```go
func predictN(goalns int64, prevIters int64, prevns int64, last int64) int {
    if prevns == 0 {
        // Round up to dodge divide by zero. See https://go.dev/issue/70709.
        prevns = 1
    }

    // Order of operations matters.
    // For very fast benchmarks, prevIters ~= prevns.
    // If you divide first, you get 0 or 1,
    // which can hide an order of magnitude in execution time.
    // So multiply first, then divide.
    n := goalns * prevIters / prevns
    // Run more iterations than we think we'll need (1.2x).
    n += n / 5
    // Don't grow too fast in case we had timing errors previously.
    n = min(n, 100*last)
    // Be sure to run at least one more than last time.
    n = max(n, last+1)
    // Don't run more than 1e9 times. (This also keeps n in int range on 32 bit platforms.)
    n = min(n, y...: 1e9)
    return int(n)
}
```

Go's prediction of iterations uses the following:

1) Goal bench duration
2) Previous iterations
3) Previous duration.

A $n := goalns * \frac{prevIters}{prevns}$.

Breaking it down:

$timePerIteration = \frac{prevns}{prevIters}$

① based on previous iterations.

② No. of iterations to be done to have goalns =

$$n = \frac{goalns}{timePerIteration}$$

③ Substituting timePeriteration,

$$n = \frac{goalns}{prevns} \Big/ \frac{}{prevtes}$$

Thus,

$$n = \frac{goalns * prevtes}{prevns}$$

B $nt = n/5$ ⟹ Buffer

Have more iterations than calculated. (1.2x)

C $n = \min(n, last * 100)$

Ensure n is not growing too fast, avoid explosive growth.

D $n = \max(n, last + 1)$

Ensure n is atleast one more than last iteration count

E $n = \min(n, 1e^9)$

Avoid impractically long runs.