**Problem :**[**https://leetcode.com/problems/super-egg-drop/**](https://leetcode.com/problems/super-egg-drop/)

n=no of floors, k= no of eggs

**Approach 1 : O(k\*(n^2))**

-> Initially we have ‘n’ floors & ‘k’ eggs. (Imagine 0 to n written from bottom to top).

-> Try dropping egg from every floor from i=1 to n.

If the egg breaks at floor ‘i’ then it will break at all floors above also, then we only have to search for floors from 0 to i-1, so basically just ‘i-1’ floors.

If the egg doesn’t break , then it will not break at any of the floors below, so we just have to check for floors above , means from i+1 to n, which are n-i floors.

-> It feels as if : Why are we trying to drop from all floors,we can always keep dropping from the middle floor and that should give optimal answer, but that’s not always true,as we either go up or down and based on that number of eggs change ,

->So this problem  **depends on (1)no of eggs left, (2)no of floors left**

(**so one side we might have less no of floors but the eggs might also be less,so probability of no of eggs reaching 1 increase, when eggs=1, no of moves=no of floors ,as we need to start from bottom to check one by none**)

-> So, for every i , find 1+max(dp[i-1][k-1], dp[n-i][k]) as we want the worst case scenario.

And out of all results of all ‘i’, the minimum one is the answer.

**-> So basically we tried dropping from all floors, and at every floor ‘i’ we take**

**1+max(**dp[i-1][k-1], dp[n-i][k]**),means max moves are taken going above or below from floor‘i’?**

**And out of all results of ‘i’,we take the minimum answer,means we found out dropping from which floor is optimal.**

**Time Complexity :**

n\*k states , transition time = O(n), **so Time = O(k\*(n^2))**

**Code1 :** [**https://leetcode.com/submissions/detail/527053608/**](https://leetcode.com/submissions/detail/527053608/)

**Approach 2 : O(n\*k)**

=> Define dp[M][K]means that, given K eggs and M moves,what is the maximum number of floors that we can check.

=> The dp equation is:

dp[m][k] = 1 + dp[m - 1][k - 1] + dp[m - 1][k] ,

which means we take 1 move to a floor,

-> if the egg breaks, then we can check dp[m - 1][k - 1] floors.

-> if the egg doesn't break, then we can check dp[m - 1][k] floors.

-> So we start checking from m=1 move till we guarantee that using current ‘m’ moves we can check for all floors.

**Time Complexity :**

No of moves can be as big as no of floors = n.(eg: when k=1 egg)

So, basically a state here is dp[moves][k] and as moves can be equal to no. of floors,

So there can be n\*k states and transition time=constant.

**So time = O(n\*k)**

**Code2 :** [**https://leetcode.com/submissions/detail/527060455/**](https://leetcode.com/submissions/detail/527060455/)