CSCI_6212_11 - Project 3 - Magical Eggs and Tiny Floors (*aka* The Cell Phone Drop Testing Problem)

Team 10 - Namana Y Tarikere – G21372717, Siddhesh Kocharekar – G41505497, Parithosh Dharmapalan – G27479699, Pooja Srinivasan – G33105018

1. Problem Statement: We are required to analyse the below problem statement:

Input: We are given m eggs and an n floor building. We need to figure out the highest floor an egg can be dropped without breaking, assuming that (i) all eggs are identical, (ii) if an egg breaks after being dropped from one floor, then the egg will also break if dropped from all higher floors, and (iii) if an egg does not break after being thrown from a certain floor, it retains all of its strength and we can continue to use that egg.

Objective: The goal is to minimize the number of throws and to describe an algorithm to find the floor from which to drop the first egg.

2. Problem Analysis: The above problem statement can be solved using binary search with a dynamic programming approach. In this Implementation, we use m to represent the number of eggs and n to denote the number of floors. The function find_first_drop(m, n) is responsible for determining the initial floor from which the egg should be dropped such that it minimizes the number of throws. The pseudocode of this algorithm is given below:

```
def find_first_drop(egg, fl):
    dp = [[0] * (fl + 1) for _ in range(egg + 1)]
    for i in range(1, fl + 1):
        dp[1][i] = i
    for i in range(2, egg + 1):
        for j in range(1, fl + 1):
            dp[i][j] = float('inf')
            low, high = 1, j
            while low <= high:
                mid = (low + high) // 2
                br_egg = dp[i - 1][mid - 1]
                no_br_egg= dp[i][j - mid]
                answer = 1 + max(br_egg, no_br_egg)
                dp[i][j] = min(dp[i][j], answer)
                if br_egg < no_br_egg:
                    low = mid + 1
                else:
                    high = mid - 1
    return low
```

Example Test Case: Suppose we have 3 eggs (m) and 5 floors (n). According to the pseudocode, we drop the first egg from the 3rd floor. In this instance, assume that the egg breaks on the 3rd floor and we proceed to check the floors below. In the worst-case scenario, imagine that the 2nd egg also breaks on the 2nd floor, leaving us with only 1 egg. Again, the remaining egg breaks on the 1st floor. In this worst-case scenario, the minimum number of throws required is 3. A similar situation arises if we drop the egg from the 3rd floor, but this time, let's assume that the egg doesn't break. In this case, we still have 3 eggs remaining. We continue by trying one floor above and find that the egg doesn't break on the 4th floor. We proceed to the 5th floor, where the egg finally breaks. Even in this scenario, the minimum number of throws required remains at 3. The algorithm operates with time complexity of **O(m log n)**.

3. Time Complexity: The following calculations were done to acquire the time complexity of the algorithm.

- **Initialization:** We initialize the dynamic programming array, which takes a constant amount of time, denoted as O(1).
- **Base Cases:** Setting up the base cases in the dynamic programming array for one egg and one floor involves iterating through n floors, this takes linear time with time complexity is O(n).
- **Binary Search with Dynamic Programming:**
    - **Outer Loop:** The outer loop runs m times, representing the number of eggs. So, the complexity here is O(m).
    - **Inner Loop (Binary Search):** The binary search loop runs log n iterations, as we repeatedly reduce the search space by half with each iteration. Thus, the time complexity is O(log n).
    - **Innermost Loop:** The innermost loop is a constant operation, regardless of the input size, making it O(1).

- Therefore the total time complexity of the binary search loop is O(m log n).
- **Overall Time Complexity:** Therefore, the overall time complexity of the algorithm is O(1+n+m log n). Since m is typically less than n, we can simplify the time complexity to **O(m log n)**.

4. Numerical Results

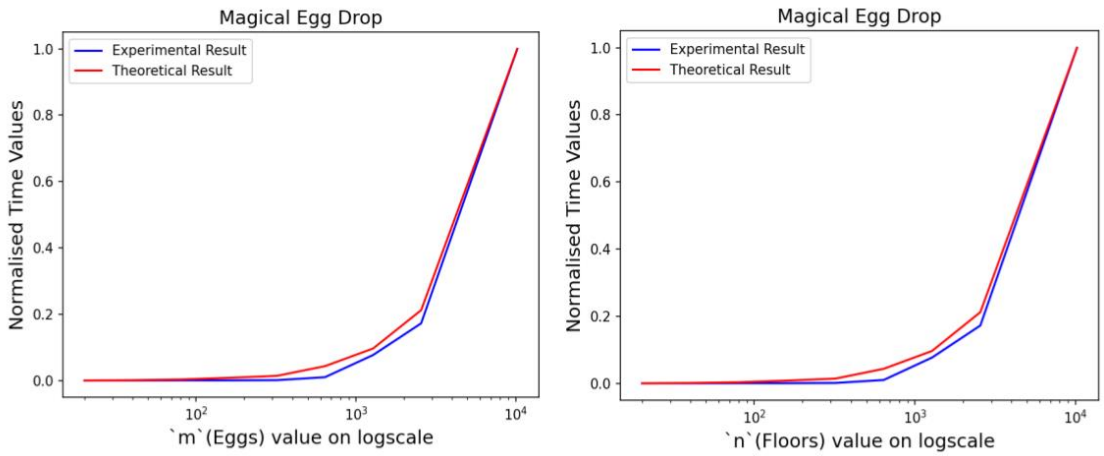
4.1 Data Normalization Notes:

Average of Experimental results is: **4056473875.24**
Average of Theoretical result is: **2102.426411**
Scaling constant = Average of Experimental result / Average of Theoretical result
Therefore, Scaling Constant for normalizing the theoretical results = **1929424.9**

4.2 Output Numerical Data

Eggs (m)	Floors (n)	Experimental Result in ns	Theoretical Result (m * log n)	Scaling Constant	Scaled Theoretical Result
2	20	64773.60	8.64385619		16677671.37
4	40	309214.20	21.28771238		41073042.33
8	80	1366246.40	50.57542476		97581483.86
16	160	5202459.00	117.1508495		226033766.1
32	320	25614709.60	266.301699		513809129
64	640	131145920.40	596.6033981		1151101452
128	1280	354687222.00	1321.206796		2549169290
256	2560	2735510732.00	2898.413592		5592271355
1024	10240	33254363600.00	13641.65437		26320547619
	Average	**4056473875.24**	**2102.426411**	**1929424.9**	

5. Graph and Observations:



The plot of the experimental results **increases exponentially** along with the increase in theoretical result at all the points on the graph for respective increase in values of eggs and floors.

6. Conclusions

The line plots of the experimental and theoretical analysis are intersecting and seem to be convergent. Hence the Big O notation for the above problem statement using dynamic programming is **O(m log n)** where m is the number of eggs, and n is the number of floors.

7. Github Link: https://github.com/namanayt/CSCI_6212_11_Algorithms_Project_3_Team_10