

Reach a Number

You are standing at position `0` on an infinite number line. There is a goal at position `target`.

On each move, you can either go left or right. During the n -th move (starting from 1), you take n steps.

Return the minimum number of steps required to reach the destination.

Example 1:

Input: `target = 3`

Output: `2`

Explanation:

On the first move we step from `0` to `1`.

On the second step we step from `1` to `3`.

Example 2:

Input: `target = 2`

Output: `3`

Explanation:

On the first move we step from `0` to `1`.

On the second move we step from `1` to `-1`.

On the third move we step from `-1` to `2`.

Note:

- `target` will be a non-zero integer in the range `[-109, 109]`.

Solution 1

This didn't feel like an easy to me
written by [molehero](#) original link [here](#)

Solution 2

- We can always take $\text{abs}(\text{target})$, since the axis is symmetric.
- First of all we keep adding $\text{sum}=1+2+\dots+n \geq \text{target}$, solve this quadratic equation gives the smallest n such that $\text{sum} \geq \text{target}$.
- If $1+2+\dots+n = \text{target}$, return n .
- Now we must minus $\text{res} = \text{sum} - \text{target}$. If res is even, we can flip one number x in $[1, n]$ to be $-x$.
- Otherwise if res is odd, and $n+1$ is odd, we can first add $n+1$, then res is even. Next flip an x to be $-x$.
- If res is odd and $n+1$ is even, we add $n+1$ then subtract $n+2$, res becomes even, then flip an x .

```
class Solution {
public:
    int reachNumber(int target) {
        target = abs(target);
        long long n = ceil((-1.0 + sqrt(1+8.0*target)) / 2);
        long long sum = n * (n+1) / 2;
        if (sum == target) return n;
        long long res = sum - target;
        if ((res&1) == 0)
            return n;
        else
            return n+((n&1) ? 2 : 1);
    }
};
```

written by [fsqhxrcjw](#) original link [here](#)

Solution 3

Step 0: Get positive `target` value (`step` to get negative `target` is the same as to get positive value due to symmetry).

Step 1: Find the smallest `step` that the summation from 1 to `step` just exceeds or equals `target`.

Step 2: Find the difference between `sum` and `target`. The goal is to get rid of the difference to reach `target`. For `i`th move, if we switch the right move to the left, the change in summation will be $2*i$ less. Now the difference between `sum` and `target` has to be an even number in order for the math to check out.

Step 2.1: If the difference value is even, we can return the current `step`.

Step 2.2: If the difference value is odd, we need to increase the step until the difference is even (at most 2 more steps needed).

Eg:

`target = 5`

Step 0: `target = 5`.

Step 1: `sum = 1 + 2 + 3 = 6 > 5`, `step = 3`.

Step 2: Difference = `6 - 5 = 1`. Since the difference is an odd value, we will not reach the target by switching any right move to the left. So we increase our `step`.

Step 2.2: We need to increase `step` by 2 to get an even difference (i.e. `1 + 2 + 3 + 4 + 5 = 15`, now `step = 5`, difference = `15 - 5 = 10`). Now that we have an even difference, we can simply switch any move to the left (i.e. change `+` to `-`) as long as the summation of the changed value equals to half of the difference. We can switch 1 and 4 or 2 and 3 or 5.

```
class Solution {
    public int reachNumber(int target) {
        target = Math.abs(target);
        int step = 0;
        int sum = 0;
        while (sum < target) {
            step++;
            sum += step;
        }
        while ((sum - target) % 2 != 0) {
            step++;
            sum += step;
        }
        return step;
    }
}
```

written by [jun1013](#) original link [here](#)

From [LeetCoder](#).