

**Problem 1: Lookup Tables and Iteration** (20 pts.)**Background**

Suppose a function  $f : [n] \rightarrow [n]$  is given as a lookup table, say, a plain array of integers. Assume that  $n$  is machine-sized so that  $f(x)$  can be determined in  $O(1)$  time. In order to determine  $f^t(x)$  we can use repeated lookup. However, if we need to determine  $f^t(x)$  repeatedly it is better to perform a pre-computation that augments the table for  $f$ . The enlarged table then allows for speedy lookups of iterated values of the function  $f$ .

**Task**

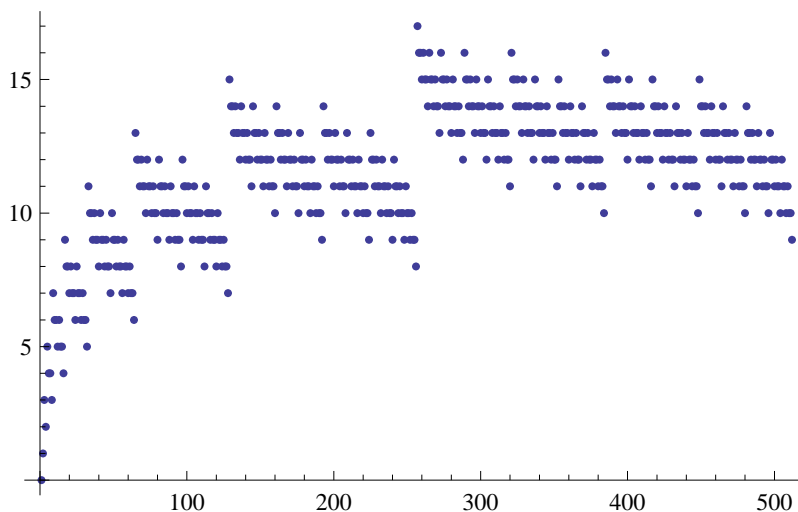
- A. Explain how to augment the table for  $f$  so that lookups of  $f^t(x)$  are fast.
- B. How long does it take in your solution to determine  $f^t(x)$ ?
- C. What is the memory requirement for your table?
- D. What is the cost of the pre-computation?

**Problem 2: UnCollatz** (40 pts.)**Background**

Here is a function on the positive integers that looks quite similar to the infamous Collatz  $C$  function.

$$U(x) = \begin{cases} 1 & \text{if } x = 1, \\ x/2 & \text{if } x \text{ even,} \\ x + 1 & \text{otherwise.} \end{cases}$$

Unlike with the Collatz function, though, it is easy to show that all orbits of  $U$  lead to the fixed point 1. Hence we can define the stopping time  $\sigma(x)$  of  $x$  to be the number of steps  $U$  takes to reach the fixed point. A plot of the first 512 values is below. Note the nice fractal structure.

**Task**

- Show that any orbit of  $U$  ends in the fixed point 1.
- Give as simple a description of the stopping time as you can manage.
- Describe the distribution of stopping times for all  $k$ bit numbers.

**Comment**

For the second and third part some experimentation is probably helpful; it's a bit tricky to get a completely correct answer.

**Problem 3: Floyd and Teleportation** (40 pts.)**Background**

Recall Floyd's trick to determine the transient and period of a finite orbit without using extra memory: two pebbles move at speeds 1 and 2, respectively. Here is a modification of this approach where the "slow" pebble does not move at all, except that it occasionally teleports to the location of the "fast" pebble.

Here is the crucial first part of the algorithm that computes the period of the orbit. The function in question is  $f$  and the starting point is  $a$ .

```
slow = a;
fast = f(a);
per = bnd = 1;
while( fast != slow )
    if( per == bnd )
        { slow = fast; per = 0; bnd *= 2; }
    fast = f(fast);
    per++;

return per;
```

The claim is that this algorithm finds a point on the limit cycle and returns the period. In a separate procedure one can then compute the transient (just as in Floyd's method).

**Task**

- A. Prove that upon termination the pebbles are located on some position on the limit cycle. Conclude that the return value is indeed the period of the orbit.
- B. Determine the time complexity of the teleportation algorithm (i.e., count the number of times the loop is executed).
- C. Now consider the task of computing both transient and period. Compare the performance of Floyd's classical algorithm and the teleportation algorithm. Is one better than the other? Why?