

Solution: Lookup Tables and Iteration

Part A: Augmented Table

For simplicity assume that $n = 2^k$, the general case is entirely similar. First assume $t < n$. Using the same approach as in fast exponentiation

$$f^t(x) = f^{t_1}(f^{t_2}(\dots(f^{t_r}(x))\dots))$$

where the t_i are powers of 2 and $t = \sum t_i$, $r \leq k$. Hence we should precompute $f^{2^i}(x)$ for $0 \leq i < k$.

Alas, for arbitrary t this approach fails. The solution is to compute the transient and period for each point. This is easy to do in linear time and linear space; the resulting table has two additional columns. Given transient/period information we can reduce the problem of computing $f^t(x)$ to the problem of computing $f^{t'}(x)$ for some $t' < n$.

Part B: Fast Lookup

Given the precomputed powers-of-two table we can determine $f^t(x)$ in at most k table lookups, perhaps after reducing t as described above.

Part C: Space Requirement

Using a uniform cost function (an integer has size 1) the table has size $O(nk)$.

Part D: Cost Pre-Computation

Initially we are given the table for $f = f^1$. It takes $2n$ lookups in this table to obtain f^2 , another $2n$ lookups in this table to obtain f^4 and so on. In total the cost of precomputing the table is $O(nk)$, even if we have to compute transients/periods.

Solution: UnCollatz

Part A: Convergence

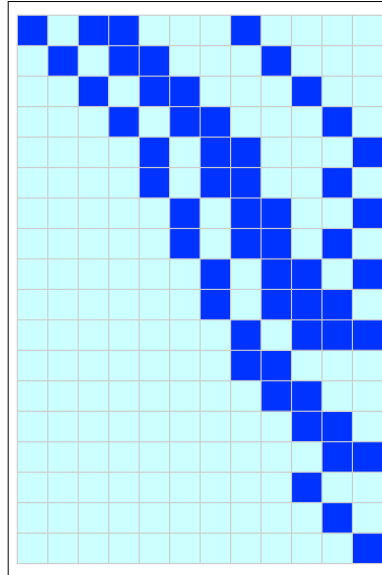
For x even we have $U(x) < x$ and for x odd, $x \neq 1$, we have $U(U(x)) < x$. Hence the orbit of any point $x > 1$ contains a point less than x . Done by induction.

Part B: Characterization

Unsurprisingly the binary expansion of x is the key for a compact description of the transient lengths. Since trailing 0's are simply removed in as many steps as there are 0's it is natural to write the input in terms of alternating blocks of 1's and 0's:

$$1^{a_k}0^{b_k}1^{a_{k-1}}0^{b_{k-1}} \dots 1^{a_1}0^{b_1}1^{a_0}0^{b_0}$$

Here $a_i, b_i > 0$ except possibly for b_0 . A typical orbit in binary looks like so:



It is easy to deal with the special case $k = 0$: the stopping time in this case is $a_0 + b_0 - 1 + 2[a_0 > 1]$. Let's suppose the trailing 0's have been removed. The question then is how many steps are required to remove a trailing segment of the form $0^b 1^a$, $a, b \geq 1$. From the definition of U , after $2(b - 1) + a + 1$ the segment is replaced by a single 1 (which is attached to the next block of 1's). Adding up all the steps produces a transient length of

$$\sum a_i + 2 \sum b_i - b_0 + 2$$

except in the special case $k = 0$.

The expression can be simplified a little bit by writing the sums in terms of $|x|$, the length of x , and $|x|_0$, the number of 0's in the binary expansion, minus the number of trailing 0's (of course there are no leading 0's):

$$\sigma(x) = |x| - |x|_0 + 1 - 2[\mathbf{ds}(x) = 1]$$

For those who mistrust the argument above it is a healthy exercise to verify the last characterization by induction.

Part C: Distribution

From our characterization of σ it is easy to see that, for $k + 1$ -bit numbers, the minimum value is $\sigma(2^k) = k$. Somewhat less clear is the distribution of the other values.

Claim: Every stopping time value $k + 2 + \ell$, $0 \leq \ell < k$, appears $\binom{k}{\ell+1}$ times.

To see this it suffices to show that the number of binary expansions of $k + 1$ -bit numbers, excluding 2^k , that have ℓ internal 0's is $\binom{k}{\ell+1}$. To see this, consider $\ell + 1$ markers, to be placed on the last k positions in the binary expansion. The first ℓ markers indicate the positions of the internal 0's and the last one indicates the position of the last 1. Done.

Solution: Floyd's Cycle Detection Trick

Part A: Correctness

Think of adding a step counter s which is initialized to 1 before the loop and is stepped every time `fast` is moved ahead. Let t be the transient of the orbit and p its period. Write β for the value of `bnd`. We use 0-indexing to number the items on the orbit.

First note that the fast pebble simply moves at speed 1. Let's refer to a sequence of moves in which `bnd` is constant as a round. In the first few rounds, as long as $s < t$, both pebbles move along the transient part of the orbit. In the next round, the fast pebble enters the limit cycle. At the end of that round, the slow pebble is moved onto the limit cycle. The algorithm continues until $\beta \geq p$. Note that in the last round the fast pebble simply travels once around the limit cycle so that `per` is indeed the length of that cycle.

Part B: Running Time

We claim that the positions that the slow pebble occupies are $0, 1, 3, \dots, 2^k - 1$ where $k = \lceil \lg \max(t, p) \rceil$. For example, when $t \geq p$ then $2^{k-1} - 1 < t \leq 2^k - 1$, so that round k places the slow pebble on the limit circle. Then one more round is required to termination.

From the claim it follows that the number of steps in the teleportation algorithm is $\Theta(t + p)$.

Part C: Running Time

From the previous result, both complete algorithms have the same behavior asymptotically, so we need to count steps a bit more carefully.

We claim that the classical algorithm always uses more applications of f than the teleportation algorithm.

Floyd's algorithm requires $s + 2(t + p)$ steps where s accounts for the initial phase of the algorithm that determines a point on the limit cycle. The Teleportation algorithm requires $s' + 2t + p$ steps where s' accounts for the initial phase of the algorithm that determines a point on the limit cycle together with the period.

With a little bit of effort one can show

$$s = \begin{cases} 3p & \text{if } t < p, \\ 3(t + (-t \bmod p)) & \text{otherwise.} \end{cases}$$

whereas

$$s' = 2^{\lceil \lg \max(t+1, p) \rceil}$$

Hence Teleportation is always better.