Question 3 Solution:

Understanding:

According to the given question, we have a ladder with “n” rungs. We have “k” jugs. We need to find the highest safest rung in which we drop a jar with the condition in which it doesn’t break. As per the question, binary search is not an effective solution as the number of jars that’ll be broken while finding the highest safe rung will be comparatively high.

Given 2 situations, we ought to find the best case:

Case a: Given a budget of 2 jars maximum, we need to find the best-case solution with a function that grows linearly

Case b: Given a budget of k>2, such that the highest safest rung needs to be found with the function that grows in an asymptotic manner for each k.

The solution for (a):

We need to consider a function that grows linearly. Also, we have a budget of k =2. In such a case, we can think of a function of n moves. Let’s say n =3. We move the rung by 3 at a time. Assuming we have 12 rungs. We start with 3rd rung, drop the jar. If it crashes, then we get to a tentative conclusion, or we move the rungs by 3. In our situation, we try 3rd rung, assuming nothing happened (Jar did not break). Now at 6th run, we drop. It crashes. Our conclusion could be that the safest rung is somewhere between 3rd and 6th. This could result in a complexity of O(n/2), which is not the best one.

We try to get the solution where O(n/2) is not the best one. But effectively, O(n,c) could result somewhere closer. Consider c as a constant where it could be ½. With the given condition, it could be log n or n. With the given budget of 2 jars, **log n** seems ineffective, but **n** seems possible. Considering n as an optimal number of rungs and with the perfect square to make it easy for our assumption. We are assuming a number that could prove the solution. It must not be too small as the number of trials performing the first jar could result in the worst case ever possible. Now considering the function the idea could be to drop the first jar at heights that are multiples of the function n If we reach the top and the jar does not break, then we are done, and we had to drop the jar √n times. For example, *if the number of rungs is 9, the jar breaks at 9th rung, which is 3√n. We could conclude that, if the jar breaks at height n √n, then we know that the highest safe rung will be between (n − 1) √n and n √n.* As a conclusion, we could say the upper bound limit for this algorithm is √n and when n is infinity √n / n=0. This could be the possible solution for the question. We could also consider the worst-case scenario where we will have to perform N trials with the first jar (it will break at the very top, and we will limit ourselves to the topmost block) and the second jar will break only at the last trial, so we will need n/N for that too. Thus Y(n,2) = N + n/N. After taking the derivative and finding the root, N=√n so Y(n,2) = 2√n still results in the **predicted solution**, which is **O(√n).**

The solution for (b):

In this case, we are given a condition where the function $\lim_{n\to\infty}\left(\frac{fk(n)}{fk-1(n)}\right)$ grows asymptotic, rather than the previous type. Also, the budget given is k>2. Thus, to find the best case, we could say k could be 3 or 4 or any value that grows asymptotically. From the solution (a) we concluded that O(**√n)** could be the suitable best case. Now, since the budget is k>2, we can have a function like $n^{k-1/k}$. Considering this, we may assume k=3. Now, we can get the $n^{2/3}$ as the function. Now, we drop a jar at the 6/6$^{2/3}$ rung. Since there is an asymptotic growth, we can have the possible sequence that has an interval length at most $n^{k-1/k}$. We could iterate this using recursion.

We separate the total number of blocks into an assumed number. (say N). Then use the first jar to limit towards a specific block and then separate that block into several sub-blocks. This narrows our prediction into the sub block at the cost of just one jar. Now use a second jar to limit into much narrower solution until it is one. Considering the equation for this, $fk(n) = fk - 1(n)$. Substituting this in our condition, it grows asymptotically, $fk - 2(n)/n^2, fk - 3(n)/n^3$, and so on. Now, $\lim_{n\to\infty}\left(\frac{fk(n)}{fk-1(n)}\right)$ becomes $\lim_{n\to\infty}\left(\frac{f1(n)/n\text{^}k}{f1(n)/n\text{^}k-1}\right) = \lim_{n\to\infty}\left(\frac{1}{n}\right)$ which we derives to 0. Thus, this is the answer if we drop one jar from the first rung to the last till it breaks performing k drops. a second jar to limit ourselves. So, for every sub-block level we seem to require an increase and a decrease in the number of separations of the sub-block. We could conclude by considering the equation Y(n, k)= (k-1) N + n/(k^{k-1})), is easy to come up with, and by finding the root of the derivative again, N = $n^{1/k}$ and finally the solution is **O($\boldsymbol{n^{1/k}}$)**.