Interview Cake

A building has 100 floors. One of the floors is the highest floor an egg can be dropped from without breaking.

If an egg is dropped from above that floor, it will break. If it is dropped from that floor or below, it will be completely undamaged and you can drop the egg again.

Given two eggs, find the highest floor an egg can be dropped from without breaking, with as few drops as possible.

Gotchas

We can do better than a binary

A binary search algorithm finds an item in a *sorted* list in $O(lg(n))$ time.

A brute force search would walk through the whole list, taking $O(n)$ time in the worst case.

Let's say we have a sorted list of numbers. To find a number with a binary search, we:

1. **Start with the middle number: is it bigger or smaller than our target number?** Since the list is sorted, this tells us if the target would be in the *left* half or the *right* half of our list.
2. **We've effectively divided the problem in half**. We can "rule out" the whole half of the list that we know doesn't contain the target number.
3. **Repeat the same approach (of starting in the middle) on the new half-size problem**. Then do it again and again, until we either find the number or "rule out" the whole set.

We can do this recursively, or iteratively. Here's an iterative version:

Python 3.6

```
def binary_search(target, nums):
    """See if target appears in nums"""

    # We think of floor_index and ceiling_index as "walls" around
    # the possible positions of our target, so by -1 below we mean
    # to start our wall "to the left" of the 0th index
    # (we *don't* mean "the last index")
    floor_index = -1
    ceiling_index = len(nums)

    # If there isn't at least 1 index between floor and ceiling,
    # we've run out of guesses and the number must not be present
    while floor_index + 1 < ceiling_index:

        # Find the index ~halfway between the floor and ceiling
        # We use integer division, so we'll never get a "half index"
        distance = ceiling_index - floor_index
        half_distance = distance // 2
        guess_index = floor_index + half_distance

        guess_value = nums[guess_index]

        if guess_value == target:
            return True

        if guess_value > target:
            # Target is to the left, so move ceiling to the left
            ceiling_index = guess_index
        else:
            # Target is to the right, so move floor to the right
            floor_index = guess_index

    return False
```

How did we know the time cost of binary search was $O(lg(n))$? The only non-constant part of our time cost is the number of times our while loop runs. Each step of our while loop cuts the range (dictated by `floor_index` and `ceiling_index`) in half, until our range has just one element left.

So the question is, "how many times must we divide our original list size (n) in half until we get down to 1?"

$$n * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * ... = 1$$

How many $\frac{1}{2}$'s are there? We don't know yet, but we can call that number x:

$$n * (\frac{1}{2})^x = 1$$

Now we solve for x:

$$n * \frac{1^x}{2^x} = 1$$

$$n * \frac{1}{2^x} = 1$$

$$\frac{n}{2^x} = 1$$

$$n = 2^x$$

Now to get the x out of the exponent. How do we do that? Logarithms.

Recall that $\log_{10} 100$ means, "what power must we raise 10 to, to get 100"? The answer is 2.

So in this case, if we take the $\log_2$ of both sides...

$$\log_2 n = \log_2 2^x$$

The right hand side asks, "what power must we raise 2 to, to get 2^x?" Well, that's just x.

$$\log_2 n = x$$

So there it is. The number of times we must divide n in half to get down to 1 is $log_2 n$. So our total time cost is $O(lg(n))$

Careful: we can only use binary search if the input list is *already sorted*.

approach, which would have a worst case of 50 drops.

We can even do better than **19** drops!

We can *always* find the highest floor an egg can be dropped from with a worst case of **14** total drops.

Breakdown

What if we only had *one* egg? How could we find the correct floor?

Because we can't use the egg again if it breaks, we'd have to play it safe and drop the egg from *every* floor, starting at the bottom and working our way up. In the worst case, the egg won't break until the top floor, so we'd drop the egg 100 times.

What does having *two* eggs allow us to do differently?

Since we have two eggs, we can skip multiple floors at a time until the first egg breaks, keeping track of which floors we dropped it from. Once that egg breaks we can use the second egg to try every floor, starting on the last floor where the egg *didn't* break and ending on the floor below the one where it *did* break.

How should we choose how many floors to skip with the first egg?

What about trying a binary approach? We could drop the first egg halfway up the building at the 50th floor. If the egg doesn't break, we can try the 75th floor next. We keep going like this, dividing the problem in half at each step. As soon as the first egg breaks, we can start using our second egg on our (now-hopefully narrow) range of possible floors.

If we do that, what's the **worst case number of total drops?**

The worst case is that the highest floor an egg won't break from is floor 48 or 49. We'd drop the first egg from the 50th floor, and then we'd have to drop the second egg *from every floor* from 1 to 49, for a total of 50 drops. (Even if the highest floor an egg won't break from is floor 48, we still won't know if it will break from floor 49 until we try.)

Can we do better than this binary approach?

50 is probably too many floors to skip for the first drop. In the worst case, if the first egg breaks after a *small* number of drops, the second egg will break after a *large* number of drops. And if we went the *other* way and skipped **1** floor every time, we'd have the *opposite* problem! What would the worst case floor be then?

The worst case would be floor 98 or 99—the first egg would drop a *large* number of times (at every floor from 2-100 skipping one floor each time) and the last egg would drop a *small* number of times (only on floor 99), for a total of 51 drops.

Can we balance this out? Is there some number between 50 and 1—the number of floors we'll skip with each drop of the first egg—where the first and second eggs would drop close to the *same* number of times in the worst case?

Yes, we could skip 10 floors each time. The worst case would again be floor 98 or 99, but we'd only drop the first egg 10 times and the second egg 9 times, for a total of **19 drops!**

Is that the best we can do?

Let's look at what happens with this strategy *each time the first egg doesn't break*. **How does the worst case total number of drops change?**

The worst case total number of drops *increases by one each time the first egg doesn't break*

For example, if the egg breaks on our first drop from the 10th floor, we may have to drop the second egg at each floor between 1 and 9 for a worst case of 10 total drops. But if the egg breaks when we skip to the 20*th* floor we will have a worst case of 11 total drops (once for the 10th floor, once for the 20th, and all of the floors between 11 and 19)!

How can we keep the worst case number of drops from increasing each time the first egg doesn't break?

Since the maximum number of drops increases by one each time we skip the *same amount of floors*, we could skip *one fewer floor* each time we drop the first egg!

But how do we choose how many floors to skip the *first* time?

Well, we know two things that can help us:

1. **We want to skip as few floors as possible the *first* time we drop an egg,** so if our first egg breaks right away we don't have a lot of floors to drop our second egg from.
2. **We *always* want to be able to reduce the number of floors we're skipping *by one*.** We don't want to get towards the top and not be able to skip any floors any more.

Now that we know this, can we figure out the ideal number of floors to skip the first time?

To be able to decrease the number of floors we skip by one *every time* we move up, and to minimize the number of floors we skip the first time, we want to end up skipping *just one floor at the very top*. Can we model this with an equation?

Let's say n is the number of floors we'll skip the first time, and we know 1 is the number of floors we'll skip last. Our equation will be:

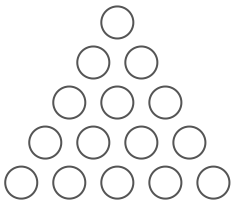
$$n + (n-1) + (n-2) + \ldots + 1 = 100$$

How can we solve for n? Notice that we're summing every number from 1 to n.

The left side is a triangular series.

A **triangular series** is a series of numbers where each number could be the row of an equilateral triangle.

So 1, 2, 3, 4, 5 is a triangular series, because you could stack the numbers like this:



Their sum is 15, which makes 15 a **triangular number**.

A triangular series *always* starts with 1 and increases by 1 with each number.

Since the only thing that changes in triangular series is the value of the highest number, it's helpful to give that a name. Let's call it n.

Python 3.6

```
# n is 8
1, 2, 3, 4, 5, 6, 7, 8
```

Triangular series are nice because no matter how large n is, it's always easy to find the total sum of all the numbers.

Take the example above. Notice that if we add the first and last numbers together, and then add the second and second-to-last numbers together, they have the same sum! This happens with *every* pair of numbers until we reach the middle. If we add up all the pairs

of numbers, we get:

```
1 + 8 = 9
2 + 7 = 9
3 + 6 = 9
4 + 5 = 9
```

Python 3.6

This is true for *every triangular series*:

1. **Pairs of numbers** on each side will always **add up to the same value**.
2. That value will always be **1 more than the series' n**.

This gives us a pattern. Each pair's sum is $n + 1$, and there are $\frac{n}{2}$ pairs. So our total sum is:

$$(n + 1) * \frac{n}{2}$$

Or:

$$\frac{n^2+n}{2}$$

Ok, but does this work with triangular series with an *odd* number of elements? Yes. Let's say $n = 5$. So if we calculate the sum by hand:

$$1 + 2 + 3 + 4 + 5 = 15$$

And if we use the formula, we get the same answer:

$$\frac{5^2+5}{2} = 15$$

One more thing:

What if we know *the total sum*, but we *don't* know the value of n?

Let's say we have:

$$1 + 2 + 3 + \ldots + (n - 2) + (n - 1) + n = 78$$

No problem. We just use our formula and set it equal to the sum!

$$\frac{n^2+n}{2} = 78$$

Now, we can rearrange our equation to get a *quadratic equation* (remember those?)

$$n^2 + n = 156$$

$$n^2 + n - 156 = 0$$

Here's the quadratic formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If you don't really remember how to use it, that's cool. You can just use an online calculator. We don't judge.

Taking the positive solution, we get $n = 12$.

So for a triangular series, remember—the total sum is:

$$\frac{n^2 + n}{2}$$

Knowing this, can we simplify and solve the equation?

We know the left side reduces to:

$$\frac{n^2 + n}{2}$$

so we can plug that in:

$$\frac{n^2 + n}{2} = 100$$

and we can rearrange to get a quadratic equation:

$$n^2 + n - 200 = 0$$

which gives us 13.651.

We can't skip a fraction of a floor, so how many floors should we skip the first time? And what's our final **worst case total number of drops**?

Solution

We'll use *the first egg* to get a *range of possible floors* that contain the highest floor an egg can be dropped from without breaking. To do this, we'll drop it from increasingly higher floors until it breaks, skipping some number of floors each time.

When the first egg breaks, **we'll use *the second egg* to find the *exact* highest floor** an egg can be dropped from. We only have to drop the second egg starting from the last floor where the first egg *didn't* break, up to the floor where the first egg *did* break. But we have to drop the second egg one floor at a time.

With the first egg, if we skip the *same number of floors every time*, the worst case number of drops will increase by one *every* time the first egg doesn't break. To counter this and keep our worst case drops *constant*, **we decrease the number of floors we skip *by one* each time we drop the first egg**.

When we're choosing how many floors to skip the *first* time we drop the first egg, we know:

1. **We want to skip as few floors as possible,** so if the first egg breaks right away we don't have a lot of floors to drop our second egg from.
2. **We *always* want to be able to reduce the number of floors we're skipping.** We don't want to get towards the top and not be able to skip floors any more.

Knowing this, we set up the following equation where n is the number of floors we skip the first time:

$$n + (n-1) + (n-2) + \ldots + 1 = 100$$

This triangular series reduces to $n * (n+1)/2 = 100$ which solves to give $n = 13.651$. We round up to 14 to be safe. So our first drop will be from the 14th floor, our second will be 13 floors higher on the 27th floor and so on until the first egg breaks. Once it breaks, we'll use the second egg to try every floor starting with the last floor where the first egg didn't break. At worst, we'll drop both eggs a combined total of 14 times.

For example:

```
Highest floor an egg won't break from
13

Floors we drop first egg from
14

Floors we drop second egg from
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Total number of drops
14
```

```
Highest floor an egg won't break from
98

Floors we drop first egg from
14, 27, 39, 50, 60, 69, 77, 84, 90, 95, 99

Floors we drop second egg from
96, 97, 98

Total number of drops
14
```

What We Learned

This is one of our most contentious questions. Some people say, "Ugh, this is useless as an interview question," while others say, "We ask this at my company, it works great."

The bottom line is some companies *do* ask questions like this, so it's worth being prepared. There are a bunch of these not-exactly-programming interview questions that lean on math and logic. There are some famous ones about shuffling cards (https://www.interviewcake.com/question/shuffle) and rolling (https://www.interviewcake.com/question/simulate-5-sided-die) dice (https://www.interviewcake.com/question/simulate-7-sided-die). If math isn't your strong suit, don't fret. It only takes a few practice problems to get the hang of these.