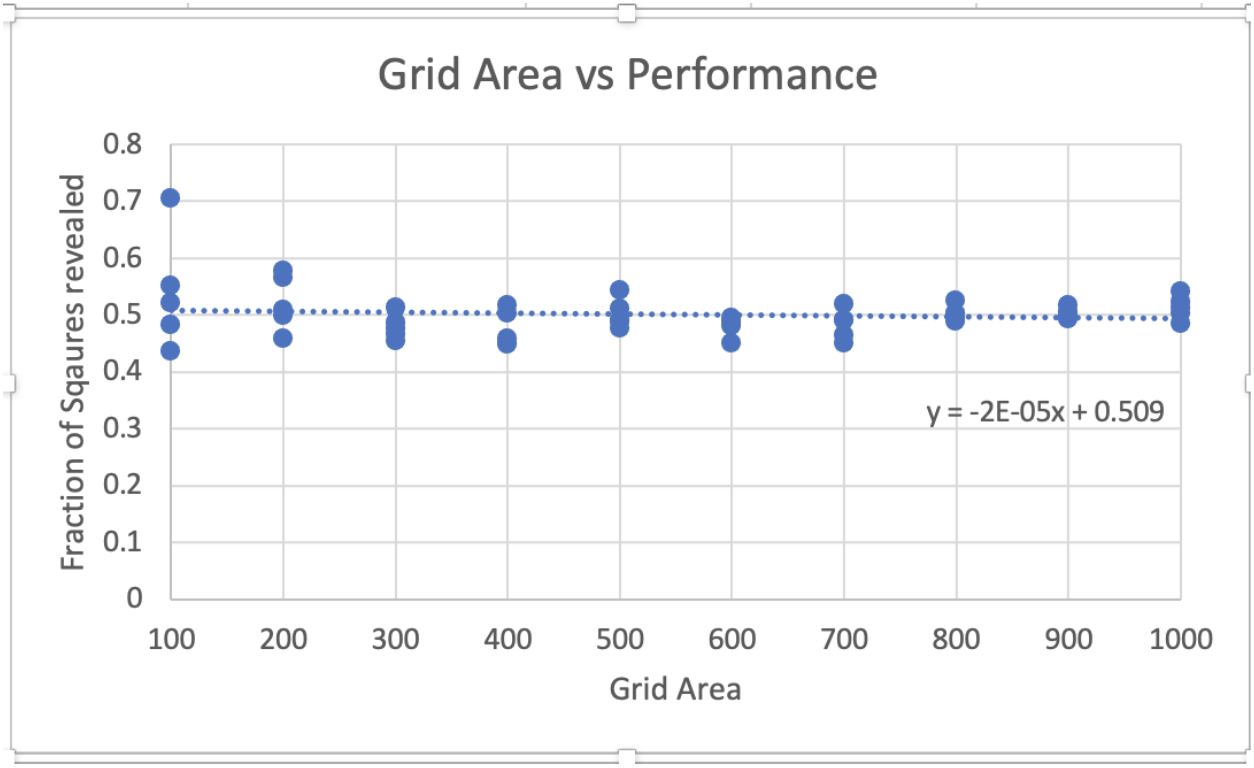
The justification of this algorithm is quite straight forward. We are simply dividing our original grid and then considering every case where a bomb can be present in our subdivisions. Our constraints simply allow us to stop mining when we deterministically know that there cannot be a bomb present in our subdivision. Since we properly divide our grid, we consider every case where there can be a bomb. Also, in terms of edge cases, we have shown we have a method to consider every case where we cannot originally evenly subdivide. This does not leave any room for our algorithm to be incorrect.

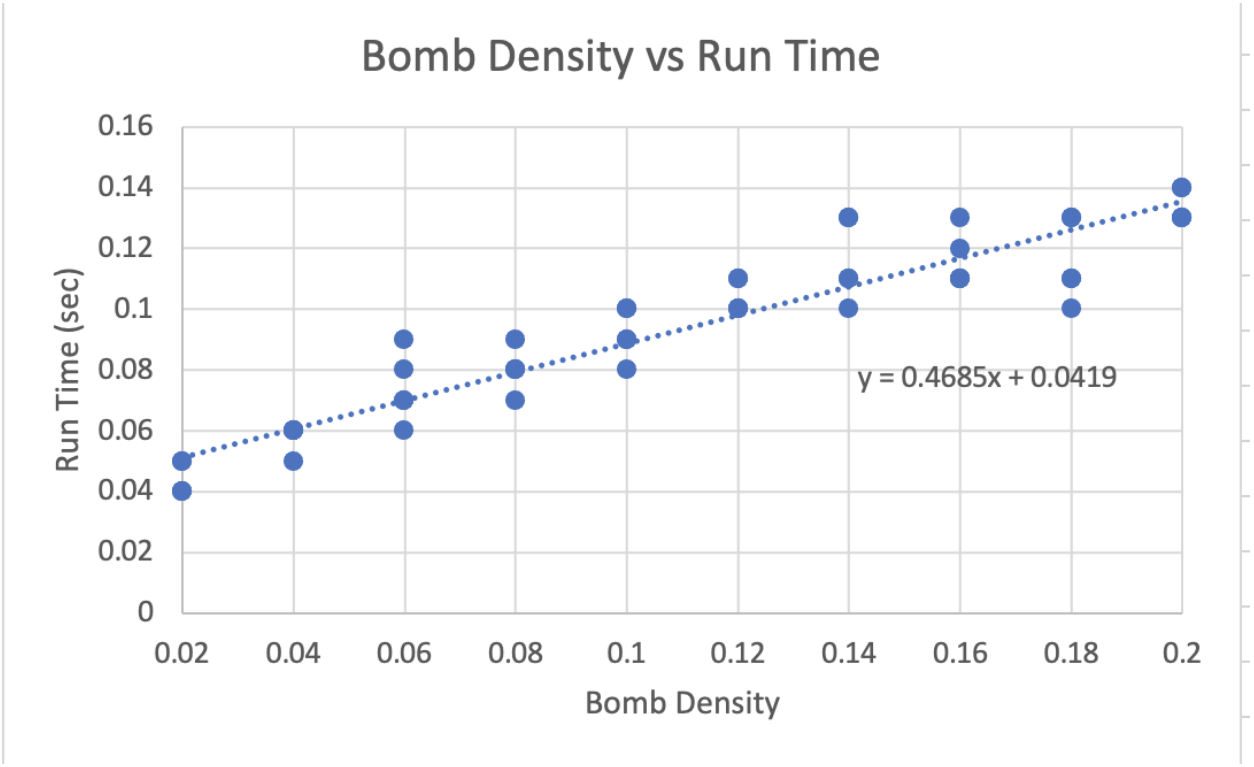
Runtime: O(n) where n = height x width

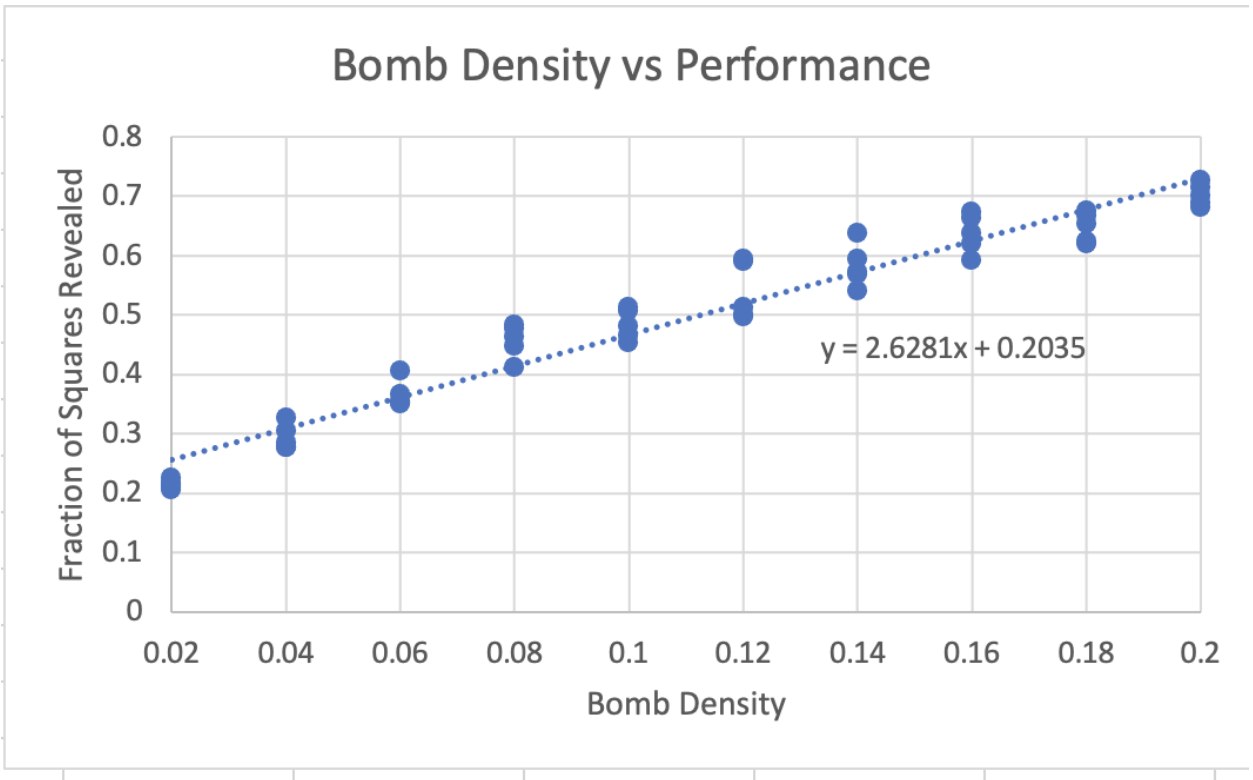
There are multiple paths that we can take to describe that our algorithm is O(n). Most simply, in the worst case, there will be a bomb in every middle square. Here, we would pass through every square once and mine it, resulting in an O(n) runtime. This case does not change if every square in the entire grid is a bomb. Again, this means our runtime is O(n). We will explore the interchange of runtime and performance with the statistics.

Graphs


Grid Area vs RunTime
Run Time (sec)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
y = 0.0003x - 0.0001
100
200
300
400
500
600
700
800
900
1000
Grid Area


Grid Area vs Performance
Fraction of Sqaures revealed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y = -2E-05x + 0.509
100
200
300
400
500
600
700
800
900
1000
Grid Area

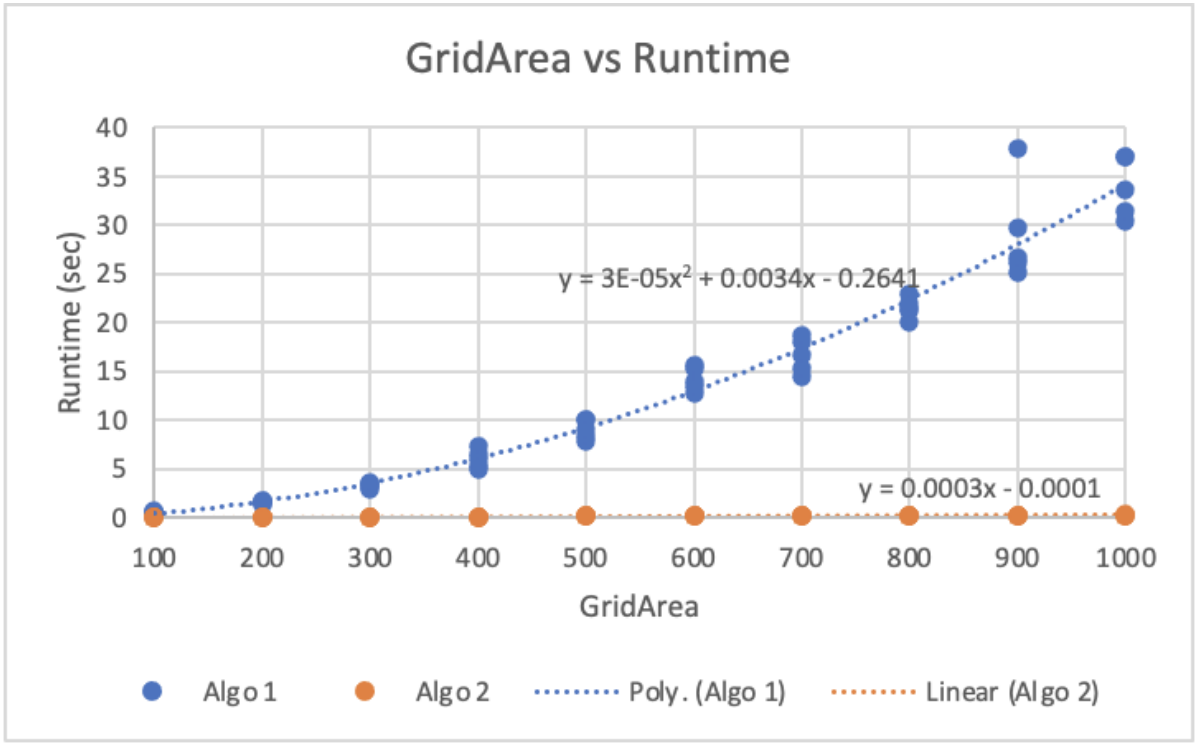

Bomb Density vs Run Time
Run Time (sec)
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
y = 0.4685x + 0.0419

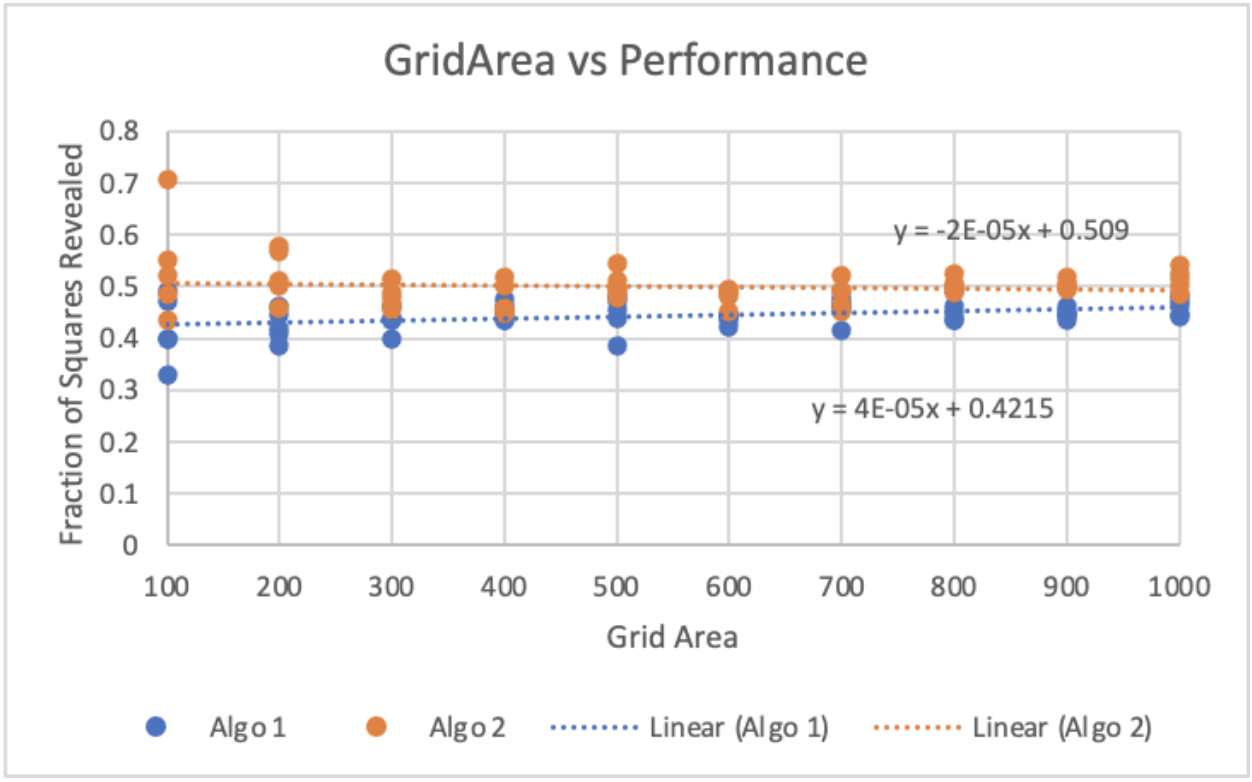

Bomb Density vs Performance
Fraction of Squares Revealed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
y = 2.6281x + 0.2035

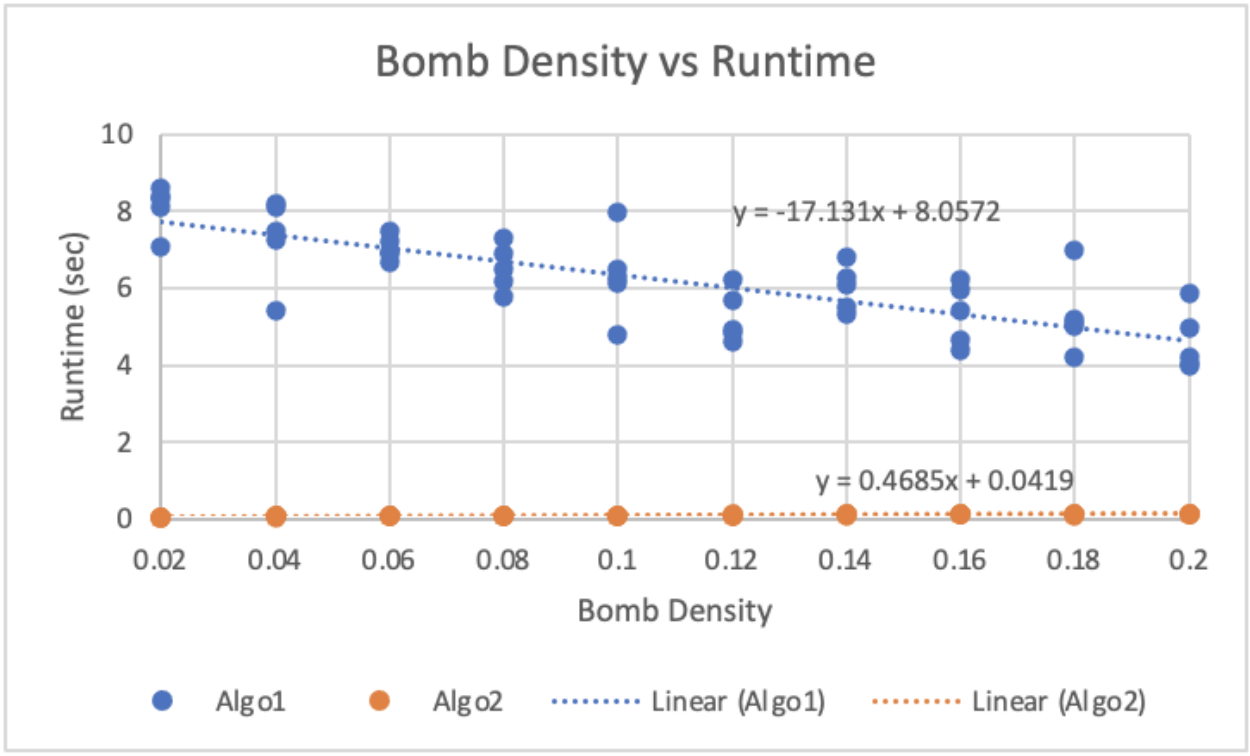
Statistics

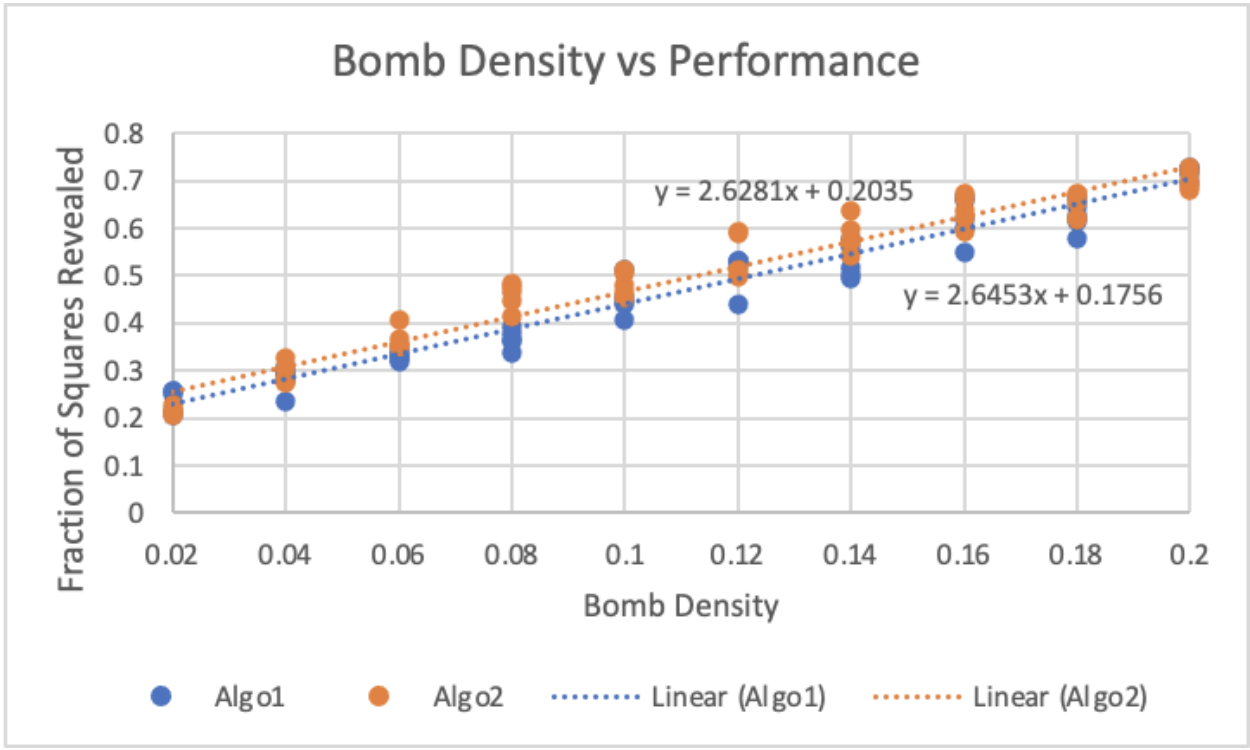
As we can see, our data shows the same things which we outlined in our description/analysis of the algorithm. We primarily hoped for this algorithm to be significantly faster than our algorithm 1, which certainly ended up being the case. We can see that the graph shows we have a linear time complexity and that our performance doesn't deteriorate as we work on larger grid sizes. Of course, since this algorithm relies on randomness, for every data point, we averaged 5 runs of our algorithm. We will include more statistics in our final comparison report.

Comparison of Algorithms






Bomb Density vs Runtime
Runtime (sec)
10
8
6
4
2
0
y = -17.131x + 8.0572
y = 0.4685x + 0.0419
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
Algo1
Algo2
Linear (Algo1)
Linear (Algo2)


Bomb Density vs Performance
Fraction of Squares Revealed
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y = 2.6281x + 0.2035
y = 2.6453x + 0.1756
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Bomb Density
Algo1
Algo2
Linear (Algo1)
Linear (Algo2)

Statistics Comparison

Algorithm 1's primary objective was to minimize the amount of squares mined to find all the bombs. In this process, we tried to optimize this value as much as possible, letting the runtime become slower in the process. However, our second algorithm sought to still minimize the amount of squares mined to find all the bombs, however, we put more of an emphasis on the speed of the algorithm. For example, in the grid area x runtime graph, the slowest case for algorithm 1 was 35 - 40 seconds, whereas the slowest case for algorithm 2 was around .3 seconds (averaged). Furthermore, in this graph, algorithm 1 had a quadratic runtime while algorithm 2 had a linear runtime. We can see this trade off in performance, looking at the grid area x performance graph which compares the performance of the two algorithms (by having both curves on the graph). While both algorithms were generally effective, algorithm 1 consistently performed better at minimizing the amount of squares mined.

For both algorithms, the fraction of squares revealed was larger than the bomb density. This is obvious, as it was the constraint on the holistic problem. It is also interesting to note the intuition that we get from bomb density x runtime comparisons. For algorithm 1, we can again see that as bomb density increases, the runtime also decreases. This is because this algorithm relies on the existence of bombs to build intuition of the board (and subsequently its environment). While it may seem on the comparison graph that the bomb density does not affect the runtime of algorithm 2, this is not the case. As the bomb density increases, the runtime also increases for algorithm 2, as outlined through its own graph. Since algorithm 2 only builds intuition from middle blocks, as bombs increase, it will have mine more squares. Looking at

bomb density x performance comparisons, we can see that the two algorithms followed the same trend. As there exist more bombs, we have to mine more squares to get all of the bombs.

All in all, our two algorithms performed very effectively. Our algorithm 1 sought to build intuition through its environment and to do a lot of number crunching to make extremely informed decisions. Our algorithm 2 was meant to be elegant, simple, and effective, which it was.