**Answer a)**Move 2 rungs at a time, starting from rung 2 and so on. If it crashes at the current rung, then the maximum rung is the current rung or the previous rung. This would give me O(n/2) bound which doesn’t answers correctly the question since we need a limit that tends to 0, not 2.  
A better solution to O(n) can be n/2.But it can’t be n/2 since O(n/2) is essentially O(n.k) where “k” is 1/2, so it may be sqrt(n) or log n. With 2 jars we can’t make it in log n, but we can make it in sqrt(n).

Let’s suppose that “n=9”, then divide the ladder in blocks of sqrt(n). In this case we will have 3 blocks with 3 rungs each. Test the last rung of the first block (bottom up), if it doesn’t crashes, continue by testing the last rung of each remaining block until it crashes. Now you will have the highest block it can be dropped, so you can test inside this block from bottom up until it breaks again and you will have the highest rung that a jar can be dropped. The upper bound limit for this algorithm is sqrt(n) and within the limits.

**Answer b)**For the scenario having more than 2 jars.  
We can use the similar approach of having two jars where we used the square root function we can use the cuberoot or the x.  
Let’s see, with 2 jars (“k” = 2) the equation was n/x = y, where x = y, giving us x = y = sqrt(n) which is the same as n/n^(1/2) = n^(1/2).

With more jars we could tweak this equation to be n/x = n^(1/k) until it breaks the first jar, in this case x = n^((k-1)/k), then it would go into the crash block to repeat the process with (n^((k-1)/k)) and so on until it reaches the highest rung, crashing the last jar. As you can see our algorithm will do n^(1/k)+ (n^(1/k))^(1/k-1) which will be less than n^(1/k) giving us a O(n^(1/k) bound.