

## You have a function `Rand7()` that generates a random integer from 1 to 7. Use it to write a function `Rand5()` that generates a random integer from 1 to 5.

`Rand7()` returns each integer with equal probability. `Rand5()` must also return each integer with equal probability.

### Gotchas

Your first thought might be to simply take the result of `Rand7()` and take a modulus:

```
public int Rand5()
{
    return Rand7() % 5 + 1;
}
```

C# (beta) ▼

**However, this won't give an equal probability for each possible result.** We can write out each possible result from `Rand7()` (each of which is equally probable, per the problem statement) and see that some results for `Rand5()` are more likely because they are caused by more results from `Rand7()`:

<code>Rand7()</code>	<code>Rand5()</code>
1	2
2	3
3	4
4	5
5	1
6	2

So we see that there are two ways to get 2 and 3, but only one way to get 1, 4, or 5. This makes 2 and 3 twice as likely as the others.

The answer takes worst-case infinite time. However, we can get away with worst-case  $O(1)$  space. **Does your answer have a non-constant space cost?** If you're using recursion (and your language doesn't have tail-call optimization<sup>1</sup>), you're potentially incurring a worst-case infinite space cost in the call stack<sup>1</sup>. But replacing your recursion with a loop avoids this.

## Breakdown

Rand5() must return each integer with equal probability, but we don't need to make any guarantees about its runtime...

In fact, the solution has a small possibility of *never* returning...

## Solution

We simply "re-roll" whenever we get a number greater than 5.

```
public int Rand5()  
{  
    int result = int.MaxValue;  
    while (result > 5)  
    {  
        result = Rand7();  
    }  
    return result;  
}
```

C# (beta) ▼

So each integer 1,2,3,4, or 5 has a probability  $\frac{1}{7}$  of appearing at each roll.

## Complexity

Worst-case  $O(\infty)$  time (we might keep re-rolling forever) and  $O(1)$  space.

Note that if we weren't worried about the potential space cost (nor the potential stack overflow<sup>1</sup>) of recursion, we could use an arguably-more-readable recursive approach with  $O(\infty)$  space cost:

```
public int Rand5Recursive()  
{  
    int roll = Rand7();  
    return (roll <= 5) ? roll : Rand5Recursive();  
}
```

C# (beta) ▼

## Bonus

This kind of math is generally outside the scope of a coding interview, but: if you know a bit of number theory you can *prove* that there exists no solution which is guaranteed to terminate. Hint: it follows from the fundamental theorem of arithmetic<sup>1</sup>.

## What We Learned

Making sure each possible result has *the same probability* is a big part of what makes this one tricky.

If you're ever struggling with the math to figure something like that out, don't be afraid to *just count*. As in, write out all the possible results from `Rand7()`, and label each one with its final outcome for `Rand5()`. Then count up how many ways there are to make each final outcome. If the counts aren't the same, the function isn't uniformly random.

---

Want more coding interview help?

Check out **[interviewcake.com](https://interviewcake.com)** for more advice, guides, and practice questions.