Rahul Thapa

Here are some of the test cases that I used in my experiment. In the tuple, the first value is the triangle, second value is the minimum path value, and third value is the system time it took to run.

1. `(1, [[1]], 1.9073486328125e-06)`
2. `(6, [[3], [3, 4]], 4.5299530029296875e-06)`
3. `(15, [[3], [7, 8], [5, 6, 4]], 6.198883056640625e-06)`
4.

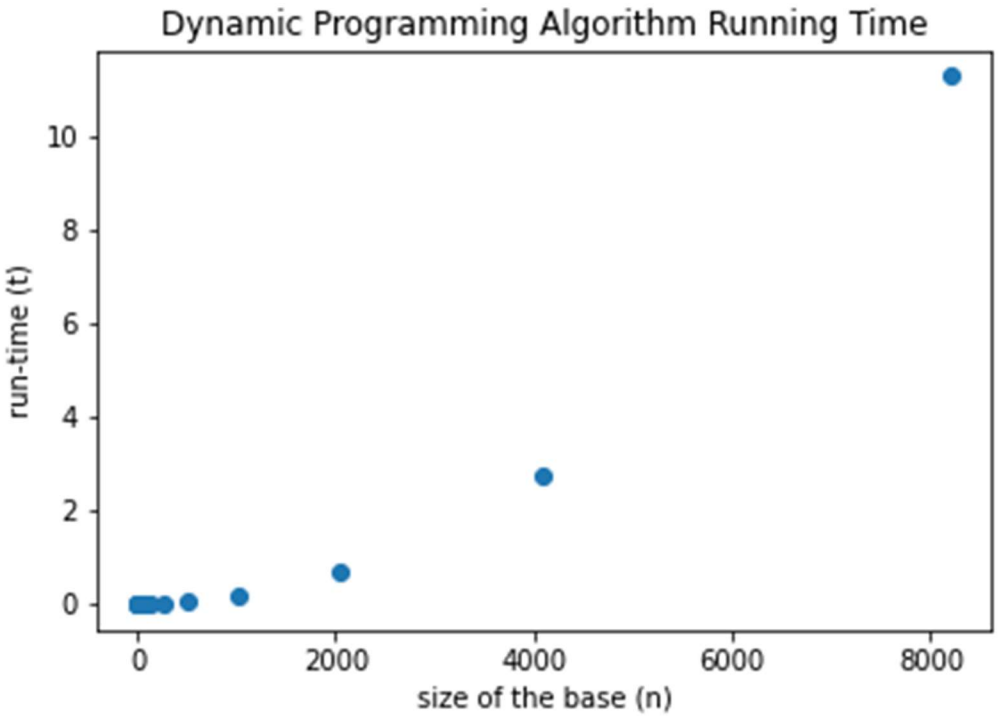
```
(378,
 [[51],
  [73, 54],
  [83, 69, 65],
  [84, 56, 31, 71],
  [65, 92, 41, 50, 18],
  [90, 37, 90, 15, 79, 47],
  [2, 60, 64, 50, 54, 9, 11],
  [98, 47, 35, 40, 61, 23, 73, 66],
  [10, 15, 68, 34, 16, 50, 32, 24, 32],
  [55, 23, 89, 30, 15, 59, 46, 79, 79, 3]],
   2.0742416381835938e-05)
```

All these test cases passes.

a. The algorithm is given in the attached python file.
b. The algorithm's order of efficiency is $\Theta(n^2)$. This is because I am inside my main dynamic programming algorithm, I have a nested for loop. The first loop runs from n-2 down to 0. For each of those iteration, another loop goes through each row in the triangle array. However, each row in triangle array does not have same length. The length of row increases as 1, 2, 3 … n. Therefore, the sum really is

$$1 + 2 + 3 + 4 + \dots (n-1) = (n-1)(n-4)/2$$

   Note, the way my algorithm is set up, it does not go through the last row with n elements. This falls under quadratic class $\Theta(n^2)$.
c. The scatter plot is given below. We can see that the plot is growing more than linearly but certainly less than exponentially. I hypothesize it to be some polynomial degree, possibly quadratic just looking at the graph.
   Looking closly at the values, I found out that the curve is increasing quadratically. I figured that out by taking two points and checking the quadratic relation, which holds true.



d. For my system, the largest value of n for which the algorithm can solve the problem in 1 minute is approximately 19074. Note: This is the value based on my algorithm which has the time efficiency of $\Theta(n^2)$.

e. 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 2.7668 seconds to solve the problem with base n=4096. We know that the order of growth of the algorithm is given as $\Theta(n^2)$. Therefore,

$$\frac{2.7668}{t} = \frac{4096^2}{n^2}$$

When t = 24 hours = 86400 secs

$$\frac{2.7668}{86400} = \frac{4096^2}{n^2}$$

Solving for n, we get approximately 723815.

When t2 = one year = 31556952 secs

$$\frac{2.7668}{31556952} = \frac{4096^2}{n^2}$$

Solving for n, we get approximately 13833070.