ED 341 562 SE 052 490

AUTHOR	Hastings, Harold M.; And Others
TITLE	The Computer as an Experimental Tool in Teaching Mathematics. Dissemination Packet--Summer 1989: Booklet #9.
INSTITUTION	Hofstra Univ., Hempstead, NY. Dept. of Mathematics.; Hofstra Univ., Hempstead, NY. School of Secondary Education.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	89
CONTRACT	TEI8550088,8741127
NOTE	54p.; For related documents, see SE 052 482-489. Pages 46-54 contain light print.
PUB TYPE	Guides - Classroom Use - Teaching Guides (For Teacher) (052) -- Computer Programs (101)
EDRS PRICE	MF01/PC03 Plus Postage.
DESCRIPTORS	*Computer Assisted Instruction; Computer Software Evaluation; *Discovery Learning; Higher Education; High Schools; *Inservice Teacher Education; *Mathematics Education; *Mathematics Teachers; Program Descriptions; Secondary School Mathematics; Secondary School Teachers; Teacher Education Programs; Teacher Workshops
IDENTIFIERS	*Hofstra University NY; *PASCAL Programing Language

ABSTRACT

This booklet is the ninth in a series of nine from the Teacher Training Institute at Hofstra University (New York) and describes the content and the approach of an institute course in which the participants use the personal computer as a personal tool within the mathematical discovery process of making conjectures, testing those conjectures, and verifying results and/or retesting. Pedagogical commentary and appropriate Pascal programs are presented for four topics, including: (1) iterated quadratic maps; (2) fractal geometry with applications; (3) conditional probability as applied to baseball league trends and results; and (4) recursion and induction in sorting routines. (JJK)

HOFSTRA UNIVERSITY

TEACHER TRAINING INSTITUTE

Department of Mathematics and School of Secondary Education
Hofstra University
Hempstead, NY 11550

DISSEMINATION PACKET – SUMMER 1989

Booklet #9

HAROLD M. HASTINGS,
JOYCE BERNSTEIN, and ROBERT SILVERSTONE

THE COMPUTER AS AN EXPERIMENTAL TOOL IN TEACHING MATHEMATICS

NSF Grant # TEI8550088, 8741127

This booklet is the last in a series of nine booklets which constitute the Hofstra University Teacher Training Institute (TTI) packet. The Institute was a National Science Foundation supported three-year program for exemplary secondary school mathematics teachers. Its purpose was to broaden and update the backgrounds its participants with courses and special events and to train and support them in preparing and delivering dissemination activities among their peers so that the Institute's effects would be multiplied.

This packet of booklets describes the goals, development, structure, content, successes and failures of the Institute. We expect it to be of interest and use to mathematics educators preparing their own teacher training programs and to teachers and students of mathematics exploring the many content areas described.

"The Computer as an Experimental Tool in Teaching Mathematics" was a basic course offered as part of TTI's cycle of courses. This booklet describes the content and approach of this course - mathematics is taught in a way analagous to how science is generally taught. Using the personal computer as an experimental tool in a mathematics laboratory provides the student with an opportunity to get involved in the discovery process: to make conjectures, to test them, to see the results and thus be able to adjust the conjectures being tested. A list of course topics is presented and several are described more fully: Iterated Quadratic Maps, Fractals, Momentum in Baseball, and Sorting with the Computer.

The computer as an experimental tool in teaching mathematics

Harold M. Hastings[1]
Joyce Bernstein[1]
and
Robert Silverstone[1]

Department of Mathematics
Hofstra University
Hempstead, NY 11550

[1] Partially supported by NSF Grant 8550088.

This is booklet number 9 in the series of publications by the Hofstra University Teacher Training Institute.

Science in high school and college is generally taught with the aid of laboratories. Laboratories provide the student with a chance to get involved in the discovery process: to make conjectures, to test them, and to see the results of this process. The advent of personal computers with graphics has made it possible to use the computer as a similar experimental tool in a mathematics laboratory. This paper describes the authors' collective experience in using the computer as an experimental tool in such a laboratory.

This paper is an expanded version of a talk by one of the authors (HMH) at the December 1987 meeting of the National Council of Teachers of Mathematics. The paper reflects the experience of HMH in teaching an experimental course to selected high school teachers in a National Science Foundation Sponsored Teacher Training Institute at Hofstra University, as well as the experience of JB and RS in applying the techniques of that course in the high schools. We have not sought to provide a text on education, but rather to share some of our experiences.

We thank Marie Hermann and Helene Morris for preparing this manuscript.

Contents.

1. Overview

This set of notes is based on 1987 and 1988 class in the NSF-sponsored Teacher Training Institute at Hofstra University. The 1987 class met for a total of 30 hours (twelve 2-1/2 hour classes); the 1988 class for 9 such sessions. The audience consisted of superior, well-motivated high school teachers, who had a prior course in Pascal. Many had no recent experience with calculus. Many possible courses can be designed around these topics, which provide enough material and references for a one-year sequence.

The goal of the class was to develop the use of the computer (in high school and calculus level mathematics) as an experimental tool in discovering mathematical ideas. The course emphasized experimental mathematics, in analogy with typical physics and chemistry classes. Thus we seek to use the "discovery method" in a variety of "advanced" topics accessable to high school students. Specific topics were chosen to demonstrate numerical and graphical techniques. The topics were chosen on the basis of mathematics level required, accessibility to the discovery method, and my personal interests, as well as to provide a useful and diverse experience for the audience. No specific attempt was made to cover the Advanced Placement syllabus or other syllabi, although the topic on sorting was added to the course at student request. A typical one-semester course would cover 4-6 topics. The topics are:

1. A model for population dynamics.

A simple model for population dynamics: the discrete time logistic model. Construction of the model. Method of computer simulation. Chaotic dynamics - determinism versus randomness.

2. Random numbers.

Linear congruence random number generator. What is a random number? Use of computer graphics. The birthday problem and its consequences. Probability and baseball - see *Section 4*.

3. Fractals I.

Self-similarity. Regular fractals and recursion. See *Section 3.*

4. Fractals II

Fractals constructed by iterating quadratic maps. The Mandelbrot set and Julia set. Programming for time efficiency. See *Section 2.*

5. Numerical integration.

From a classical Greek formula for volume to Simpson's rule. Project on error analysis.

6. Numerical differentiation.

Computer evaluation of limits, choosing the denominator appropriately. Project on error analysis.

7. Hooke's law and the vibrating spring.

Derivation of sinusoidal motion from elementary principles.

8. Matrix models (age structured populations).

Leslie matrices. Eigenvalues, eigenvectors, and convergence. Numerical computation of dominant eigenvalues.

9. Sorting.

See *Section 5.*

We now illustrate several of the above topics.

2. Iterated quadratic maps

The study of iterated quadratic maps is a nice demonstration of the use of computer graphics to uncover beautiful and interesting phenomena in mathematics.

We begin by considering the process of *iteration.* Let f be a function from the real numbers to the real numbers. One may choose a real number x_0, and form the sequence

$$x_0, x_1 = f(x_0), x_2 = f(x_1), x_3 = f(x_2), \dots .$$

(Many students have seen the iteration process in the computation of compound interest. Here the amount of money in an account at the end of each period is obtained by multiplying the amount at the end of the previous period by the quantity (1 + the interest rate per period). Thus compound interest involves iterating the function

$$f(x) = (1 + r)x$$

where r denotes the interest rate and x the amount of money.) We are concerned with properties of the sequence of points obtained by iterating the map f. Among the simplest questions one can ask is whether the sequence $x_0, x_1, x_2, \dots$ is bounded or not.

We also make this question simpler by first considering only linear maps:

$$f(x) = ax + b.$$

In this case, it is not hard to see that the sequence of points obtained by iteration is bounded if $|a| < 1$, and in general unbounded if $|a| > 1$. If $b = 0$, then f takes the simpler form $f(x) = ax$, and an easy calculation yields

$$x_n = a^n x_0.$$

The convergence properties of the sequence of points $\{x_n\}$ follow easily. (In the example of compound interest above, $a = 1 + r$.) In the case where b is not necessarily equal to 0,

$$x_n = a^n x_0 + a^{n-1}b + a^{n-2}b + a^{n-3}b + \dots + b$$

$$= a^n x_0 + (1 - a^n)\, b \,/\, (1 - a),$$

summing a geometric series. The convergence properties of the sequence of points $\{x_n\}$ follow easily. We leave details and a more precise discussion to the reader.

We now consider the case of quadratic maps:

$$f(x) = ax^2 + bx + c.$$

The answer is now more interesting, even in seemingly trivial cases such as $f(x) = x^2$. Here the sequence of points obtained by iteration is bounded provided the first point x_0 satisfies $|x_0| \leq 1$, and unbounded provided that the first point x_0 satisfies $|x_0| > 1$. The most frequently studied case is

$$f(x) = x^2 + c.$$

Here the results depend upon both the starting point x_0 and the parameter c in a complex way.

We now consider iterating the equation

$$f(z) = z^2 + c \tag{2.1}$$

for complex z (and possibly complex c).

(We recall here that a complex number is a number of the form $a + bi$, where i denotes $\sqrt{-1}$. Complex numbers are added as if they were binomials with i as a variable; for example, $(a + bi) + (c + di) = (a + b) + (c + d)i$. Similarly, complex numbers are multiplied as if they were binomials with i as a variable, except that i^2 is replaced by -1; for example, $(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$. Complex numbers may be represented as points in the plane, with the complex number $a + bi$ plotted as the point (a,b). The length of a complex number is then given by the Pythagorean theorem: $| a + bi | = (a^2 + b^2)^{1/2}$.)

As above, for the results of iterating (2.1) for $c = 0$ are easy to see: if $|z| < 1$ then the iterates approach 0; if $|z| = 1$, then the iterates all have absolute value 1, and if $|z| > 1$ then the iterates approach ∞. The question is how do we study equation (2.1) if c is not zero. It is here that computers come to the rescue.

First, to a computer a complex number is just a pair of real numbers or a vector, eg,

$z = x + iy$ corresponds to (x,y).

If we suppose that c is real, we may write the results of applying f to $z = x + iy$ as

$$x_{new} = x^2 - y^2 + c,$$

$$y_{new} = 2xy.$$

Therefore one can just write a brief computer program to study the iteration, for example:

```
program iterate;
        {Copyright (C) Harold M. Hastings. All rights reserved except that
copies may be made for non-profit educational purposes provided that a
copy of this copyright notice appears on all such copies.}


uses
        crt;
var
        x,y,xnew,ynew,c : real;
        i,imax : integer;
begin
        writeln('How many iterates ?');
        readln(imax);
        writeln('What are the coordinates of the starting point, z0 ?);
        readln(x,y);
        for i = 1 to imax do
                begin
                        xnew := x*x - y*y  +c;
                        ynew := 2xy;
                        x := xnew;
                y := ynew;
                writeln(i,' ',x,' ',y);
            end;
end.
```

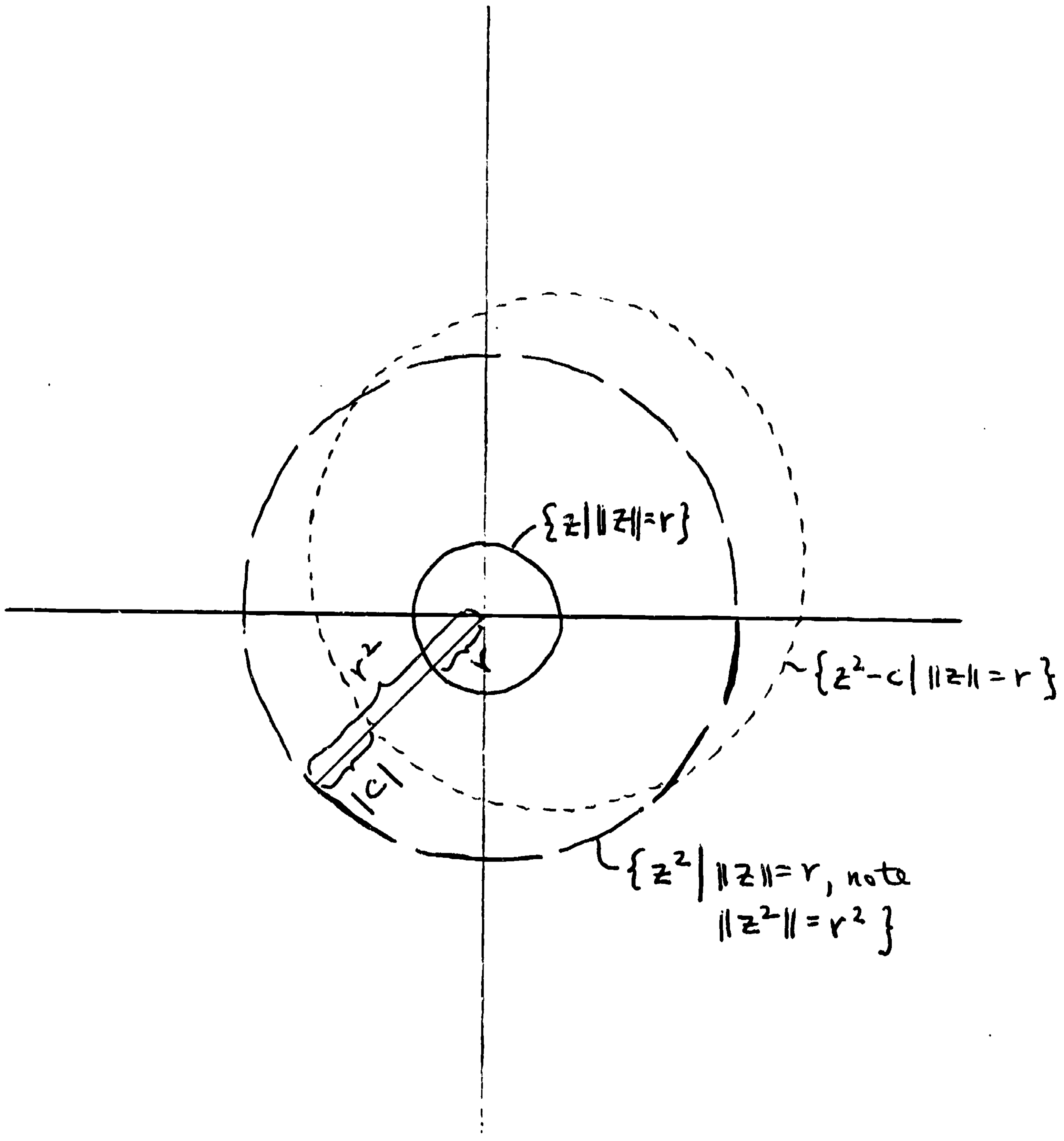
Perhaps one can tell whether the sequence of points generated by this program approaches ∞ or not. However, since we cannot compute the entire sequence of iterates, it would be nice to have a criterion for testing whether the sequence of points generated by this program approaches ∞. The following lemma provides such a criterion.

2.2. Lemma. Suppose $|c| < 1$ and $|z| \geq 2$. Then $|z^2 + c| \geq |z| + 1$.

Proof.
$$\begin{aligned} |z^2 + c| &\geq |z^2| - |c| && \text{(by the triangle inequality)} \\ &\geq |z^2| - 1 && \text{(since } |c| < 1\text{)} \\ &\geq 2|z| - 1 && \text{(since } |z| \geq 2\text{)} \\ &= |z| + (|z| - 1) \\ &\geq |z| + 1 && \text{(since } |z| \geq 2\text{),} \end{aligned}$$

as required.//

Lemma 2.2 implies that once the length of any iterate z_n exceeds 2, then subsequent iterates march off to ∞: $|z_{n+k}| \geq 2 + k$. The following figure illustrates the inequalities in the Lemma 2.2.


$\{z \mid \|z\| = r\}$
$\{z^2 - c \mid \|z\| = r\}$
$\{z^2 \mid \|z\| = r$, note $\|z^2\| = r^2\}$
r^2
1
$|c|$

We now combine Lemma 2.2 with computer graphics to draw a picture of which points approach ∞. The program is written in Turbo Pascal 5 for the IBM PC using Hercules compatible color graphics, and is easily modified for other languages and computers.

```
program mandel;
        {Copyright (C)  Harold M. Hastings.  All rights reserved except that
copies may be made for non-profit educational purposes provided that a
copy of this copyright notice appears on all such copies.}

uses
        crt, graph;
var
   i,j: integer;           {screen coordinates }
   k: integer;             {counter}
   col: word;              {color}
   x,y:    real;           {"math" coordinates}
   xn:     real;           {nxew; the variable ynew is not needed in this
                            program}
   z:      real;           {square of length of (x,y)};
   c:      real;           {constant in quadratic map}

begin
     readln(c);            {c must satisfy -1 < c < 1; the r  der may add a
                  check on c if required}
```

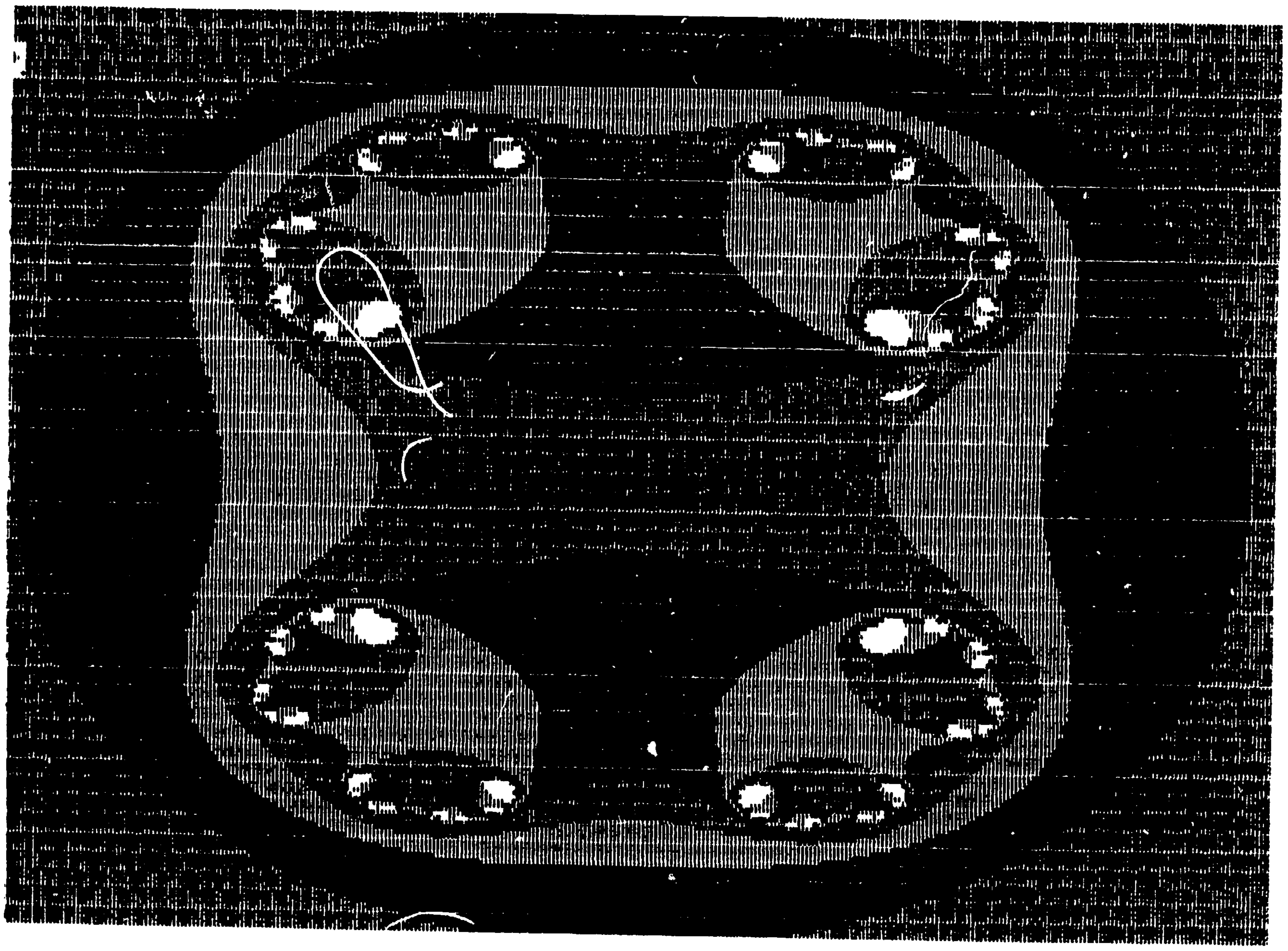
```
initgraph(0,0,' ');    {Turbo 5 comand for hercules color graphics on
                        IBM PC, modify as appropriate}

for i := 0 to 319 do
   for j := 0 to 199 do    {loop over 320 x 200 graphics screen}
      begin
            x := (i-160)/128.0;
            y := (100-j)/64.0;    {x and y scale factors, adjust as
                        appropriate, note the use of "100-j" in order to make
                        the y-axis point up as usual, the action usually takes
                        place within -1 < x < 1 and -1 < y < 1}
            k := 0;          {initialize counter}
            z := x*x + y*y;   {initialize length squared}
            while  ((k < 10) and (z < 4.0)) do
                        {the "k < 10" criterion prevents infinite loops and
                        might be adjusted by the reader.  The "z < 4" criterion
                        detects points for which we know that future iterates
                        will approach infinity; see Lemma 2.2, above.  Some
                        points take more than10 interations to escape the
                        disk  "z < 4".}
               begin
                        {first replace x + iy by (x+iy) squared + c}
                        xn := x*x - y*y + c;       {real part}
                        y := 2*x*y;                {imaginary part}
                        x := xn;
```

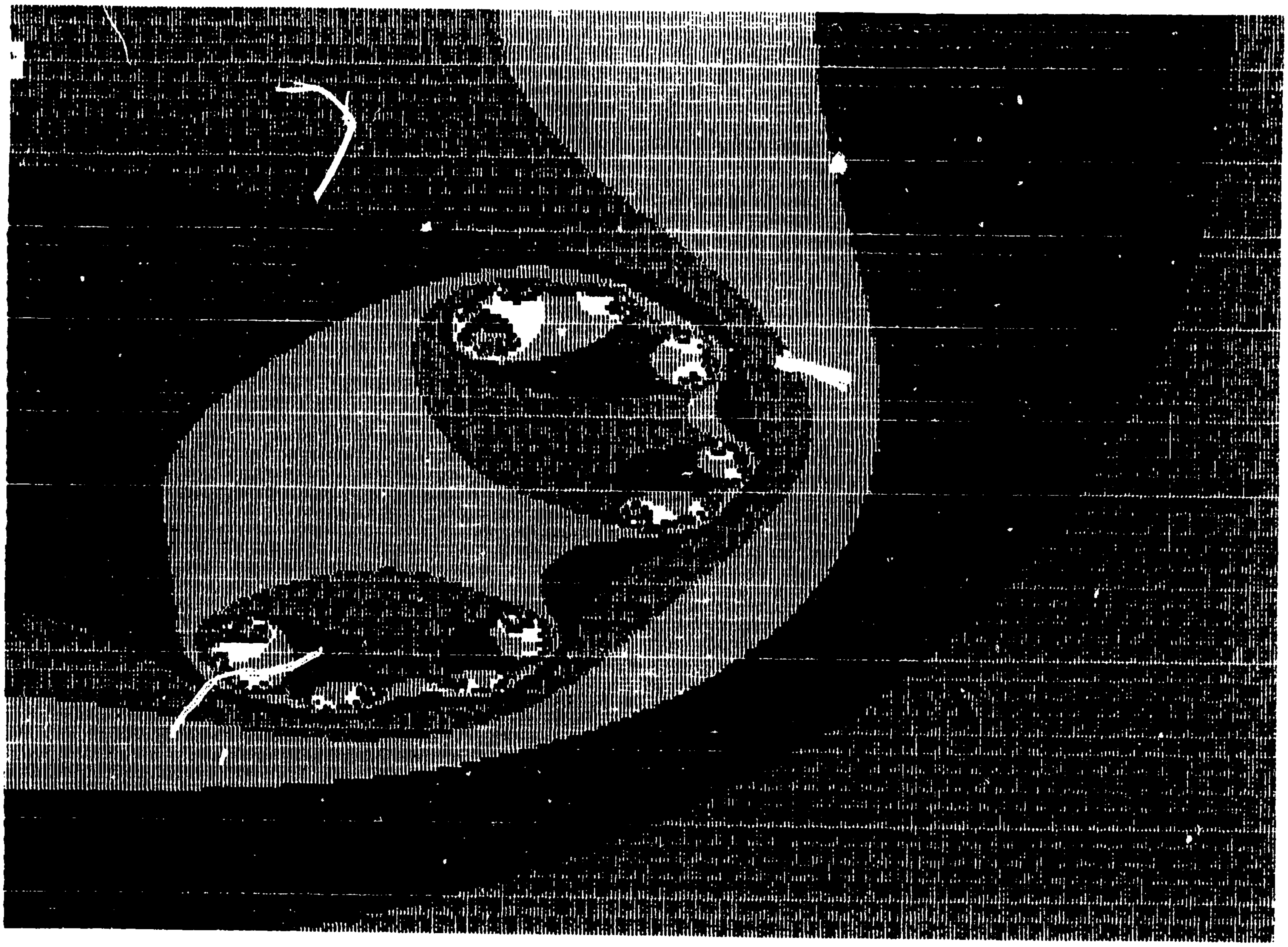
```
                z := x*x + y*y;          {compute length}
                k := k+1;                {iterate counter}
            end;
        if k := 10 then  {point did not escape}
            col := 0
        else
            col := k - 3*(k div 3) + 1;     {point did escape;
                                            color := k mod 3 + 1
                                            indicates time until
                                            escape.  Other formulas
                                            might be used.}
          putpixel(i,j,col);
      end;          {loop through points}
end.  {program}
```

We now illustrate the results of several runs of this program.

<u>page 14</u>: run with $c = 0.56$.

<u>page 15</u>: run with $c = 0.56$, but with "$k < 10$" replaced by "$k < 20$", "if $k = 10$" replaced by "if $k = 20$", and blown up by 200 %.



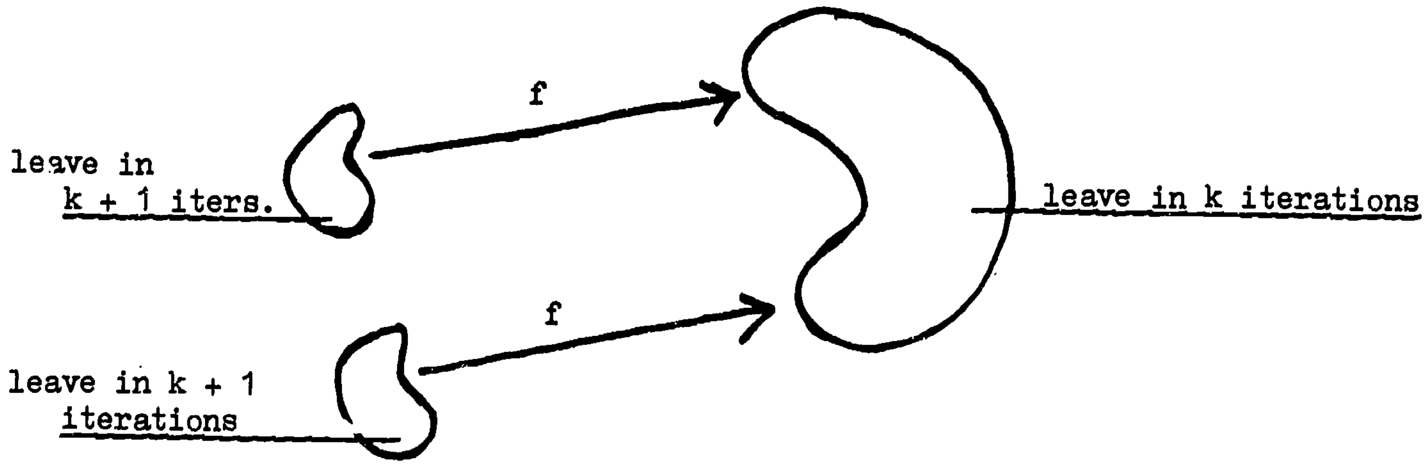


We make several observations and suggest several exercises.

(1) If c is real, as in the program, the resulting picture is symmetrical about both the x and y axes. We leave the proof as an exercise, but give a few hints. First, replace z by $-z$, and observes that $z^2 + c = (-z)^2 + c$. Thus the fates of the points z and -z are the same. This yields symmetry about the origin. For symmetry about the y-axis, replace y by -y, and compute the length of $z^2 + c = (x+iy)^2 + c$ and $(x-iy)^2 + c$. Then show symmetry about the x-axis, using geometry.

(2) What happens if c is not real ?

(3) The figures drawn by the program *mandel* above appear self-similar in that that each blob corresponds to two smaller blobs, of roughly similar shapes. The term "self-similar" will be defined in Section 3, below. Here is a sketched explanation of why the figures are self similar. Suppose that a given blob consists of points which leave the circle $|z| < 2$ in k iterations. Then there are points which map to this blob in one iteration, and thus leave the circle $|z| < 2$ in $k+1$ iterations. Since the map $f(z) = z^2 + c$ is usually two to one, there will be two blobs which leave the circle $|z| < 2$ in $k+1$ iterations. See the following figure.



For self-similarity, we invoke the fact that f has a derivative $f'(z) = 2z$. Thus f rotates and stretches small vectors Δz by a locally constant amount:

$$f(z + \Delta z) = f(z) + 2z \cdot \Delta z + (\textit{small error}).$$

Multiplication by $2z$ multiplies lengths by $|2z|$ and rotates vectors through an angle $arg(z)$ with $cos(arg(z)) = x$ and $sin(arg(z)) = y$ where $z = x + yi$. We refer the reader to any introductory text on complex variables. This implies that blobs are stretched and rotated by locally constant amounts.

We encourage the reader to experiment further. For further reading we suggest the following.

References

Mandelbrot, B.B. 1977, 1982. **Fractals: form, chance and dimension, The fractal geometry of nature,** Freeman, San Francisco.

Pietgen, H.O., and P.H. Richter, 1986. **The beauty of fractals: images of complex dynamical systems,** Springer- Verlag, New York and Berlin.

Remarks. We began with some simple questions. We first observed that the computer could be used to conduct experiments in order to try to answer those questions. We then saw that one could prove a useful rule (the lemma above) in order to use the computer to answer questions about the properties of certain sequences. This combines inductive reasoning from the experiments with deductive reasoning used in mathematical proofs. Many mathematicians are familiar with this combination of techniques, yet current curricula provide little useful experience for the student. Finally, we used computer graphics (combining algebra and geometry) to illustrate the answers to a mathematical question. The study of the properties of the answers leads to a new field, the field of fractals, which is explored further in the next chapter.

3. MORE ON FRACTALS

FRACTAL GEOMETRY

AND ITS APPLICATIONS

IN THE MATH CLASS

Fractal Geometry is one of the fastest growing mathematical disciplines.

The term "*fractal*" was coined by B. Mandelbrot of I.B.M, in the 1970's to describe the geometry of chaos. There are many ways to describe what a fractal is. Some of these descriptions are:

a) A fractal is a figure that has a FRACTIONAL dimension.

A line	has dimension	1
A square	has dimension	2
A cube	has dimension	3
A fractal	can have dimension	1.2345···

b) A fractal is a process of becoming rather than being

c) A fractal can be a self-similar object.
This means that at any level of magnification, any p 't of a fractal can look exactly like the initial view. Self-similarity is related to the process of **RECURSION**

d) An object of great complexity.

e) A geometry that accurately describes the real world (or any other world)

In this session, we will examine the following ideas:

a) Randomness and chaos... The CHAOS GAME

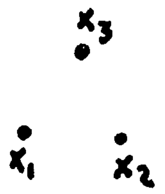
b) Self-similarity ....The KOCH SNOWFLAKE and its cousins

c) Dimension, both topological and fractal

d) Applications of these ideas in the Math curriculum

e) Computer generation of these shapes

THE CHAOS GAME



Start with the vertices of a triangle, say X_1, X_2 and X_3.
Let **P** be any point in the plane. The **CHAOS GAME** is played as follows:

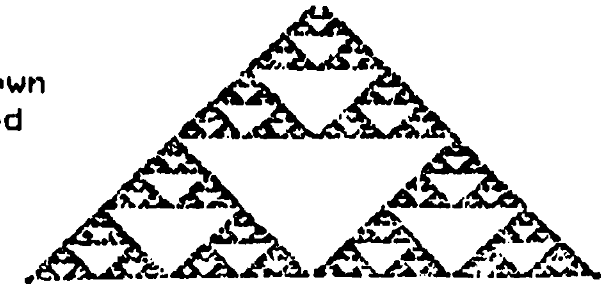
a) Choose any one of the three vertices at random, say X_1.

b) Mark the **MIDPOINT** of the segment $\overline{P X_1}$. call this point **P**.

c) Go back to step (a)

Question: WHAT is the resulting object, and what are its properties

WHY does the object appear as it does?



The figure produced by the CHAOS GAME is shown at the right. Although the object was formed by a random process, it appears to have a definite form and structure.

Observations:

a) The object has the property of being **SELF-SIMILAR** If you examine any of the sub-triangles, you will notice that they are **IDENTICAL** to the original, except for **SCALE**

This property of <u>self-similarity</u> defines the figure as a **FRACTAL**

b) We can calculate the area of the object.

$$A - A/4 - A(3/16) - A(9/64) - \cdots - A(1/4)(3/4)^n - \cdots$$

where A is the area of the entire triangle

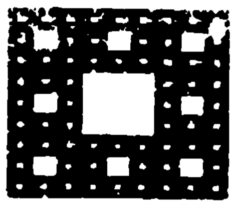
c) This concept leads to further exploration:

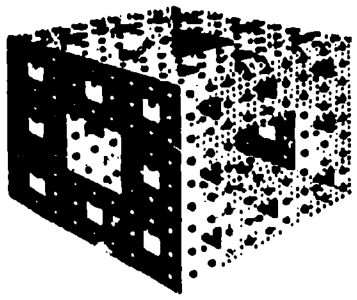
i) What happens if we went 1/3 the way from the vertex?

ii) What happens if we went 1/3 the way from P ?

iii) What if a square, or any other polygon were ured instead of the triangle?

The figure formed is called the **SIERPINSKI TRIANGLE.** Cousins of the triangle are the **SIERPINSKI CARPET**, pictured below left, and the **MENGER SPONGE**, pictured below right.





By changing the initial conditions, but by using the same process, more realistic images, such as the fern, pictured below, can be generated.

A simple BASIC program to generate the Sierpinski triangle is:

Program is in ^PPLESOFT.. Adjust graphics for your own computer)

```
10 X(1) =__ : Y(1) = __ : X(2) = __ : Y(2) = __ : X(3) = __ : Y(3) = _
   ( Choose your own coordinates for the triangle's vertices )
20 XP .= __ : YP = __    : REM  choose your own starting point (XP,YP)
30 HGR : HCOLOR = 7
40 R = INT(3 * RND(1) + 1 )
50     XP = ( X(R) + XP ) / 2  :  YP = ( Y(R) + YP ) / 2
60     HPLOT XP,YP
70 GOTO 40
```

THE KOCH SNOWFLAKE

The KOCH SNOWFLAKE is one of a class of fractals in which a straight line segment is replaced with a polygonal line, called a generator.

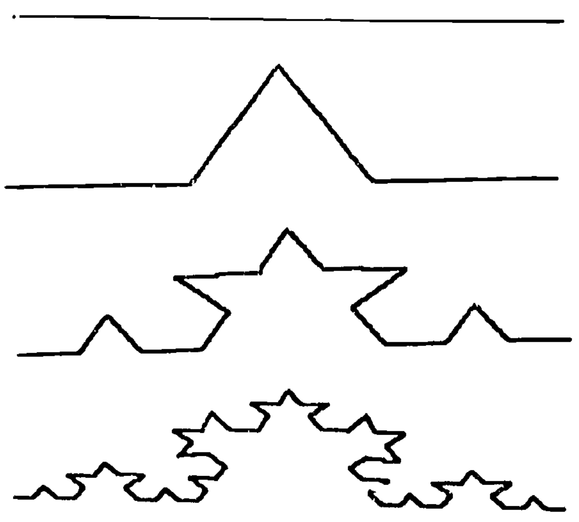
The SNOWLINE is constructed in the following manner:

a) Begin with a line segment

b) Divide the segment into 3 equal parts, and replace the middle third with two line segments,

c) repeat (b) on each of the resulting line segments.

d) go to (c)



We observe that at each stage, the length of line segment increases by a factor of 4/3 , hence at stage n, the length of the "curve" is

$$L_n = L*(4/3)^n$$

where L is the length of the initial line segment.

Clearly, we can see that $\lim_{n \to \infty} L_n = \infty$, hence is an unbounded length in a finite span.

To do this program in BASIC is quite complex, because to do it efficiently requires **RECURSION**, or the ability of a subroutine (procedure , function) to call itself up. Below are two programs to draw the Koch line and the snowflake:

LOGO

```
TO KOCH :LENGTH :LEVEL
   IF :LEVEL = 0 [FORWARD :LENGTH STOP]
   KOCH :LENGTH/3 :LEVEL-1
   LEFT 60
   KOCH :LENGTH/3 :LEVEL-1
   RIGHT 120
   KOCH :LENGTH/3 :LEVEL-1
   LEFT 60
   KOCH :LENGTH/3 :LEVEL-1
END
```

PASCAL

```
PROCEDURE KOCH(D:REAL;L:INTEGER)
 BEGIN
   IF L = 0 THEN MOVE(D)
   ELSE
     BEGIN
       KOCH(D/3,L-1);TURN(60);
       KOCH(D/3,L-1);TURN(-120);
       KOCH(D/3,L-1);TURN(60);
       KOCH(D/3,L-1)
     END
 END;
```

The Koch Line has some interesting properties:

a) For any initial span, the "length" of the curve is infinite
b) Although the curve is continuous, at no point is there a derivative (an example of where continuity is not sufficient for the existenc of a derivative.)

The koch line can be expanded to the KOCH SNOWFLAKE by:

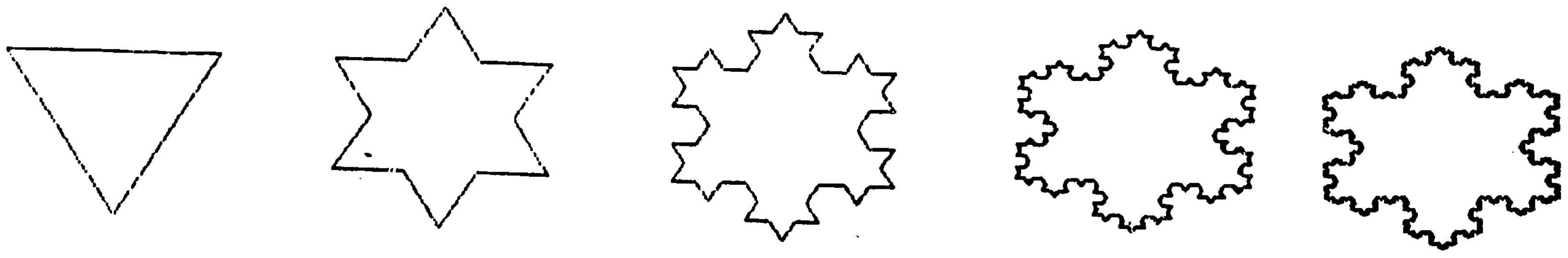
LOGO

```
TO SNOWFLAKE :LENGTH :LEVEL
   REPEAT 3 [KOCH :LEVEL :LENGTH RIGHT 120]
END
```

PASCAL

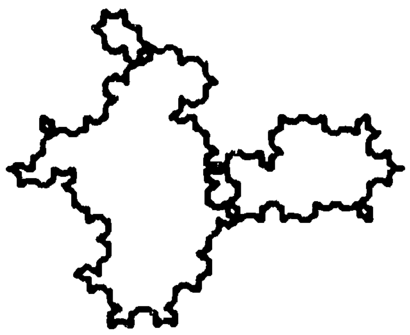
```
FOR I := 1 TO 3 DO
  BEGIN
    KOCH(LENGTH,LEVEL);
    TURN(-120)
  END;
```

The first five stages of the **SNOWFLAKE** are shown below:



Can we prove that the area is bounded, without actual computation?
Can the "actual" area of the snowflake be calculated?
Here we have an example of an infinite perimeter bounding a finite area.

Fractals are used to model the real world. One application of the snowflak is in the representation of islands and clouds. The coastlines of islands and the boundries of clouds can be thought of as random. By introducing rnadom lengths and angles into the generator, we can simulate these natura structures.



Random KOCH ISLAND

DIMENSION

It is our normal understanding that the "dimension" of an object is an **integer**. We know this because we asssociate dimension with direction. We say that a line is 1 -dim because we can travel in essentially one direction, forwards (or backwards). A plane is 2-dim because we can walk not only forwards (& backwards) but up and down. We have a sense that dimension is related to the "amount of space that is taken up". This notion of dimension is called the **TOPOLOGICAL** definition of dimension.

We can see how dimension is developed by the following considerations:

a) consider the line of length 1 ___________

Suppose that we triple the length. Now we can see that three of the original segments will cover the new length.

We can write $3^1 = 3$, where the power, 1 , is the dimension.

b) If we consider a unit square, and triple the dimensions, it requires 9 of the original squares to cover the new one.

We can write $3^2 = 9$, where the power, 2 , is the dimension.

c) If we consider a unit cube, and triple the dimensions, it requires 27 of the original cubes to "fill" the new one.

We can write $3^3 = 27$, where the power, 3 , is the dimension.

These examples lead to a relationship between the number of self-similar parts N generated and the scaling factor (R). It is:

$R^{\text{dimension}} = N$ or $R^D = N$ or $D = \log(N)/\log(R)$

	R	N	D	
Unit line	3	3	1	$3^1 = 3$
Unit square	3	9	2	$3^2 = 9$
Unit cube	3	27	3	$3^3 = 27$

and these conform to our usual sense of dimension.

Applying this relationship to our two "strange" objects, the gasket and the snowflake, we see:

TRIANGLE If we think of the lower left as the "unit" triangle, the triangle's sides are twice as large, and there are three triangles generated. Hence

$R = 2$ $N = 3$ $\text{Log}(3)/\log(2) = 1.5849\cdots$

SNOWFLAKE Each length is divided into 3 equal sections, and 4 segments replace the original 3, hence

$R = 3$ $N = 4$ $\text{Log}(4)/\text{Log}(3) = 1.2618\cdots$

These strange objects have fractional dimensions. A question that arises is: "What is a meaning of a fractional dimension?"

The CARPET's dimension can be calculated by by observing that the original square is reduced by a factor of 3, and 8 squares are generated:

$$R = 3 \qquad N = 8 \qquad \log(8)/\log(3) = 1.89\cdots$$

THE CANTOR SET

One of the earliest fractals developed was the CANTOR SET. It is derived by

```
a)   Start with  a line segment           ---------------------------
b)   Divide into 3 equal parts, and       ---------         ---------
     remove the middle third              ---   ---         ---   ---
                                          - -   - -         - -   - -
c)   on each of the resulting segments,
     repeat part (b)
```

FRACTAL GENERATION USING TRANSFORMATIONS OF THE PLANE

The regular fractals discussed so far can be generated by considering the movements of points in the plane (or space) by means of

AFFINE TRANSFORMATIONS

An AFFINE TRANSFORMATION is a LINEAR TRANSFORMATION followed by some TRANSLATION or SHIFT. LINEAR FUNCTIONS consist of such movements as

REFLECTION EXPANSION DILATION ROTATION SHEARING

Suppose that T is a linear function of the plane to itself, and suppose that the point (x,y) is mapped, under T, to the point (x',y'). We can then write:

$$(x,y)T = (x',y')$$

$$\begin{tikzcd}(x,y) \arrow[r, bend left, "T"] & (x',y')\end{tikzcd}$$

(x',y') is called the **IMAGE** of (x,y) under T

This means that there are real numbers **a, b, c and d,** where

$$x' = ax + by$$

$$y' = cx + dy$$

This system of linear equations can be written as a **MATRIX** equation

$$(x',y') = (x,y)T = (x,y)* \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

An AFFINE TRANSFORMATION, A, can be written

$$(x,y)A = (x,y)T + (r,s)$$

where (r,s) is a trarslation, or shift r units Horizontally and s units vertically.

0

Returning to the SERIPINSKI TRIANGLE:

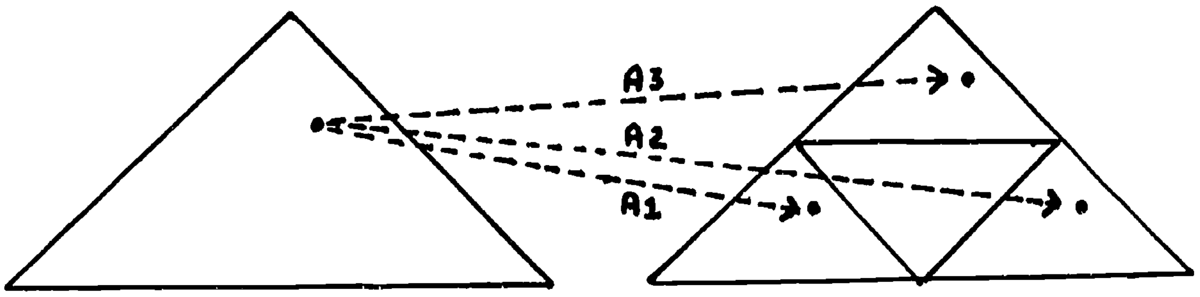
Our goal is to see where points go under affine transformations.

For the TRIANGLE, we will need to consider **THREE** transformations

A_1 : The image of (x,y) will be in the **LOWER-LEFT** triangle.

A_2 : The image of (x,y) will be in the **LOWER-RIGHT** triangle

A_3 : The image of (x,y) will be in the **TOP** triangle.

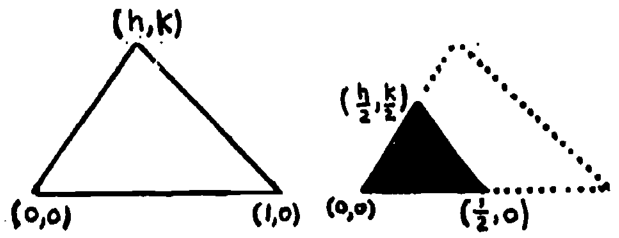


This gives the following set of affine transformations:

For A_1 : $(0,0)A_1 = (0,0)$

$(1,0)A_1 = (1/2,0)$

$(h,k)A_1 = (h/2,k/2)$

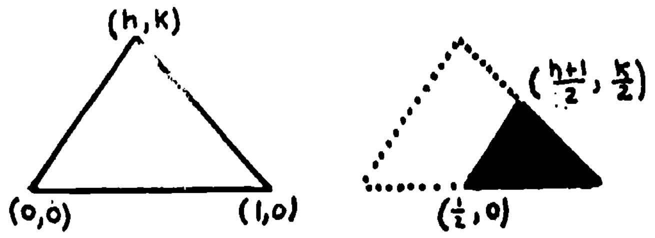


Resulting in: $(x,y)A_1 = (x,y) * \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$

For A_2 : $(0,0)A_2 = (1/2,0)$

$(1,0)A_2 = (1,0)$

$(h,k)A_2 = ((h+1)/2,k/2)$

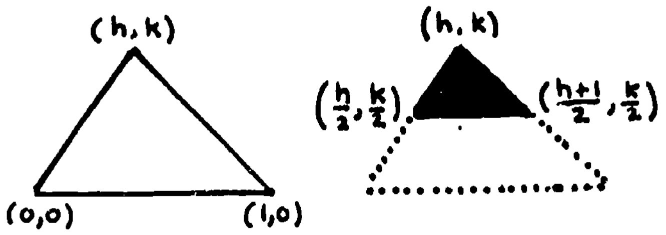


Resulting in: $(x,y)A_2 = (x,y) * \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} + (1/2,0)$

For A_3 : $(0,0)A_3 = (h/2,k/2)$

$(1,0)A_3 = ((h+1)/2,k/2)$

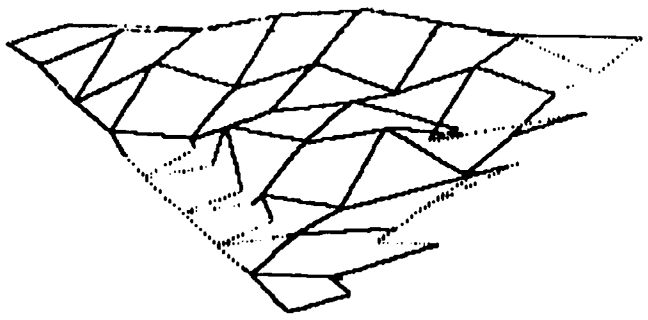
$(h,k)A_3 = (h,k)$

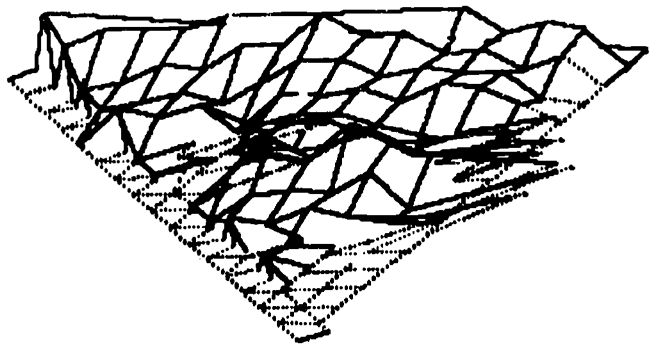


Resulting in: $(x,y)A_3 = (x,y) * \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} + (h/2,k/2)$

SOME APPLICATIONS OF FRACTALS

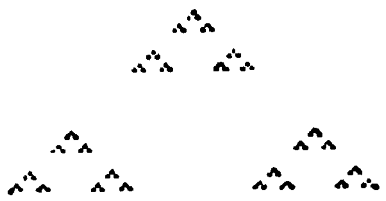
Fractal analysis is being used in studying almost all natural phenomenon. One example is in Motion pictures. The STAR TREK AND STAR WARS worlds were generated by fractal programming of computers. Below is a "simple" example of a randomly-generated landscape.



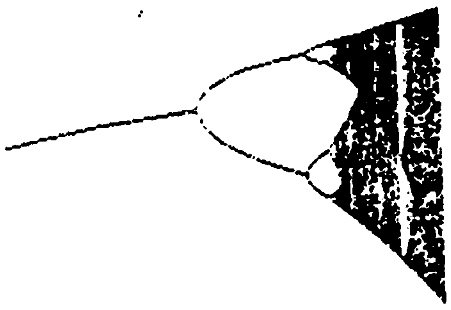


A GALLERY OF FRACTALS

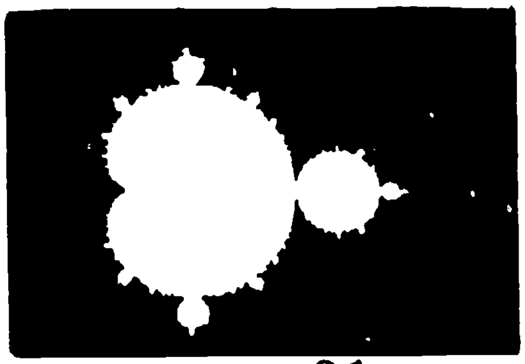
a) From the chaos game, if the point is chosen a distance 1/3 to the vertex instead of the 1/2, the result is:



b) The world-famous **BIFRUCATION** graph

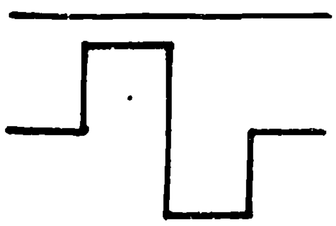


c) The grand-daddy of all fractals, the **MANDELBROT SET**



A SET OF PROBLEMS FOR YOU TO TRY

1. Start with a square and replace each side



of the square with the generator

Let P = initial perimeter and A = initial area.

a) Letting SQ(0) be the initial square, draw SQ(1) and SQ(2)

b) Find the dimension of this fractal (SQ(∞))

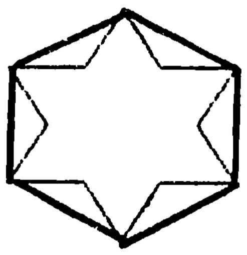
c) Find the area A(n) of SQ(n) and hence $AF = \lim_{n\to\infty} A(n)$

d) Find the perimeter P(n) of SQ(n) , and $PF = \lim_{n\to\infty} P(n)$

e) Write a program to produce the this fractal.

This fractal is known as the **KOCH ISLAND**

2. Give a geometric argument to show that the area of the **SNOWFLAKE** is bounded by the hexagon that circumscribes it at level 1.

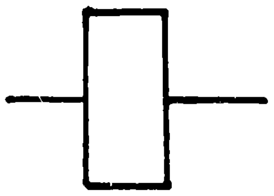


3. Refer to the **CANTOR SET**, page 6,

a) Calculate its dimension

b) calculate its length

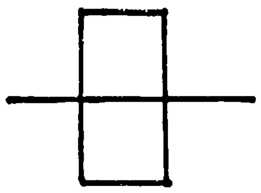
4. Start with a square and form the fractal by replacing each of the sides ———————— with the generator



a) Draw level 1 and level 2 of this fractal.

b) What is the dimension of this fractal?

5. Do the same as problem 3 with the generator



6. Refer to the **SIERPINSKI CARPET** on page 2

Let M(0) be the complete square at level 0,
M(1) be the square, at level 1, with the central square removed,
M(2) be at level 2, where the 8 additional squares are removed.

a) What is the dimension of this fractal?

b) At stage n
   i) S(n) = number of squares removed. What is S(n)?
   ii) What is the area of each of these squares ?
   iii) What is the total area, TA(n), removed

c) What is the area of this fractal ?

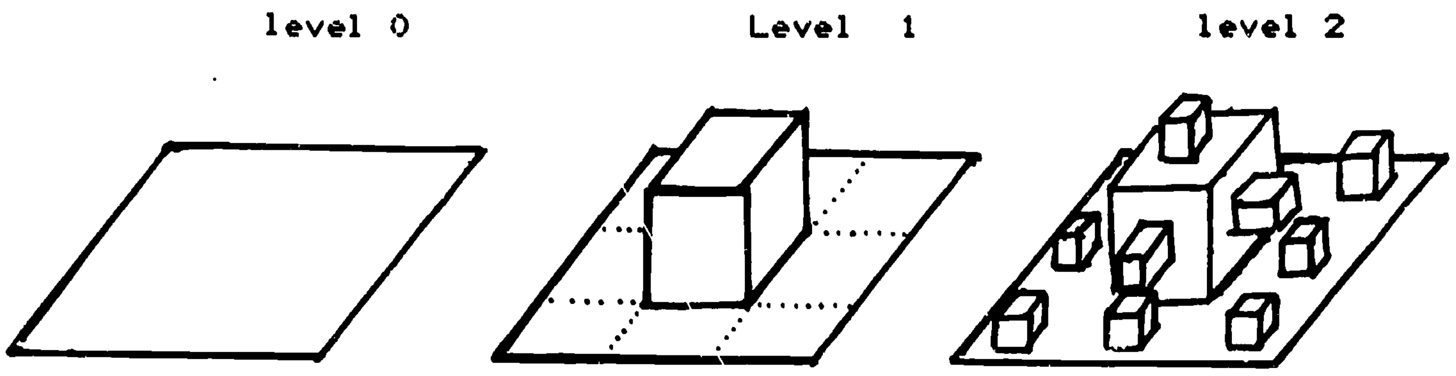
7. Refer the the **MENGER Sponge** on page 2.

a) Find the fractal dimension of the sponge

b) Calculate the surface area of this fractal.

c) Calculate the volume of this fractal.

8. Consider the following construction (drawn in 3-D perspective)



a) What is the dimension of this fractal

b) What is the surface area of this fractal?

c) What is the volume of this fractal?

d) What is the height of this fractal?

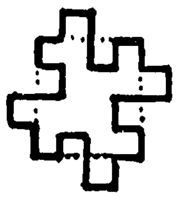
e) What is the relation between this fractal and the **MENGER SPONGE**?

9. a) Derive the affine transformations for the **CARPET**

b) Derive the affine transformations for the **SPONGE**

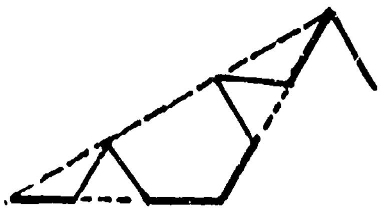
SOLUTION SUGGESTIONS

1. a) SQ(1) is



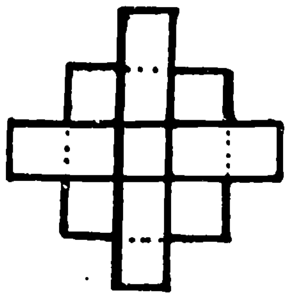
b) D = Log(8)/log(4) = 1.5
c) A(n) = A for every A.
d) P(n) = 2^n*P --> ∞
e) see next page (after # 9

2. Just prove that at each stage the triangles added in are within th line joining the furthest endpoints of the two adjacent sides:



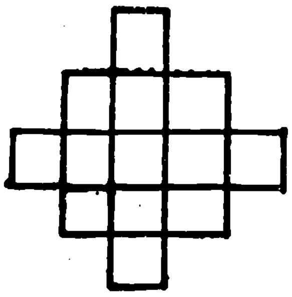
3. a) D = Log(2)/Log(3) ≈ .63   b) 0

4. Level 1:



Dim = log(8)/log(3) ≈ 1.89

5. Level 1:



Dim = log(9)/Log(3) = 2
(Is this a fractal???)

6. a) D = log(8)/log(3)
b) i) S(n) = 8^n ii) $(1/9^n)$*A   iii) TA(n) = $(8/9)^n$*A
c) 0 (consider the geometric series described by part (b)

7. a) Log(20)/log(3) ≈ 2.7268
b) At each stage, each sub-cube's surface area is increased by a factor of 2
c) 0... relate to CARPET

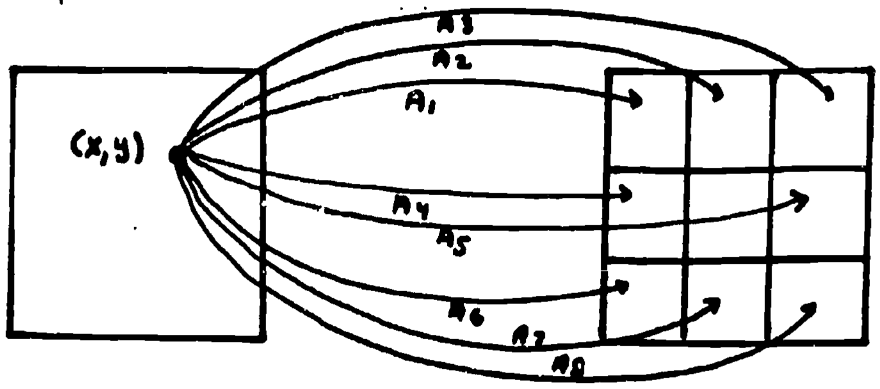
8. a) log(13)/log(3) ≈ 2.334
b) At stage n, the surface-area is $(13/9)^n$ of the initial square hence is infinite.

c) At stage n, the volume is increased by $(13/27)^n/27$
(assume there is is an initial cube of volume 1)

Volume = 1/14 of the initial surrounding cube

d) Height is (1/3) + (1/9) + (1/27) + ··· = 1/2

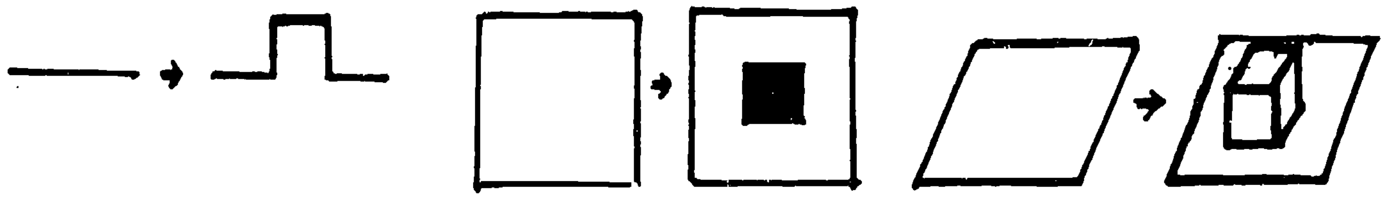
9. a) Requires 8 affine transformations



$(x,y)A_5 = (x,y)\begin{pmatrix}\frac{1}{3} & 0 \\ 0 & \frac{1}{3}\end{pmatrix} + (\frac{2}{3}, \frac{1}{3}$

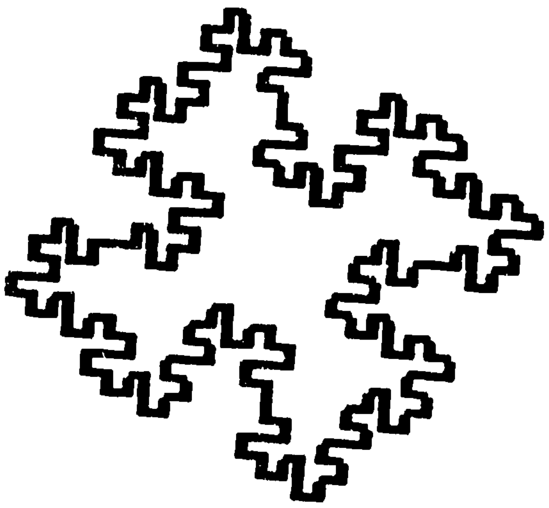
b) Requires 20 affine transformations.

Note the geometric "similarities" between the following fractals. Do you suppose that there may be some relationhip between them?



1e) LOGO program

```
TO ISLAND :LENGTH :LEVEL
   IF :LEVEL = 0  FD :LENGTH STOP
   ISLAND :LENGTH/4 :LEVEL-1
   LEFT 90
   ISLAND :LENGTH/4 :LEVEL-1
   RIGHT 90
   ISLAND :LENGTH/4 :LEVEL-1
   ISLAND :LENGTH/4 :LEVEL-1
   LEFT 90
   ISLAND :LENGTH/4 :LEVEL-1
   LEFT 90
   ISLAND :LENGTH/4 :LEVEL-1
   RIGHT 90
   ISLAND :LENGTH/4 :LEVEL-1
END
```


KOCHISLAND 60 2

```
TO KOCHISLAND :LENGTH :LEVEL
   REPEAT 4 [ ISLAND :LENGTH :LEVEL RIGHT 90 ]
END
```

BIBLIOGRAPHY

BOOKS

Barnsley, Michael   Fractals Everywhere
Academic Press, 1988

Discusses the mathematics behind creating fractal images on the computer. It is a highly technical book, but there is a wealth of information about applications of transformations.

Gleick, James   CHAOS, Making a New Science
Viking Press, 1987

A very readable introduction to the new science of CHAOS and its implications and applications. Discusses the history of fractals and their applications.

Mandelbrot, Benoit B.   The Fractal Geometry of Nature
W.H.Freeman & Co., 1983

The original, and classic text. This is the book that started it all. Very technical, and not an easy book to read.

Peitgen, H.O & Richter,P.H.   The Beauty of Fractals
Springer-Verlag, 1988

A book for the non-specialist. Examines how computer generated graphics via fractals are created. Gives good mathematical foundations for fractal geometry, and presents a concise history of the subject.

Peitgen, H.O & Saupe, D.   The Science of Fractal Images
Springer-Verlag, 1988

A HOW-TO-DO-IT book for creating your own fractals. Discusses algorithmic construction and applications. A worthwhile book.

Poundstone, William   The Recursive Universe
Wm. Morrow & Co.,1985
A delightful book giving much of the underlying rational and tools for understanding iteration and recursion, and how it applies to Math and Science.

Rucker, Rudy   Mind Tools
Houghton, Mifflin, 1987

Explores how fractals are 'used' in our everyday life, and how we use fractals in our thinking and viewing the world around us.

Stevens, Peter S   Patterns in Nature
Little, Brown and Co., 1974

A delightful book by a mathematically-inclined architect who presents nature in a most remarkable way.

ARTICLES

Barcellos, A	"The Fractal Geometry of Mandelbrot" The College Mathematics Journal March, 1984 pages 98 - 118
Dewdney, A.K.	**Computer Recreations of** Scientific American
	"A computer Microscope zooms ..." August, 1985 pages 16 - 24
	"Wallpaper for the Mind Sept,. 1986 pages 14 - 23
	"Probing the strange attractions of Chaos" July, 1987 pages 108 - 111
Thornburg, David	"Learning Curve the Math Microscope" A+ Magazine Nov. 1988 pages 105-107
van de Panne, Michael	"3-D Fractals" Creative Computing Vol 11, No,. 7
Wardrop, Simon	"Plotting Fractals on your Computer" MICRO March 1984

SOFTWARE

"Fractal Explorer"
ECLAT Micro Products
P.O. Box 750-756
Miami, FL 33257-0756

FILMS, ETC

ART MATRIX
P.O. Box 880
Ithaca, New Y[illegible] 14850

A "FRACTAL ST[illegible], selling posters,
picture cards, [illegible]oks , videos ,
tee shirts and other fractal goodies.

I

The following problems are examples of how the idea of ***SELF-SIMILARITY*** can be used to simplify the solution.

1. CONTINUED FRACTIONS

Let $$X = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}$$

The self-similarity in this problem is quite evident. We can re-write this expression as:

$$X = 1 + \frac{1}{X}$$

which results in the quadratic equation $X^2 - X - 1 = 0$

hence $X = \dfrac{1 + \sqrt{5}}{2}$

2. This is a problem that was given to me by a student:

Evaluate $\mathrm{Ln}(X^{\mathrm{Ln}(X^{\mathrm{Ln}(X^{\cdot^{\cdot^{\cdot}}})})})$

Let this expression be represented by Y, then clearly, we can write

$$Y = \mathrm{Ln}(X^Y)$$

$$Y = Y\cdot\mathrm{Ln}(X)$$

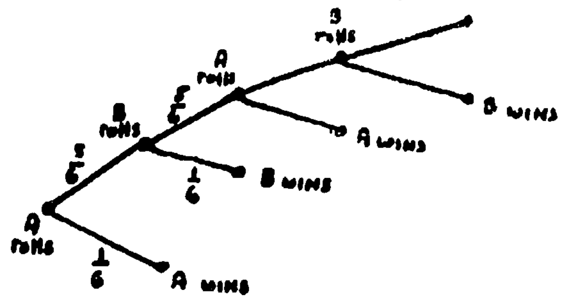
$$1 = \mathrm{Ln}(X)$$

$$e = X$$

The following two problems were given to me by Richard Brown of Exeter Academy, Exeter, NH.

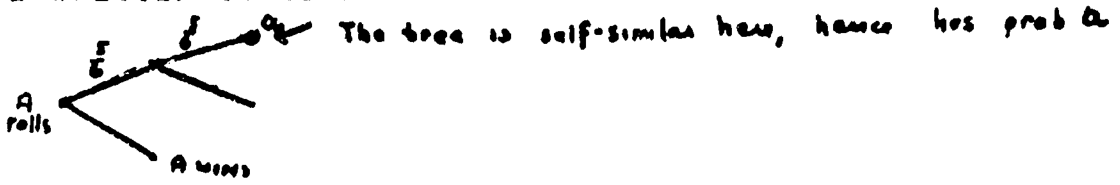
3. A and B, in turn, roll a fair die. A rolls first. The first person to roll a "6" wins. What is the probability, a, that A wins?

The tree below shows the progress of the game for the first 5 rolls.



Clearly, **a** = (1/6) + (25/36)(1/6) + (25/36)²(1/6) + ···

We can use the idea of *SELF-SIMILARITY* by observing that the tree can be re-written to look like



giving the equation **a** = (1/6) + (25/36)**a** --> **a** = 6/11

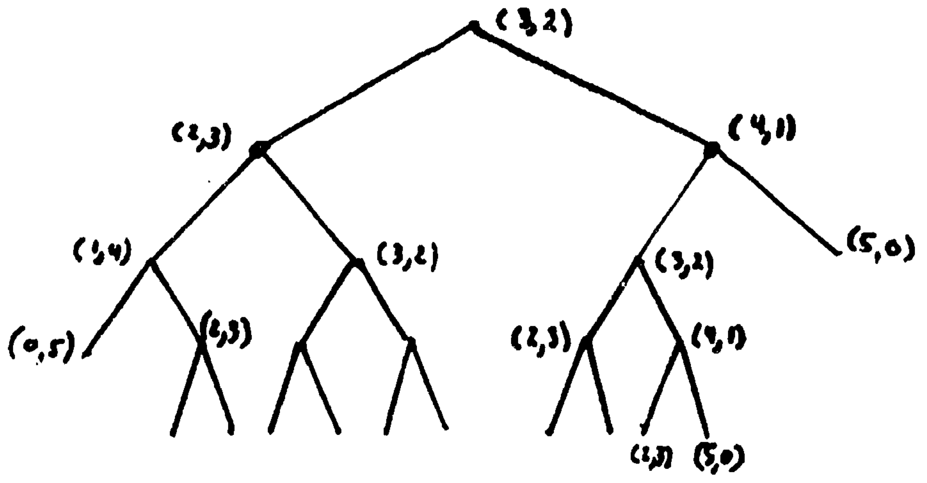
4. The classic <u>GAMBLER'S RUIN PROBLEM</u>

A and B play a game. A starts with \$3 and B starts with \$2. The game is as follows:

Each, in turn, flips a fair coin. If the tosser gets a **HEAD**, then he receives \$1 from the other person; otherwise he gives \$1 to the other person.

What is the probability that the first person wins?

In the solution, a tree will again be used. Let (a,b) represent how much money each has as each stage of the game. A tree, for the first 5 tosses is shown below:

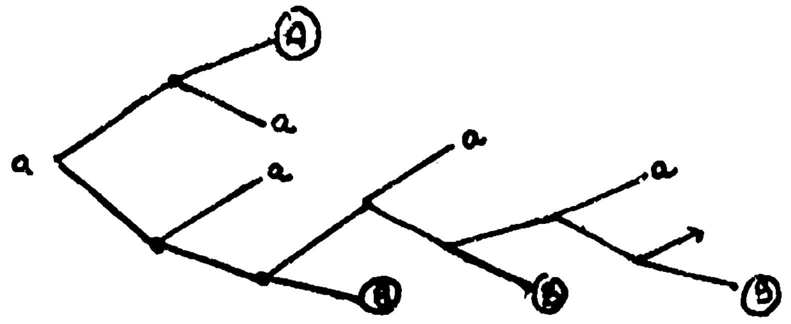


which gives the infinite series

a = (1/4) + 2(1/16) + 5(1/64) + 13(1/256) + ···

which is not an easy series to evaluate.

By appealing to *SELF-SIMILARITY* again, we can reduce the tree to the following by using the state (3,2) as the starting point:



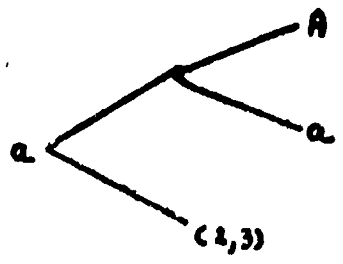
which gives the infinite series:

$$\mathbf{a} = (1/4) + (1/4)\mathbf{a} + (1/4)\mathbf{a} + (1/16)\mathbf{a} + (1/64)\mathbf{a} + \cdots$$

$$= (1/4) + (1/4)\mathbf{a} + \underline{(1/3)}\mathbf{a} \quad \text{(Sum the infinite geometric series)}$$

Hence $\mathbf{a} = 3/5$

We can reduce the tree still further by making the observation that the state (2,3) is the same as the state (3,2), except that it represents the probability of B winning ($\mathbf{b} = 1 - \mathbf{a}$)



This now gives the equation $\mathbf{a} = (1/4) + (1/4)\mathbf{a} + (1/2)(1-\mathbf{a})$

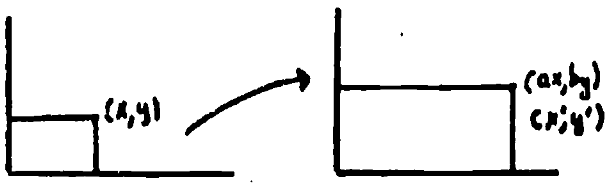
The class of problems where we can use the techniques of *SELF-SIMILARITY* is large and varied. This method of solution introduces the student to another way of viewing phenomena as a process, and in a fractal way. This method can also help to introduce the student to the ideas and techniques of RECURSION and and to reinforce the tool of INDUCTION.

I I

Following is a collection of <u>AFFINE TRANSFORMATIONS</u> that you can [illegible] in construction your self-similiar fractals.

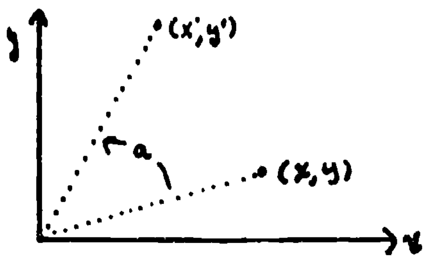
Let (x,y) be the given point, and (x',y') = (x,y)A

SCALING



$$A = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

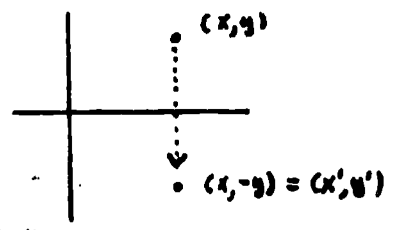
ROTATION



$$A = \begin{pmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{pmatrix}$$

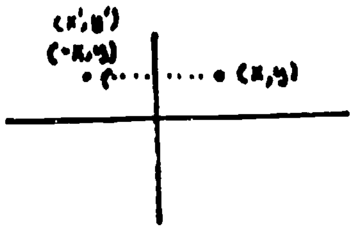
REFLECTION

a) <u>X-axis</u>



$$A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

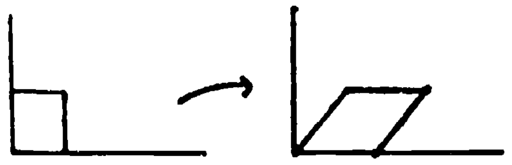
b) <u>Y-axis</u>



$$A = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

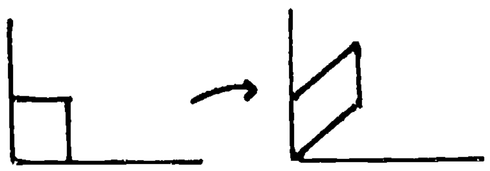
SHEAR

a) <u>X-direction</u>



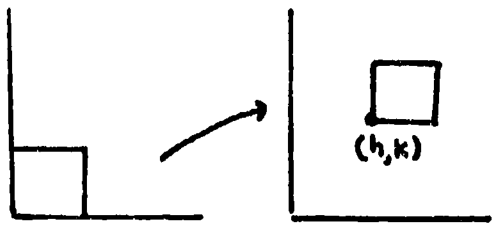
$$A = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix}$$

b) <u>Y-direction</u>



$$A = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$$

TRANSLATION (note: write (x,y) as (x,y,1))



$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h & k & 1 \end{pmatrix}$$

4. Is there momentum in baseball?

Harold M. Hastings
and
The 1987 TTI Class*

Department of Mathematics
Hofstra University
Hempstead, NY 11550

Sportswriters often discuss "momentum". We sought to determine whether there is momentum in baseball, in one special case, the 1986 Mets, using elementary probability. The results are interesting and the problem and techniques interested our class (outstanding high school teachers participating in a National Science Foundation sponsored Teacher Training Institute) and should interest many others. The calculations may stimulate additional investigation into ordinary events using elementary probability.

The 1986 Mets season** may be summarized as follows, where Wn denotes a string of n Wins, and Ln a string of n losses:

W 2, L 3, W 11, L 1, W 6, L 1, W 1, L 2, W 1, L 2, W 3, L 2,
W 6, L 2, W 1, L 1, W 2, L 1, W 3, L 1, W 7, L 2, W 1, L 1,
W 1, L 2, W 8, L 1, W 1, L 3, W 5, L 3, W 3, L 2, W 3, L 2,
W 3, L 2, W 3, L 1, W 3, L 2, W 1, L 4, W 6, L 1, W 5, L 1,
W 1, L 1, W 4, L 2, W 1, L 4, W 3, L 1, W 1, L 1, W 2, L 1,
W 4, L 1, W 5 (108 wins in 162 games).

We shall say define momentum as the tendency for wins to follow wins. This allows the questions about momentum to be phrased mathematically as follows:

1) If the Mets won the last game, what is the probability that they win the present game?

2) If the Mets won the last n games (for a fixed n), what is the probability that they win the present game?

3) Are there any statistically significant differences?

We shall use test the null hypothesis that there is no momentum, in which case the results of successive games are independent. More precisely, we consider the Bernoulli trials (see [1]) or any elementary statistics textbook) model that the 1987 Mets season consisted of 162 independent games, with a probability p of winning each game of 108/162 or .667.

For n independent Bernouilli trials, each with a probability p of success and $q=1-p$ of failure, and np and $nq \geq 25$, the expected results are essentially normal with mean np and variance npq [1].

There were $n=107$ games played following wins (the Mets won the last game). Of these, the Mets won 76, this gives a conditional probability for a win following a win of

$$p(\text{win}|\text{previous win}) = 76/107 = .710.$$

However, the null hypothesis gives an expected number of wins of

$$np = 107 \times .667 = 71.3,$$

with a standard deviation of

$$\sqrt{npq} = \sqrt{107 \times .667 \times .333} = 4.88$$

games. The number of excess wins, 4.7, is less than one standard deviation, and thus not statistically significant.

These calculations were repeated for the case of 2 or 3 previous consecutive wins, with the following results.

Number of previous wins	Actual results: Wins (trials)	(successes)	Prob. of win	Null hypothesis: Expected wins np	Standard deviation $\sqrt{npq}$
2	75	54	.720	50	4.08
3	53	35	.660	35.3	3.43

In neither case is the difference between the null hypothesis and the actual results statistically significant.

We invite the reader to continue with larger n, as we did, and see what happens. (We found no statistically significan "momentum" with our definition.) We note that had we found momentum from one game to the next, we would have considered a Markov chain model (see, for example [1]) which estimates the probability of winning each game from the results of the immediately previous game. We also invite other definitions of momentum.

Reference.

1. K.J. Smith, Finite Mathematics, Brooks/Cole Publishing Co., Monterey, CA, 1986.

*The 1987 TTI class consisted of Janet Barbera, Hilary Bernstein, Joyce Bernstein, Daniel Drance, Joan Fowler, Judith Gleicher, Arthur Hunsicker, Barbara Invidiata, John Keck, Gerald Lucchesi, Robert Silverstone, Lynne Unger, Phyllis Wald.

We all gratefully acknowledge the support of NSF grant ID No. 8550088.

**We thank the Elias Sports Bureau for graciously providing us with scores for all games in the 1986 Mets season.

5. SORTING WITH THE COMPUTER.

Joyce Bernstein

Introduction to Sorts

Teaching sort algorithms in a computer science class provides a nice bridge between programming and mathematics. It becomes necessary to understand what graphs of certain functions look like far removed from the origin. Recursion, used in the most effecient sort, is a close cousin of induction

When choosing a sort, there are three main considerations:

i: Programming time - for small arrays or files, use a simple sort.

ii: Execution time - measure of effeciency, a function of number of comparisons and data movements.

iii: Memory requirements - usually sacrificed in a trade-off for better effiociency.

Effeciency becomes important as the number of files to be sorted gets large. We measure effeciency using a figure proportional to some function g(N), of N, the size of the sort. "Big O" notation, O(g(N)), means proportional to g(N).

$$f(N) = O(g(N)) \implies k>0 \quad |f(N)| \leq k|g(N)|$$

For all of the sorts we will consider, which require both comparisons and data movements, g(N) is either n^2 or $n\log_2 n$. [illegible]structive to graph n^2 and $n\log_2 n$, especially as n gets large.

I. Exchange sorts: Exchange sorts move data into place, by position, one at a time.

<u>The Bubble Sort</u>, one of the easiest sorts to program, brings the smallest item to the front of the list in a manner which resembles "bubbling".

The algorithm does the following:

i: Compare items in pairs from the top of the list to the bottom, exchanging them if the front item is larger than the back item. (n-1 comparisons).

ii: Repeat the process another n-2 times. The jth iteration makes n-j comparisons. We can sum the comparisons, n(n-1)/2, and assume that swaps occur about half the time, or n(n-1)/4.

$$|n(n-1)/4| = (1/4)|n^2-n| \leq (1/4)|n^2| = O(n^2)$$

Bubblesort is obviously an $O(n^2)$ sort.

The Quicksort, invented by C.A.R. Hoare in 1962, moves one record at a time into its final position. However, in the process, which is recursive, the other records are moved to a position closer to their final ones. The algorithm involves a partioning of the array into 2^j sections on the jth calling. In the first pass, a pivot element is selected (usually [illegible]). At the end of the first pass, this element is in its final position, all elements smaller than it are ahead of it, and all items larger are behind it. The algorithm calls itself, choosing a pivot from each "half" of the list, and the simpler problem of sorting two smaller arrays is set up. The comparison takes place as follows: (assume the first pivot is item 1. When would you want to make a different choice?). Assign two variables, i&j, with i initially 2 and j initially N. Compare element i to the pivot. If it is smaller, increase i by 1. Stop when you get to an element greater than the pivot. Do the reverse with item j. Keep decreasing j by 1 until you get to an item less than the pivot. Exchange items i and j.

The quicksort is a powerful $O(n\log_2 n)$ sort. Only necessary moves are made, reducing overhead. This algorithm can be made even more powerful by moving pointers insead of data.

II. Insertion sorts: Each successive item is moved into an already sorted list.

Linear insertion mimmicks the way many of us arrange playing cards in our hand when we pick them up one at a time. Pick up any card. Pick up the second card and put it in order. Pick up the third card and put it in order, etc. The algorithm inserts the new element j by first comparing it to element j - 1, which is already sorted, then to element j - 2, etc, moving each of these elements up one place until it finds an element smaller than it. The [illegible] element is placed in its proper place. Notice the special condition which takes place when the element to be inserted is the smallest element on the list, up to the time of this insertion. The while loop is bypassed, so that the comparison a[place] > temp is not read w'hen place = 0.

In order to insert the jth item, approximately (j-1)/2 comparisons and data movements are required. Summing over the n elements in the array, we see that linear insertion is an $O(n^2)$ sort. It is a very effecient sort when the data is already almost in order.

Binary insertion is a simple, effecient sort. Each new element is inserted into the previously sorted partial list by using a binary search to locate the place of insertion. As with all "divide and conquer" strategies, binary insertion is an $O(n\log_2 n)$ sort.

Shellsort is a powerful insertion sort. Studies of N randomly arranged data items show that elements, on the average, travel a distance of N/3 places to final, sorted positions. In 1959, Donald Shell used this fact in his sort algorithm, which initially moves data over large distances, approximately equal to N/3. This method has the tendency, on the average, of moving elements closer to their final location very early in the sort process.

Shell partitioned the list to be sorted into k "chains", each approximately 1/3 the size of the total list. Each chain was "sorted" using an exchange (bubbl[illegible] .. The algorithm shown here, somewhat improved, uses a linear insertion for each chain. The first chain consists of elements 1, k+1, 2k+1, etc. The second chain consists of elements 2, k+2, 2k+2, etc., and the jth chain (j<=k) consists of elements j, j+k, j+2k, etc. Thus, in the initial set of passes, elements are separated by a distance of k. K is then decremented to approximately k/3 and the process is repeated. When k is one, the list should be nearly sorted, and a final linear insertion sort of the entire list occurs. A linear sort at the end is faster than a binary sort because, although binary methods are better for randomly sorted large lists, a linear sort is faster if the number of comparisons is small or if the list is almost in order (see note).

There are various versions of this sort, each using different rules for partitioning the list. A less effecient algorithm uses N/2, N/4, etc. The one shown here is better than this version. Another version of the Shell sort uses decrements of ...,2^p-1, ...,31,15,7,3,1. The number of comparisions and moves is relatively low. Note that each pair of successive decrements is relatively prime.

note: It is interesting to note that for small n, an $O(n^2)$ sort is often more effecienct than an $O(n\log_2 n)$ sort because their simplicity requires little over[illegible] except comparisons and passes, making the constant of proportionality relatively small.

Dromey, R.G., How To Solve It By Computer, Prentice-Hall International, Englewood Cliffs, New Jersey, 1982. pp. 209-226.
Rhoads, Samuel E., Advanced Placement Computer Science, Addison - Wesley Publishing Company, Inc., Menlo Park, California, 1986. pp. 134-137, 159-160.

```
Program Sorts(input,output);
{Joyce Bernstein}
{Feb. 22, 1988}
{Program which demostrates an improved bubblesort, quicksort, linear}
{insertion sort, binary insertion sort, and shellsort}
{copy for educational use only}
const
  maxnum = 2000;
type
  list = array [1..maxnum] of integer;
var
  n,                      {size of array}
  choice:integer;         {menu selection}
  a,b:list;         {unsorted and sorted lists}

{******************procedure makelist**********************************}
procedure makelist(var a:list; n:integer);
{generates an array of size n of random numbers greater than or equal}
{zero and less than 1000}
var
  i:integer;
begin
  for i := 1 to n do
    a[i] := random(1000);
end;
{******************procedure showlist**********************************}
procedure showlist(a:list;n:integer);
{displays array in rows of 10 elements}
var
  i:integer;
begin
  for i := 1 to n do
  begin
    write(a[i]:4,'  ');
    if i mod 10 = 0 then writeln;
  end;
  writeln;
end;
{******************procedure insertion*********************************}
procedure insertion(var a:list;n:integer);
var
  i,       {list index for move once place is found}
  j,       {index for array item begin placed}
  place,   {eventually, index of first item smaller than item to insert}
  temp:   integer;      {value of item being inserted}
  found:  boolean;      {flag for location of insertion spot}
begin
  for j := 2 to n do
    begin
      temp := a[j];        {first j - 1 elements already sorted}
      place := j -1;
      found := false;
      while (place > 0 ) and not found do
        if a[place] > temp
          then place := place - 1
          else  found := true;
        for i := j - 1  downto place + 1 do
          a[i + 1] := a[i];
        a[place + 1] := temp
      end
    end;

{*****************procedure binaryinsertion*************************}
procedure binaryinsertion(var a:list; n:integer);
var
  i,              {list index for move once place is found}
  j,              {index of item being inserted}
  top,bottom,middle,    {section boundaries and center}
  temp:  integer;
begin
  for j := 2 to n do
    begin
      temp := a[j];
      top := 0;
      bottom := j;
      repeat
        middle := (top + bottom) div 2;
        if a[middle] <= temp
          then top := middle
          else  bottom := middle;
      until top + 1 = bottom;
     for i := j - 1 downto bottom do
      a[i + 1] := a[i];
     a[bottom] := temp
  end
end;
{*****************procedure shellsort********************************}
procedure shellsort(var a:list; n:integer);
var
  i,    {index used for mass shifting of chain items}
  j,    {some multiple of k, plus m --- index of item being inserted}
  k,    {becomes approx 1/3 size of array, size of first increment}
  m,    {loop  marker for each of the k chains}
  place, {index if item in chain smaller than insertion spot}
  temp : integer;   {item being inserted}
  found: boolean;
  begin
    k := 1;
      while (3*k +1) < (n div 3) do
        k := 3*k +1;        {sets size of increment based on n}
      repeat
        for m := 1 to k do   {for each of the k chains}
          begin
            j := k + m;
          while j <= n do

           begin
            temp := a[j];  {element to be sorted}
            i := j - k;
             found := false;
```

BEST COPY AVAILABLE

```
            while not found and (i > 0) do
              if a[i] > temp
                 then i := i - k
                 else found := true;
             place := i + k; {where item is being inserted}
             i := j - k;
             while i >= place do
              begin   {move rest of chain down}
                 a[i +k] := a[i];
                 i := i - k;
              end;
           a[place] := temp;
            j := j + k;
          end {while}
        end; {for}
     k := (k - 1) div 3    {next increment size}
     until k < 1
     end;

{*****************procedure quicksort********************************}
procedure quicksort(var a:list; n:integer);
procedure partition(r,s:integer;var j : integer);
var
  i : integer;
  temp: integer;
begin
  i := r + 1;   {index one past pivot}
  j := s;       {last index in swap range}
  repeat
    while (a[i] <= a[r]) and (i < s) do  {find an element to swap}
      i := i + 1;
    while (a[j] >= a[r]) and (j > r) do  { ''   '' ''      ''  ''}
      j := j - 1;
    if i <j
      then
        begin                          {swap}
          temp := a[i];
          a[i] := a[j];
          a[j] := temp
        end;
  until i >= j;    {Swap finished for this pivot}
  temp := a[r];
  a[r] := a[j];
  a[j] := temp;   {puts pivot in place} {j returned to calling program}
end;
procedure sort(m,n:integer);
{recursive procedure which redivides the sort field}
var
  j : integer;
  temp : integer;
begin
  if n - m > 1      {provides the halting case}
    then
      begin
        partition (m,n,j); {receives  the next dividing point, j}
```

```
          sort(m,i - 1);
          sort(i + 1,n);
        end
    else if (n - m = 1) and (a[m] > a[n])
      then
        begin  {do a final swap}
          temp := a[m];
          a[m] := a[n];
          a[n] := temp;
        end
  end;
  begin
    sort(1,n)
  end;
{* * * * * * * * * procedure bubble* * * * * * * * * * * * * * * * }
procedure bubblesort(var a:list;n:integer);
var
  i,            {index of items being compared}
  j,            {loop variable limit}
  lastswitch,   {flag for order in the data}
  temp :        {swap temp} integer;
begin
  j := n - 1;
    repeat
      lastswitch := 1;
      for i := 1 to j do
        if a[i] > a [i + 1]
        then
          begin             {swap}
            temp := a[i];
            a[i] := a[i + 1];
            a[i + 1] := temp;
            lastswitch := i;
          end;
      j := lastswitch - 1; {end of unsorted data}
      until lastswitch = 1
end;
{* * * * * * * * *procedure selection* * * * * * * * * * * * * * * }
procedure selection(var a:list;n:integer);
var
  i,j,s,k:integer;
begin
  for j := 1 to n - 1 do
  {generates a[j] thru a[n]}
  begin
    k := j;
    {k carrirs smallest element in decreasing block}
    s := a[j];
    for i := j + 1 to n do
      begin
        if a[i] < s then
          begin
            s := a[i];
            k := i;
          end;
        end;
      s := a[j];
      a[j] := a[k];
      a[k] := s
    end;
  end;
  {* * * * * * * * * procedure heap* * * * * * * * * * * * * * * * * * }
   Procedure heap(var a:list;n:integer);
    var
      i,   {index}
      x:integer;     {temp for array element}


  procedure fixheap(var a:list;top,bottom:integer);
  {assumes items top + 1 to bottom are a heap}
  {returns items top to bottom in a heap}
  var
   i,   {index}
   x:integer;     {temp for array element}
  begin
    i := 2 * top;
    if i <= bottom then
      begin
        if i < bottom then if a[i] <a[i + 1] then i := i +1;{largest child}
        if a[top] < a[i] then
          begin
            x := a[top];
            a[top] := a[i];
            a[i] := x;   {heap is good except at i}
            fixheap(a,i,bottom)
          end;
      end;
    end;

  begin{heap}
    for i := n div 2 downto 1 do fixheap(a,i,n);
    for i := n downto 2 do
      begin
        x := a[1];
        a[1]:= a[i];
        a[i] := x;
        fixheap(a,1,i-1)
      end;
  end;

  {* * * * * * * * * Main Program* * * * * * * * * * * * * * * * * * }
  begin
  choice := 0;
  Writeln('This program sorts any number of items from 2 to ',maxnum);
  Writeln('Enter the number to be sorted');
  readln(n);
  makelist(a,n);
  showlist(a,n);
  while choice <> 8 do
  begin
```

```
repeat
  Writeln('Please choose a sort.');
  writeln('  1:  bubblesort(improved)');
  writeln('  2:  quicksort');
  writeln('  3:  linear insertion');
  writeln('  4:  binary insertion');
  writeln('  5:  Shell');
  writeln('  6:  selection');
  writeln('  7:  heap');
  writeln('  8:  end');
  readln(choice);
until (choice >0) and (choice < 9);
case choice of
  1:  begin
        b := a;
        bubblesort(b,n);
        showlist(b,n);
      end;
  2:  begin
        b := a;
        quicksort(b,n);
        showlist(b,n);
      end;
  3:  begin
        b := a;
        insertion(b,n);
        showlist(b,n);
      end;
  4:  begin
        b := a;
        binaryinsertion(b,n);
        showlist(b,n);
      end;
  5:  begin
        b := a;
        shellsort(b,n);
        showlist(b,n)
      end;
  6:  begin
        b := a;
        selection(b,n);
        showlist(b,n);
      end;
  7:  begin
        b := a;
        heap(b,n);
        showlist(b,n);
      end;
end;
end;
end.
```