

Introduction

Does “Complexity of an Algorithm” make sense, or is it an abuse of language? The *cost* of an algorithm makes sense; cost can be measured in time consumed, memory occupancy, arithmetic operations performed, memory movements, The complexity of a *task* makes sense; it is the minimum, among all algorithms that perform the task, of these algorithms’ costs. This use of “complexity” appeared in the 1960s in the works of Richard Karp and of Shmuel Winograd, and spread rapidly to other computer scientists. In the 1970s a few mathematicians misapplied “complexity” to their determinations of the least costly algorithms in families of algorithms without establishing whether some families other than the ones they studied might perform the same tasks at lower costs. By the late 1980s “complexity” had become a synonym for “cost” in some circles. Kenneth Rosen, the author of the course’s text *Discrete Mathematics and its Applications* (4th ed., 1998), uses “complexity” this way in his Chapter 2.

When this kind of abuse of language gains currency it obscures distinctions that really do make a difference. Lest these distinctions be lost altogether, examples will be presented here of tasks and algorithms whose complexities and costs are interesting enough to persuade the student (and other readers, I hope) that “cost” deserves to be distinguished from “complexity”.

The first example is the factorization of an integer, for which task simple algorithms cost far more than complicated ones. The second example is the evaluation of a quartic polynomial, a task whose complexity is slightly less than the costs of the most widely used algorithms. The third example is Collatz’s $3x + 1$ puzzle, whose cost is small and complexity unknown. The fourth example is Floyd’s algorithm for finding the ultimate period of an iteration; cost and complexity are about the same. The fifth example is M. Brown’s recurrence whose complexity is about nine times its cost. The sixth example is a modification of Brown’s recurrence that costs only slightly more but behaves with chaotic complexity.

Factoring Integers

Finding the large prime factors of a very big integer n , presented as a string of perhaps several hundred digits, is a task that codebreakers would like to solve very quickly for reasons discussed in the text at the end of Ch. 2.5 . The simplest algorithm known forms successive quotients $n/2, n/3, n/5, n/7, n/9, n/11, n/13, n/15, \dots, n/(\text{consecutive odd integers}), \dots$ until one of these trial divisions goes without remainder, whereupon the smallest prime divisor of n has been found. (Can you see why it must be prime?) Because a nonprime n must have at least one prime divisor no bigger than \sqrt{n} , the number of trial divisions need never exceed roughly $(\sqrt{n})/2$; and if $n = p \cdot q$ for nearly equal primes p and q the number of trial divisions will not fall far short of $(\sqrt{n})/2$, so this is a rough but fair estimate of this simple algorithm’s cost in worst cases. An almost-simplest algorithm forms successive quotients $n/2, n/3, n/5, n/7, n/11, n/13, n/17, n/19, n/23, n/29, \dots, n/(\text{consecutive odd integers not divisible by } 3, 5 \text{ nor } 7), \dots$ until one of these trial divisions goes without remainder, whereupon n ’s smallest prime divisor has been found.

Roughly how much faster than the simplest algorithm is this one? Can you program it to run efficiently?

In case the smallest prime factor of n is not much smaller than \sqrt{n} , the foregoing algorithm can be sped up by interleaving it with Fermat’s factorization algorithm: For $k = \lceil \sqrt{n} \rceil, \lceil \sqrt{n} \rceil + 1, \lceil \sqrt{n} \rceil + 2, \dots$ in turn test whether $m := \sqrt{(k^2 - n)}$ is an integer, in which case $n = (k-m) \cdot (k+m)$. Why does this algorithm always terminate?

In any event, these simple algorithms cost $\Omega(\sqrt{n})$ operations in worst cases, which is intolerable when the integer n is over a hundred decimal digits long. However, there are horrendously complicated algorithms that run far faster, with worst-case costs conjectured to be as small as $O(\exp(2 \cdot (\ln(n))^{1/3} \cdot (\ln(\ln(n)))^{2/3}))$; some of them are accessible through automated algebra systems like *Macysma*, *Maple*, *Mathematica*,

How does \sqrt{n} compare with $\exp(2 \cdot (\ln(n))^{1/3} \cdot (\ln(\ln(n)))^{2/3})$ when $n \approx 10^{10}$? 10^{100} ? 10^{1000} ?

The assignment of greater “complexity” to the simplest algorithms because they run so much slower than than the horrendously complicated algorithms is an assignment that affronts common and other senses.

But the complexity of integer factorization is a concept that makes sense even if, as seems likely, no single factorization algorithm minimizes the worst-case cost for all integers n . Conceivably an infinite collection of factorization algorithms may exist, each algorithm faster than all others for a relatively widely scattered set of integers n , but none faster than *all* others for *all* integers n . In this case, if there is a function $f(n)$ such that, for every tiny $\mu > 0$, an algorithm exists whose cost is $O(f(n)n^\mu)$ but none exists whose cost is $O(f(n))$, this $f(n)$ can reasonably be regarded as *the* complexity of integer factorization. But no such f is known yet.

Suppose you discovered an algorithm that factorizes huge integers faster by orders of magnitude than anyone had thought possible before, thereby undermining the security of widely used encryptions. Which of the following actions would you pursue?

- 1• Publish your algorithm in a mathematics or computer science journal for all the world to see and use.
- 2• Embed your algorithm in a computer program and patent that, and collect royalties from licenced users.
- 3• Keep your algorithm secret and sell it to the Mafia for a billion dollars.

(Action 2• is not so different from 1• unless you can afford to hire lots of lawyers and expert witnesses to defend your patent against infringers. Action 3• may place your life in jeopardy.)

Quicker Quartic Polynomials

Given real numerical coefficients a, b, c, d, e of a quartic polynomial

$$q(x) := a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \quad \text{with } a \neq 0,$$

we seek a formula by which $q(x)$ may be computed for a huge number N of different real numerical arguments x with as few multiplications as possible. Can such a formula run faster than the usual procedure,

$$\text{Horner's Recurrence: } q(x) = (((a \cdot x + b) \cdot x + c) \cdot x + d) \cdot x + e,$$

which costs four multiplications per argument x ? What is the complexity (minimum cost), measured in multiplications per argument x , of quartic polynomial evaluation?

These questions seemed important back in the days when multiplication took appreciably longer than addition, recording and recalling. Nowadays this is the case only for multiplication of multi-word operands, which is what we shall assume. And we shall lump subtractions in with additions. We shall assume also that rounding errors are negligible although in practice they can vitiate much of what we shall do.

The answers to our questions depend upon technicalities. For instance multiplications by small integers are free if they are replaced by additions; $4 \cdot Z = (Z+Z)+(Z+Z)$ takes two additions. Consequently quartics like $k_4 \cdot (x - \zeta_1)^4 + k_2 \cdot (x - \zeta_1)^2 + \zeta_2$, in which k_4 and k_2 are small integers and ζ_1 and ζ_2 are real constants, cost only the two multiplications per argument x required to compute $(x - \zeta_1)^2$ and $(x - \zeta_1)^4$. But there are other quartics that cost more than two multiplications per argument x unless these are evenly spaced. In the special case that every argument $x_j := x_0 + j \cdot \beta$ for $j = 0, 1, 2, \dots, N-1$ and real constants x_0 and $\beta \neq 0$, the cost of computing all N values $q(x_j)$ can be reduced to a few multiplications and a few more than $4N$ additions, as if the complexity (measured in multiplications per argument) of polynomial evaluation were practically zero. To see how to handle this special case, look up "Finite Differences" in a Numerical Analysis text. We shall exclude this special case despite its importance for fast graphic displays of curves.

A family of formulas that cost three multiplications per argument x will be exhibited. This family is complicated, though less so than some other published families; still, the formulas presented below are complicated enough to challenge reader who wonder if they are correct.

First a slight simplification. We shall construct first a formula to calculate a monic quartic

$$F(B,C,D,E; z) := z^4 + B \cdot z^3 + C \cdot z^2 + D \cdot z + E$$

with only two multiplications per argument z . ("Monic" means that the leading coefficient is 1.) Then we can calculate any given non-monic quartic $q(x)$ with three multiplications per argument x in either of two ways:

$$q(x) = a \cdot F(b/a, c/a, d/a, e/a; x), \quad \text{or} \quad q(x) = \text{sign}(a) \cdot F(b/h^3, c/h^2, d/h, e; x \cdot h)$$

where $h = |a|^{1/4}$. There are many formulas that compute F with two multiplications; a family of candidates is provided by

$$G_{ijklm}(U,V,W; z) := (z^2 + iz + U) \cdot (z^2 + jz + V) + kz^2 + mz + W = G_{jikm}(V,U,W; z),$$

which produces a monic quartic polynomial in z with just two multiplications provided i, j, k, m are all small integers drawn, perhaps, from the set $\{-1, 0, 1\}$. For almost every choice of those integers we get a formula to compute F with two multiplications after the constants U, V, W and some others have been assigned appropriate values as follows:

Choose small integers i, j, k, m with $i \neq j$;

$$T := (B - i - j)/4; \quad S := D - 2CT + (8T + 3(i+j))T^2 - m;$$

$$R := C - 3T(2T+i+j) - ij - k; \quad U := (iR - S)/(i-j);$$

$$V := R - U; \quad W := E - G_{ijklm}(U,V,0; T);$$

and then

$$F(B,C,D,E; z) = G_{ijklm}(U,V,W; z+T).$$

The reader, perhaps aided by a computerized algebra system, is urged to confirm the correctness of these formulas.

Finally, in one of the two ways mentioned above, $F(\dots)$ generates a formula for $q(x)$ that costs only three multiplications per argument x , not the four that Horner's recurrence costs. To attribute greater "complexity" to Horner's four multiplications than to the three needed by the formula generated by $F(\dots)$ and $G(\dots)$ above seems perverse.

What is the complexity of quartic polynomial evaluation? The formulas generated by $F(\dots)$ and $G(\dots)$ bring the cost down to three multiplications per argument but do not prove this cost to be minimal, though it is minimal. Real quartics that cannot be evaluated in so few as two multiplications do exist, but the proof of their existence is subtle; it goes roughly like this: All the polynomials that can be evaluated in two multiplications and arbitrarily many additions of expressions made up from real constants and the indeterminate x can be shown to constitute a countable set of curves and surfaces of dimensions less than five in the five-dimensional space of real quartics like $q(x) = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$. Therefore, the complexity of quartic polynomial evaluation is, with rare exceptions, three multiplications per argument.

Collatz's $3x + 1$ Puzzle

Let $\zeta(x_0)$ be the least integer n for which $x_n \leq 1$ when, for $k = 0, 1, 2, 3, \dots$ in turn,

$$x_{k+1} := \begin{cases} x_k/2 & \text{whenever } x_k \text{ is even,} \\ 3 \cdot x_k + 1 & \text{whenever } x_k \text{ is odd,} \end{cases}$$

starting from any positive integer x_0 . There is a puzzle here: nobody knows for sure why such an n exists for every positive integer x_0 , nor has anybody found an x_0 for which $\zeta(x_0)$ must be deemed infinite. Among the first 29 values of $\zeta(x_0)$, the size of $\zeta(27)$ is a surprise:

x_0 :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$\zeta(x_0)$:	0	1	7	2	5	8	16	3	19	6	14	9	9	17	17	4	12	20	20	7	7	15	15	10	23	10	111	18	18

This puzzle has been traced back to the 1920s when it was described by the late Prof. Lothar Collatz of Hamburg. The puzzle has also been attributed to the late Dr. Stanislaw Ulam of the Lawrence Los Alamos National Laboratory, among others. The cost of computing $\zeta(x_0)$ the obvious way is proportional to $\zeta(x_0)$, but other ways exist to compute it at a cost usually small compared with $\zeta(x_0)$ when it is big; see program `collatz` in the NUMTHEORY section of the SHARE library of *Maple V*. Since nobody knows for sure whether $\zeta(\dots)$ is always finite, nobody knows its complexity, though the obvious algorithm exhibited above is extremely simple.

Ultimately Periodic Sequences

Given a function f that maps a set X into or onto itself, and a member x_0 of X , a sequence $(x_0, x_1, x_2, \dots, x_n, \dots)$ is generated by the recurrence $x_{n+1} := f(x_n)$ for $n = 0, 1, 2, \dots$ in turn. The sequence is *Ultimately Periodic* when $x_n = x_{n-p}$ for some fixed $p > 0$ and all sufficiently large n , and then p is *an Ultimate Period* of the sequence. For example, all sequences generated so far for Collatz's puzzle have ultimate period $p = 3$ because they have always turned into $(x_0, x_1, \dots, 1, 4, 2, 1, 4, 2, 1, \dots)$.

Every positive integer multiple of an ultimate period p is another ultimate period; the smallest ultimate period, the greatest common divisor of all ultimate periods, is called "*The Ultimate Period*" of the sequence. Given f and x_0 , is there an algorithm that will detect the ultimate period p if one exists? (If no period exists the algorithm may run forever or until stopped by

an external agency.) A naive algorithm stores all of $(x_0, x_1, x_2, \dots, x_n = f(x_{n-1}))$ and compares each x_n with every previous member of the sequence until a match with x_{n-p} occurs. This naive algorithm costs $\Theta(N)$ memory and $\Theta(N^2)$ time for comparisons, where N is the least n for which $x_n = x_{n-p}$. There is a less costly way.

An algorithm attributed to Prof. Robert Floyd of Stanford computes simultaneously

$$x_n := f(x_{n-1}) \quad \text{and} \quad x_{2n} := f(f(x_{2n-2})) \quad \text{for } n = 1, 2, 3, \dots \text{ in turn}$$

and compares them until $x_m = x_{2m}$, at which point m must be a multiple of the ultimate period p ; then $x_{m+1}, x_{m+2}, \dots, x_{m+p}$ are recomputed and compared with x_m to determine p . This algorithm costs a fixed amount of memory and $\Theta(N)$ time; can you see why? Refinements of this algorithm exploit available memory to reduce the time spent when N or p is not too big, but as $N \rightarrow \infty$ every algorithm that detects the ultimate period must cost $\Omega(N)$ time; can you see why? Therefore the complexity of detecting the ultimate period is $\Theta(N)$ for each f and x_0 .

M. Brown's Recurrence

Prof. Morton Brown of the Univ. of Michigan presented this problem in 1983 and its solution in "A periodic homeomorphism of the plane" on pp. 83-87 of *Continuum Theory and Dynamical Systems*, Lect. Notes in Pure & Appl. Math. #149 (1993) Dekker, NY.

Prove that the sequence $(x_0, x_1, x_2, x_3, \dots)$, defined by the recurrence

$$x_{n+1} := |x_n| - x_{n-1} \quad \text{for } n = 1, 2, 3, \dots \text{ in turn}$$

started from any real x_0 and x_1 not both zero, is periodic with period 9.

Experiments corroborate (but cannot prove) the claim. For example ...

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	$x_9 = x_0$	$x_{10} = x_1$
-1	0	1	1	0	-1	1	2	1	-1	0
3	5	2	-3	1	4	3	-1	-2	3	5
π	1	$1-\pi$	$\pi-2$	$2\pi-3$	$\pi-1$	$2-\pi$	-1	$\pi-1$	π	1

That so simple a recurrence has a period so big as 9 is surprising. More surprising is the period's persistence in the face of roundoff; when executed in floating-point arithmetic (every subtraction is rounded off) the sequence is still ultimately periodic with ultimate period 9. (Evidently roundoff is not so nearly random as some people think.) Even more surprising is the length of the proof that the recurrence behaves as it does. To keep that length within bounds we shall assume all real arithmetic to be exact (no roundoff).

To prove the claim we identify consecutive pairs (x_n, x_{n-1}) in the sequence with points in the Cartesian (x, y) plane and abbreviate $(x, y) =: \mathbf{z}$ and $(x_{n+1}, x_n) =: \mathbf{z}_n$. Then the given recurrence takes the form of an iteration that starts from an arbitrary \mathbf{z}_0 and continues

$$\mathbf{z}_n = H(\mathbf{z}_{n-1}) \quad \text{for } n = 1, 2, 3, \dots$$

in which $H(\mathbf{z}) = H((x, y))$ is defined by

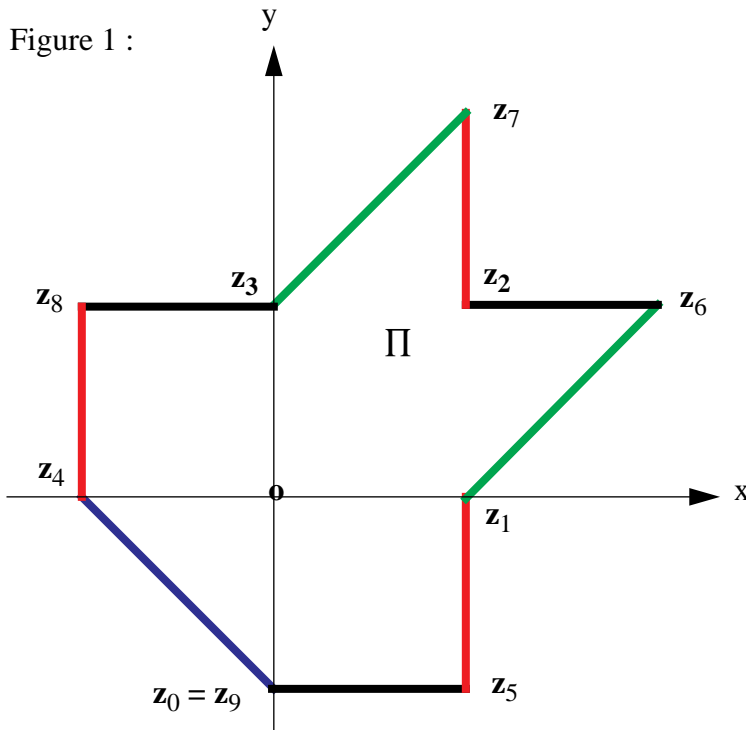
$$H((x, y)) := (|x| - y, x).$$

An example, corresponding to the first instance in the table above, is ...

- | | |
|------------------|-----------------------|
| $z_0 := (0, -1)$ | $z_5 = (1, -1)$ |
| $z_1 := (1, 0)$ | $z_6 = (2, 1)$ |
| $z_2 := (1, 1)$ | $z_7 = (1, 2)$ |
| $z_3 := (0, 1)$ | $z_8 = (-1, 1)$ |
| $z_4 := (-1, 0)$ | $z_9 = (0, -1) = z_0$ |

Our task is to prove that $H^{[9]}(z) := H(H(H(H(H(H(H(H(H(z)))))))) = z$ for every point z . Our task appears to be simply a matter of repeated substitution and simplification, but actually it is complicated enough that no automated algebra system has been able to perform it without human assistance. (*DERIVE* has required by far the least help.) The following proof is designed for human consumption.

Set the origin $o := (0, 0)$. For any $\mu \geq 0$ and any $z = (x, y)$ let $\mu z := (\mu x, \mu y)$. Provided $z \neq o$, as μ runs through all nonnegative values μz traces a ray, a semi-infinite straight line segment emanating from o and passing through and beyond z . Since $H(\mu z) = \mu H(z)$, if “ $H^{[9]}(z) = z$ ” is true at any one point $z \neq o$ on a ray it is true at every point μz on that ray. The one point we shall choose is the ray’s intersection with a 9-sided polygon Π contrived to enclose o in its interior. The vertices of Π , in order, are the ten points $z_0, z_5, z_1, z_6, z_2, z_7, z_3, z_8, z_4, z_9 = z_0$ listed above. z_0 was chosen because it and z_3 lie on the two vertical rays emanating from o ; these rays are the *singularities* of H where its derivative does not exist.



This polygon Π turns out to be “invariant” in the sense that $H(\Pi) = \Pi$; each application of H maps one edge of Π to another, and nine applications rotate Π twice counter-clockwise. Since no edge crosses the vertical rays, the action of H upon each edge takes one of the forms

$$H((x, y)) = (x - y, x) \quad \text{on an edge throughout which } x \geq 0,$$

$$H((x, y)) = (-x - y, x) \quad \text{on an edge throughout which } x \leq 0,$$

both free of the $|\dots|$ operator. This reduces our task to simple algebra. On each edge $\mathbf{z}_i\mathbf{z}_j$, where $j := i+5 \pmod 9$, we label its points with a parametric formula $\mathbf{Z}_i(t) := (X_i(t), Y_i(t))$ whose parameter t runs from 0 to 1 as $\mathbf{Z}_i(t)$ runs along the edge from \mathbf{z}_i to \mathbf{z}_j . Of course each $\mathbf{Z}_{i+1}(t) := H(\mathbf{Z}_i(t))$. The formulas are tabulated below and show $\mathbf{Z}_9(t) = \mathbf{Z}_0(t)$, thus proving that Morton Brown's recurrence has period 9.

i	\mathbf{z}_i	$\text{sign}(X_i(t))$	$X_i(t)$	$Y_i(t)$	\mathbf{z}_j
0	(0, -1)	+	t	-1	\mathbf{z}_5
1	(1, 0)	+	$t+1$	t	\mathbf{z}_6
2	(1, 1)	+	1	$t+1$	\mathbf{z}_7
3	(0, 1)	-	$-t$	1	\mathbf{z}_8
4	(-1, 0)	-	$t-1$	$-t$	\mathbf{z}_0
5	(1, -1)	+	1	$t-1$	\mathbf{z}_1
6	(2, 1)	+	$2-t$	1	\mathbf{z}_2
7	(1, 2)	+	$1-t$	$2-t$	\mathbf{z}_3
8	(-1, 1)	-	-1	$1-t$	\mathbf{z}_4
9	(0, -1)	+	t	-1	\mathbf{z}_5

Let's construe his recurrence $x_{n+1} := |x_n| - x_{n-1}$ as an algorithm; what are its cost and, if different, its complexity? Both appear now to be modest even if the proof of period 9 is deemed lengthy. However, changing Brown's recurrence in a way that increases its cost very little can increase enormously its complexity by any reasonable measure, as we shall see next.

Another Recurrence Somewhat Like Brown's

Set constant $K := +1$, or else set $K := -1$. For any real initial values x_0 and x_1 let

$$x_{n+1} := |x_n + K| - |K| - x_{n-1} \text{ for } n = 1, 2, 3, \dots$$

There is good reason to believe that the infinite sequence $(x_0, x_1, x_2, x_3, \dots)$ is *bounded*; this means that some function $F(K, x_0, x_1) > |x_n|$ for all n . Although a proof of boundedness has not yet been published, we shall take it for granted in what follows. When the recurrence is implemented in floating-point arithmetic with roundoff the sequence turns out always to be ultimately periodic albeit sometimes with a gargantuan ultimate period. Henceforth we shall assume arithmetic is exact, no roundoff, in which case the sequence must be either aperiodic or, because the recurrence is reversible, periodic; the sequence cannot be merely ultimately periodic. If x_0 and x_1 are rational the sequence must be rational and, if bounded, periodic; but the sequence can be irrational and periodic. If the period exists it must depend upon K , x_0 and x_1 . How? The dependence appears to be extremely complicated.

As before we identify consecutive pairs (x_n, x_{n-1}) in the sequence with points in the Cartesian (x, y) plane and abbreviate $(x, y) =: \mathbf{z}$ and $(x_{n+1}, x_n) =: \mathbf{z}_n$. Then the given recurrence takes the form of an iteration that starts from an arbitrary \mathbf{z}_0 and continues

$$\mathbf{z}_n = H_K(\mathbf{z}_{n-1}) \quad \text{for } n = 1, 2, 3, \dots$$

in which $H_K(\mathbf{z}) = H_K((x, y))$ is defined by

$$H_K((x, y)) := (|x+K|-|K|-y, x).$$

The map $\mathbf{z} \rightarrow H_K(\mathbf{z})$ is an *Area-Preserving* map, but this fact has not yet borne much fruit.

The fixed-points $\mathbf{z} = H_K(\mathbf{z})$ are the points of period 1. The only fixed-point of H_K is $\mathbf{z} = \mathbf{0}$. Other starting points \mathbf{z}_0 yield different periods or none. The existence has been proved of starting points \mathbf{z}_0 from which the sequence $(\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \dots)$ is aperiodic but no such starting point is known yet; perhaps $\mathbf{z}_0 = (\sqrt{2}, \sqrt{3})$ is one for both $K = \pm 1$.

There are closed polygonal domains in the \mathbf{z} -plane from which, when started there, the iteration provably produces periodic sequences. For instance, ...

When $K = +1$:

- If ever $|x_n| \leq 1$ and $|x_{n+1}| \leq 1$ and $|x_{n+1}-x_n| \leq 1$, these inequalities persist and the period is 6 except if $\mathbf{z}_0 = \mathbf{0}$ the period is 1.
- If $-3 \leq x_0 \leq -1$, $-3 \leq x_1 \leq 1$ and $-5 \leq x_0+x_1 \leq -3$, the period is 30 except when $x_0 = x_1 = -2$ the period is 5. In particular, if $\mathbf{z}_0 = (\pm\sqrt{2}, -\sqrt{3})$ or if $\mathbf{z}_0 = (-\sqrt{2}, \sqrt{3})$ the period is 30. But if $\mathbf{z}_0 = (\sqrt{2}, \sqrt{3})$ the period is huge if there is a period; see Figure 2.
- If $|x_0-4| \leq 1/2$, $|x_1-4| \leq 1/2$ and $|x_0-x_1| \leq 1/2$, the period is 114 except if $\mathbf{z}_0 = (4, 4)$ the period is 19; if $(5, 5)$, 33; if $(6, 6)$, 14.

When $K = -1$:

- If ever $x_n \leq -1$ and $x_{n+1} \leq 1$ and $-1 \leq x_{n+1}+x_n$, these inequalities persist and the period is 3 except if $\mathbf{z}_0 = \mathbf{0}$ the period is 1.
- If $-7 \leq x_0 \leq -1$, $1 \leq x_1 \leq 7$ and $-7 \leq x_0+x_1 \leq -1$, the period is 24 except when $x_0 = -8$ and $x_1 = 4$ the period is 4.
- If $|x_0+x_1-20| \leq -3$, $|2x_0-x_1-10| \leq -3$ and $|x_0-2x_1+10| \leq -3$, the period is 102 except when $x_0 = x_1 = 10$ the period is 17.
- If $\mathbf{z}_0 = (\sqrt{2}, \sqrt{3})$ the period is huge if there is a period; see Figure 3.
- If $\mathbf{z}_0 = (4, 4)$ the period is 4; if $(5, 5)$, 54; if $(6, 6)$, 25.

All claimed periods were confirmed with the aid of the computerized algebra system *DERIVE*.

Is there, for every integer $p > 2$, at least one $K = \pm 1$ and at least one starting point from which the period is p ? Given an aperiodic sequence generated by our recurrence, is there a different recurrence that generates the same sequence? Evidently our recurrence costs little to run but behaves in a complicated way that raises innumerable questions. If our recurrence is construed as an algorithm, to which of its cost or behavior should "the algorithm's complexity" refer?

(In Figures 2 and 3 below, pairs (x, y) are plotted in the complex plane as $x + yi$.)

Figure 2 :

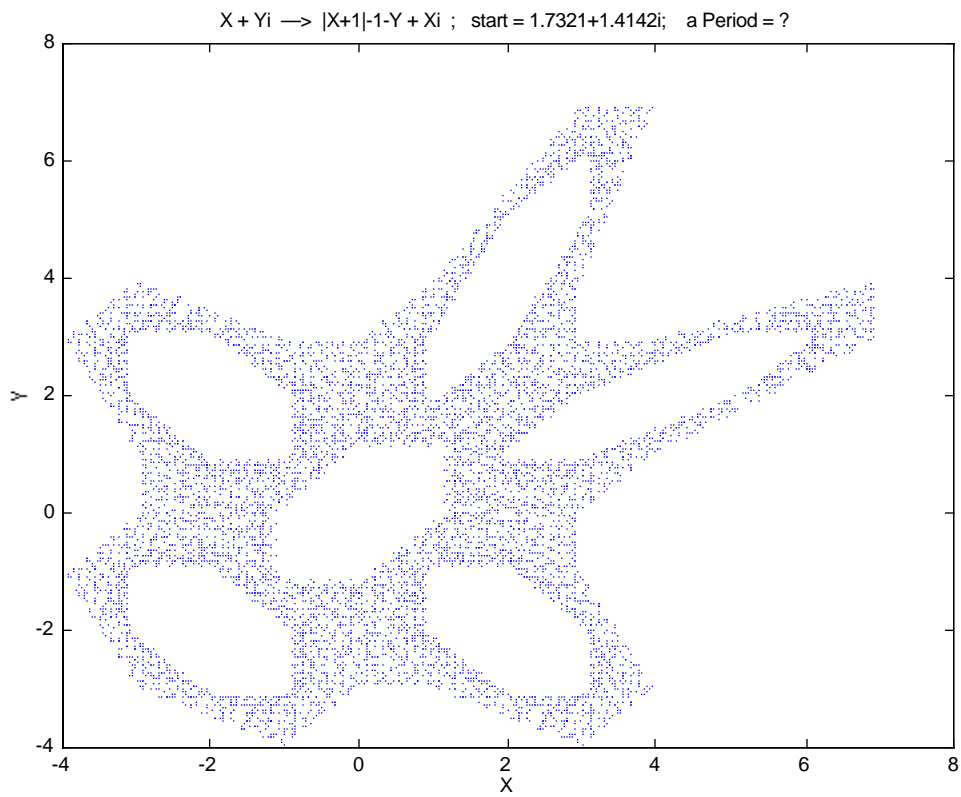
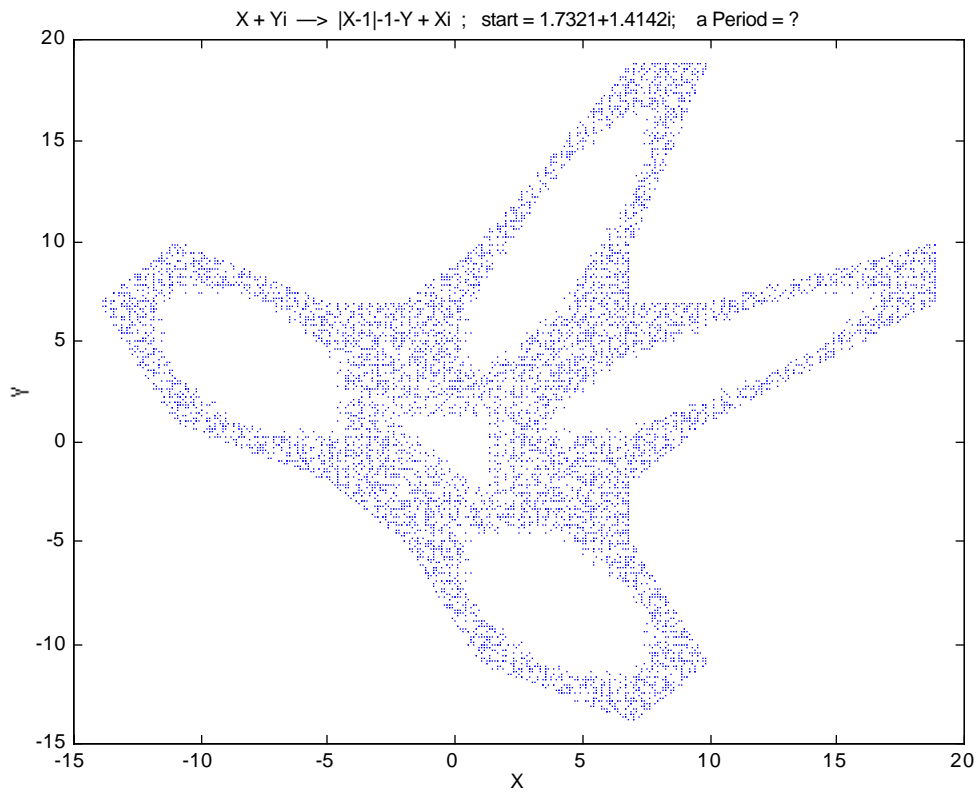


Figure 3 :



Conclusion

The cost of applying an algorithm deserves to be distinguished from its complexity. The word “complexity” may pertain better to the algorithm’s description or to its correctness proof or to its behavior than to its cost.

Advances in the mathematical sciences, I was taught, are esteemed to the extent that they allow us to understand more while obliging us to memorize less. As scientists advance, however, they sometimes substitute four-syllable words like “complexity” for four-letter words like “cost” and “work” of humbler French and Anglo-Saxon origin. Such substitutions do not strike me as advances in Science.