Hop, Skip, Jump!

Problem Solving Technique: Solve specific cases
It might give you insight to the full solution!

Question: When is the minimum number of rounds 1?

Claim
The task can be done in one round if and only if a, b, and c (in some order) form an arithmetic progression.

Proof
WLOG let $a \le b \le c$ form an arithmetic progression, so let $b = a + d$ and $c = a + 2d$. Then, to make them all equal, choose $k := d$, and: apply $+3k$ to a, and apply $+2k$ to b, and apply $+k$ to c.

On the other hand, suppose we made all values equal to m in just one round. Then, rewinding one step, $m - 3k, m - 2k, m - k$ forms an arithmetic progression with common difference k.

Question: When is the minimum number of rounds 2?

Working backwards: If the task can be solved in two moves, then after one round, it should be solvable in one move. That is to say: we need to transform the numbers into an arithmetic progression in one round.

Is this always possible? Let's explore.

Standard Approach: Algebra
Precisely mathematically describe the problem or scenario, in symbols. This enables you to use algebra—we teach that in school for a reason.

WLOG let $a \le b \le c$, and let (p_a, p_b, p_c) be a permutation of $(1, 2, 3)$.
After one round, we transform $(a, b, c) \mapsto (a + p_a k,\ b + p_b k,\ c + p_c k)$.

One way these could form an arithmetic progression is if we make it so that

$$(b+p_b k)-(a+p_a k)=(c+p_c k)-(b+p_b k)$$

(i.e. this is the common difference). Solving for k:

$$k=\frac{c-2b+a}{-p_c+2p_b-p_a}$$

Note that $p_a+p_b+p_c=6$, regardless of permutation. So, we can express the above as:

$$k=\frac{(a+b+c)-3b}{-6+3p_b}.$$

(We also rewrote the numerator in a similar evocative manner.)

We must ensure that k is an integer, so we do casework on p_b:

- If $p_b=1$, the denominator is -3.
- If $p_b=2$, the denominator is 0 (bad).
- If $p_b=3$, the denominator is 3.

We can get an integer if and only if the numerator is divisible by 3, which happens if and only if $a+b+c$ is divisible by 3 *(shelve this fact for later)*.

We also must ensure that k is positive, and this is always possible:

- If the numerator is negative, choose $p_b=1$.
- If the numerator is already positive, choose $p_b=3$.

Claim

If $a+b+c$ is divisible by 3, then the task can be done in 2 rounds.

Proof

WLOG let $a \le b \le c$.

If $a+b+c-3b > 0$, choose $(p_a, p_b, p_c) = (1, 3, 2)$ and

$$k := \frac{a+b+c-3b}{3}.$$

If $a+b+c-3b < 0$, choose $(p_a, p_b, p_c) = (2, 1, 3)$ and

$$k := \frac{a+b+c-3b}{-3}.$$

By construction, as shown earlier, this results in an arithmetic progression, which can be made all equal in one more round.

Bonus

As a bonus, complete the proof by proving the following.

- $a+b+c-3b = 0$ if and only if (a, b, c) *already* formed an arithmetic progression (in which case solvable in one move)
- As per the bounds of this problem's checker, show that this construction ensures the k chosen for the first *and second round* are both $\le 10^9$.

Finally, what about the case where $a+b+c$ is not divisible by 3?

Play around with it for a bit, and you might start to get the feeling that this case is impossible. Let's prove it.

Standard Toolbox: Invariants for Impossibility

When trying to prove a task is impossible using some operation, find a property that the operation never changes (this property is called the *invariant*) and show that the starting point and the end goal have different invariants.

Claim

If $a + b + c$ is not divisible by 3, then the task is impossible.

Proof

Note that the sum **modulo 3** is invariant under the hop, skip, jump operation.

$$\begin{aligned}(a + p_a k) + (b + p_b k) + (c + p_c k) &= (a + b + c) + (p_a + p_b + p_c)k \\ &= (a + b + c) + 6k \\ &\equiv a + b + c \pmod 3.\end{aligned}$$

However, note that our end goal is (m, m, m) for some integer m. But here, $m + m + m \equiv 0 \pmod 3$.

So, if $a+b+c$ did not already start at $0 \pmod 3$, it will be impossible to make it so, using the hop, skip, jump operation. Thus, the task is impossible.

Implementation

Implementation for this problem can be annoying because of managing all the different cases. This is in contrast to the natural-language construction of the proof, where we are able to use statements like "without loss of generality, assume $a \leq b \leq c$".

So... implement it that way!

```python
def solve(a, b, c):
    ...
    WLOGger = sorted((x, i) for i, x in enumerate([a, b, c]))
    sorted_values, sorted_indices = zip(*WLOGger)
    if sorted_values != (a, b, c):
        inverse = {
            sorted_indices[i]: i
            for i in range(3)
        }
        return [
            (v, tuple(op[inverse[i]] for i in range(3)))
            for v, op in solve(*sorted_values)
        ]
    # So, if we made it here, a <= b <= c
    ...
```

In my implementation, I have done the following.

- Define a `solve(a, b, c)` function
- If (a, b, c) is not in sorted order, have the function call itself *with* the arguments in the right order.
  - Remember to "translate" the operations back to what they should be, had we not rearranged the elements.
- That way, we only make it past that `if` statement if $a \leq b \leq c$, so it's an assumption we can safely make from that point on (greatly simplifying the logic of the code)