For generating different paths, I did not use any sophisticated packages or data structures. I simply used python’s list and went on traversing the triangle from the top and adding the path as I went along. For example, I first added the top 1 element on the list, and using that element, I created 2 valid paths for the second level from the top. Again, using these two paths, I added 4 valid paths for level 3. At this point, I have 4 paths if the base of the triangle is 3. Therefore, for the triangle with base n, I will repeat the same process n times. The brute force algorithm should have the time efficiency of O(n\*2^n). However, this version of brute force has O(2^n) time efficiency because you are not going to the top of the triangle each time, but instead using the already generated path to add on more paths. This is still a brute force approach since this method generates all the possible paths.

Additionally, I also used python’s time and random package for keeping track of run time and generating random numbers respectively.

1. The algorithm is given in the attached python file.
2. The algorithm’s order of efficiency is Θ(). This is because in the brute force algorithm, for n (the size of the base of the triangle), the total number of paths that needs to be checked is . For each path, the algorithm visits n nodes. The brute force algorithm that I used is a little augmented version because I do not go to the top of the triangle every time and generate a path, instead generate the path from top to bottom step by step, each time adding new valid nodes to the existing path.
3. The scatter plot is given below. We can see that the plot is growing exponentially. Looking closely at the values, we can see that when n increases by 1, the value of t increases by 2 times than the previous value. This is exactly what I expected from my algorithm as described in section b.

Note: The reason I am plotting arithmetic sequence for n instead of geometric is that when I used geometric sequence, the system crashed when I reached 32. Below that, I will only have 4 points and the curve did not look as smooth.

A close up of a map

Description automatically generated

1. For my system, the largest value of n for which the algorithm can solve the problem in 1 minute is 24. Note: This is the value based on my algorithm which has the time efficiency of O(2^n).
2. Note: The calculation below is based on my algorithm and my system specification. From my plot, I found out that for my system, it takes approximately 30 seconds to solve the problem with base n=24. We know that the order of growth of the algorithm is given as Θ(). Therefore,

When t2 = 24 hours = 86400 secs

Solving for n2, we get approximately 35.

When t2 = one year = 31556952 secs

Solving for n2, we get approximately 44.