```
% Read and echo names; count them.

var count : int := 0

% How many names so far?

put "Please type a series of names."
put "Tell me ’quit’ at the end"

loop
	put "Next name? " ..
	var person : string
	get person
	exit when person = "quit"
	put "Hi, ", person
	count := count + 1
end loop

put "I greeted ", count, " people."
```

Let’s use the name-entering program.

`turingx example1.t`

Please type a series of names.
Tell me ’quit’ at the end
Next name? **`John`**
Hi, John
Next name? **`Joan`**
Hi, Joan
Next name? **`Susan`**
Hi, Susan
Next name? **`Jim`**
Hi, Jim
Next name? **`John`**
Hi, John
Next name? **`quit`**
I greeted 5 people.

How is the end of an input string determined? This is a bit of a problem. If I type **`John Jones`***, is that one string or two? Let’s see.*

`turingx example1.t`

Please type a series of names.
Tell me ’quit’ at the end
Next name? **`John Jones`**
Hi, John
Next name? Hi, Jones
Next name? **`"John Jones"`**
Hi, John Jones
Next name? **`quit`**
I greeted 3 people.

So, we see that, for Turing, typing **`John Jones`** *is* two *strings, while* **`"John Jones"`** *is one. These* conventions *are rather arbitrary and must be remembered.*

```
% Read and echo names; count them.

var count : int := 0

% How many names so far?

put "Please type a series of names."
put "Tell me ’quit’ or ’Quit’ at the end"

loop
        put "Next name? " ..
        var person : string
        get person
        exit when person = "quit" or person = "Quit"
        put "Hi, ", person
        count := count + 1
end loop

put "I greeted ", count, " people."
```

turingx example1a.t

Please type a series of names.
Tell me ’quit’ or ’Quit’ at the end
Next name? **Eugene**
Hi, Eugene
Next name? **Laura**
Hi, Laura
Next name? **Susan**
Hi, Susan
Next name? **Brian**
Hi, Brian
Next name? **Quit**
I greeted 4 people.

turingx example1a.t

Please type a series of names.
Tell me ’quit’ or ’Quit’ at the end
Next name? **Eugene**
Hi, Eugene
Next name? **Susan**
Hi, Susan
Next name? **Brian**
Hi, Brian
Next name? **Laura**
Hi, Laura
Next name? **qUit**
Hi, qUit
Next name? **quit**
I greeted 5 people.

```
% A declaration associates a "name" with one or more values.
% A "variable" declaration means that the value associated
% with name can change, while a "const" declaration means
% that the value stays fixed.


% The following declarations are all declare that a
% "variable" can only take on integral values.  So these
% variables are of "type" integer


var x: int
var y:      int
var z    : int


% We can declare that a variable is a string, or boolean
% or real.

var s: string
var b: boolean
var r: real

% We can also give variables an initial value like so.

var i: int := 0
var happy: string := "I am happy!"
var f: boolean := false

% Some constant declarations follow.
% They must be initialised (why?).

var pi: real := 3.14159265


put pi
put happy

% What would happen if we execute this program?
```

```
% These rates are in percent
const GSTrate: real := 7
const PSTrate: real := 8

const GSTfactor := GSTrate / 100
const PSTfactor := PSTrate / 100

var price, GST, PST: real

% Get the retail price
put "price?  " ..
get price

% Compute those darned taxes
GST := price * GSTfactor
PST := price * PSTfactor

%
% The following makes use of formatted "put"
% statements.  The command
%
%     put A : n
%
% means to put the variable A so that it occupies
% n columns.
%
% If r is a "real", then we can also say how
% many decimal places it should occupy after
% a second colon.  So
%
%     put r : 10 : 3
%
% says to print r using 10 columns and 3 decimal
% places.  One extra column is taken up by a "."
%
put "Before tax price " : 20, "$", price : 8 : 2
put "GST" : 20, "$", GST : 8 : 2
put "PST" : 20, "$", PST : 8 : 2
put "After tax price " : 20, "$", price+GST+PST : 8 : 2
```

```
turingx gst.t
price?  1000
Before tax price    $ 1000.00
GST                 $   70.00
PST                 $   80.00
After tax price     $ 1150.00
0123456789012345678901234567890123456789012345678
```

Look above and see if the formatting conventions hold

```
turingx gst.t
price?  250.01
Before tax price    $  250.01
GST                 $   17.50
PST                 $   20.00
After tax price     $  287.51
```

Is exact? NO!

```
hoc
a = 250
a*0.08
    20
a = 250.01
a*0.08
    20.0008
a*0.07
    17.5007
```

```
% Turing is a "sequential" programming language.
% So is Pascal, FORTRAN, C, BASIC, etc.

% This means that each instruction executed one
% after another, in sequence.  The same holds
% for loops.

% In the following, for example,

var a,b: real

a := 3.4
b := a + 1.0

put a
put b
put a+b

% the assignment and "put" statements are
% ALWAYS executed exactly in the order given.


% What is the output of this program?

% Examples of non-sequential programming:
%     * spread sheets
%     * parallel programming languages
%     * window systems software
```

```
% Repetition.  Are computers ever good at this!

% If I want to say "hello world" many many times,
% I could type

put "hello world"
put "hello world"
put "hello world"
put "hello world"
put "hello world"
put "hello world"
put "hello world"
put "hello world"
put "hello world"
%       .
%       .
%       .
%
%       OR

% I could write

loop
        put "hello world"
end loop

% When will the loop end?
```

```
% Mechanics of a loop statement.

% First, you "declare" that a loop is to begin:

loop

%       Then you put a regular Turing code in here,
%       which is executed sequentially.

%       Finally, you declare the end of the loop.

end loop


% The program will cycle through the Turing code
% within the "body" of the loop until it can escape.


% Terminating a loop:


var i: int := 10;
loop
        exit when i = 0
        put i
        i := i - 1
end loop
put i

% What is the last value printed?
```

```
% Analysing loops

var i: int := 10;
loop
        exit when i = 0
        put i
        i := i - 1
end loop
put i
```

```
% Does this produce a different output?
```

```
var i: int := 10;
loop
        put i
        exit when i = 0
        i := i - 1
end loop
put i
```

```
% Loops can appear within loops
% The following computes
%     1ˆ1 + 2ˆ2 + 3ˆ3 + ... + nˆn

var n:    int;
var i,j:  int
var sum1: int := 0
var sum2: int := 0

put "Enter a positive whole number: " ..
get n
i := 1
loop
        exit when i>n
        j := 1
        sum2 := i;
        loop
                exit when j >= i
                sum2 := sum2*i
                j := j + 1
        end loop
        i := i+1
        sum1 := sum1 + sum2
end loop
put sum1
```

Let's watch how these numbers grow.

turingx loop4.t
Enter a positive whole number: **1**
1

turingx loop4.t
Enter a positive whole number: **2**
5

turingx loop4.t
Enter a positive whole number: **3**
32

turingx loop4.t
Enter a positive whole number: **4**
288

turingx loop4.t
Enter a positive whole number: **5**
3413

turingx loop4.t
Enter a positive whole number: **6**
50069

turingx loop4.t
Enter a positive whole number: **7**
873612

turingx loop4.t
Enter a positive whole number: **8**
17650828

turingx loop4.t
Enter a positive whole number: **9**
405071317

turingx loop4.t
Enter a positive whole number: **10**
Line 19 of loop4.t: Overflow in Integer expression.

```
% read two integers and print their sum.

var x,y: int

put "Please enter two whole numbers"
get x, y

put "And the answer is..."
put x + y
```

A sample run of "sumint". The program expects two integer input values, but I can enter them on one or two lines.

`turingx sumint.t`
Please enter two whole numbers
`2`
`3`
And the answer is...
5

`turingx sumint.t`
Please enter two whole numbers
`3 5`
And the answer is...
8

So far, so good. Now what happens if we enter really big integers?

`turingx sumint.t`
Please enter two whole numbers
`1111111111 2222222222`
Line 6 of sumint.t: Invalid integer input.

Now, that's not a very friendly response, is it? The fact is that they're integers, but they're just too big for Turing to represent them in a memory cell of a certain size.

Now, what if we enter a non-integral value?

`turingx sumint.t`
Please enter two whole numbers
`2 1.1`
Line 6 of sumint.t: Invalid integer input.

```
% read two integers and print their sum a little differently.

var x,y: int

put "Please enter two whole numbers"
get x, y

put "And the answer is ", x + y
```

This exercises the "suminta" program, which prints the result on the same line as the text describing it.

```
turingx suminta.t
```
Please enter two whole numbers
`2`
`3`
And the answer is 5

```
turingx suminta.t
```
Please enter two whole numbers
`5 4`
And the answer is 9

```
turingx suminta.t
```
Please enter two whole numbers
`17 13`
And the answer is 30

```
% read two "real" numbers and print their sum.

var x,y: real

put "Please enter two numbers each possibly with a decimal point"
get x, y

put "And the answer is ", x + y
```

Let’s exercise the "real" valued adding program.

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`1.1 2.2`
And the answer is 3.3

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`0.7 0.3`
And the answer is 1

Looks good. Now try some negative numbers.

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`-0.55 0.45`
And the answer is -0.1

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`-1 +1`
And the answer is 0

We can write fractional quantities without a leading zero

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`1.1 .1`
And the answer is 1.2

But there are some things we can’t get away with!

`turingx sumreal.t`
Please enter two numbers each possibly with a decimal point
`1.1 .1.`
Line 6 of sumreal.t: Invalid real number input.

```
% Choosing between two possibilities.

var age : int

put "How old are you?  I won’t laugh!  " ..
get age

if age < 0 then
        put "Umm, ages are usually positive."
        age := -age
end if

if age > 30 then
        put "Ohmygod.  You’re ancient!"
else
        put "You’re cool."
end if
```

`turingx cond.t`
How old are you? I won't laugh! **`19`**
You're cool.

`turingx cond.t`
How old are you? I won't laugh! **`-8`**
Umm, ages are usually positive.
You're cool.

`turingx cond.t`
How old are you? I won't laugh! **`31`**
Ohmygod. You're ancient!

`turingx cond.t`
How old are you? I won't laugh! **`-45`**
Umm, ages are usually positive.
Ohmygod. You're ancient!

```
% Three-way selection using "if-then-else"

var age : int

put "How old are you?  I won’t laugh!  " ..
get age

if age < 0 then
        put "Umm, ages are usually positive."
        age := -age
end if

if age > 30 then
        put "Ohmygod.  You’re ancient!"
    else if age < 16 then
        put "Come back when you’ve grown up, kid."
    else
        put "You’re cool."
    end if
end if
```

```
% Three-way selection using "if-then-else"

var age : int

put "How old are you?  I won’t laugh!  " ..
get age

if age < 0 then
        put "Umm, ages are usually positive."
        age := -age
end if

if age > 30 then
        put "Ohmygod.  You’re ancient!"
elsif age < 16 then
        put "Come back when you’ve grown up, kid."
else
        put "You’re cool."
end if

%
% elsif is a contraction of "else -- if" and allows
% you to more clearly group together a set of options.
%
```

```
%
% The "for" loop.  Another kind of loop construct.
%

% Do something 10 times.

for : 1..10
        put "Help!  I'm trapped in here."
end for

%
% This loop has the same effect.
%
for i: 1..10
        put "Help!  I'm trapped in here."
end for
```

```
%
% The "for" loop.  Another kind of loop construct.
%

%
% Rather than writing the following loop
%
var i: int := 0
loop
        exit when i = 10
        put i
        i := i + 1
end loop
put i

% an alternative program is

for i: 0..10
        put i
end for

% Notes:
%       * i does not need to be declared because
%         Turing knows it must be an integer.
%       * a for loop is more compact than a regular
%         loop, but it is not as general.
```

```
%
% The "for" loop.  Another kind of loop construct.
%

%
% Rather than writing the following loop
%
var i: int := 10
loop
        exit when i = 0
        put i
        i := i - 1
end loop
put i

% an alternative program is

for decreasing i: 10..0
        put i
end for
```

```
% For loops can appear within other loops
% The following computes
%      1^1 + 2^2 + 3^3 + ... + n^n
% using for loops this time.

var n:    int;
var sum1: int := 0
var sum2: int := 0

put "Enter a positive whole number: " ..
get n
for i: 1..n
   sum2 := i;
   for j: 2..i
       sum2 := sum2*i
   end for
   sum1 := sum1 + sum2
end for
put sum1

% This "looks" more efficient than the loop
% version.  Is it?
```

How to write a successful loop without really trying.

RULE ONE:

A loop must contain at least one "exit" statement.

```
loop
        .
        exit when done
        .
end loop
```

RULE TWO:

The variables in the exit statement must have a value.

```
var done: boolean := false
var s: string

loop
        exit when done
        get s
        done := s="quit"
        put s
end loop
```

RULE THREE:

The exit condition is:

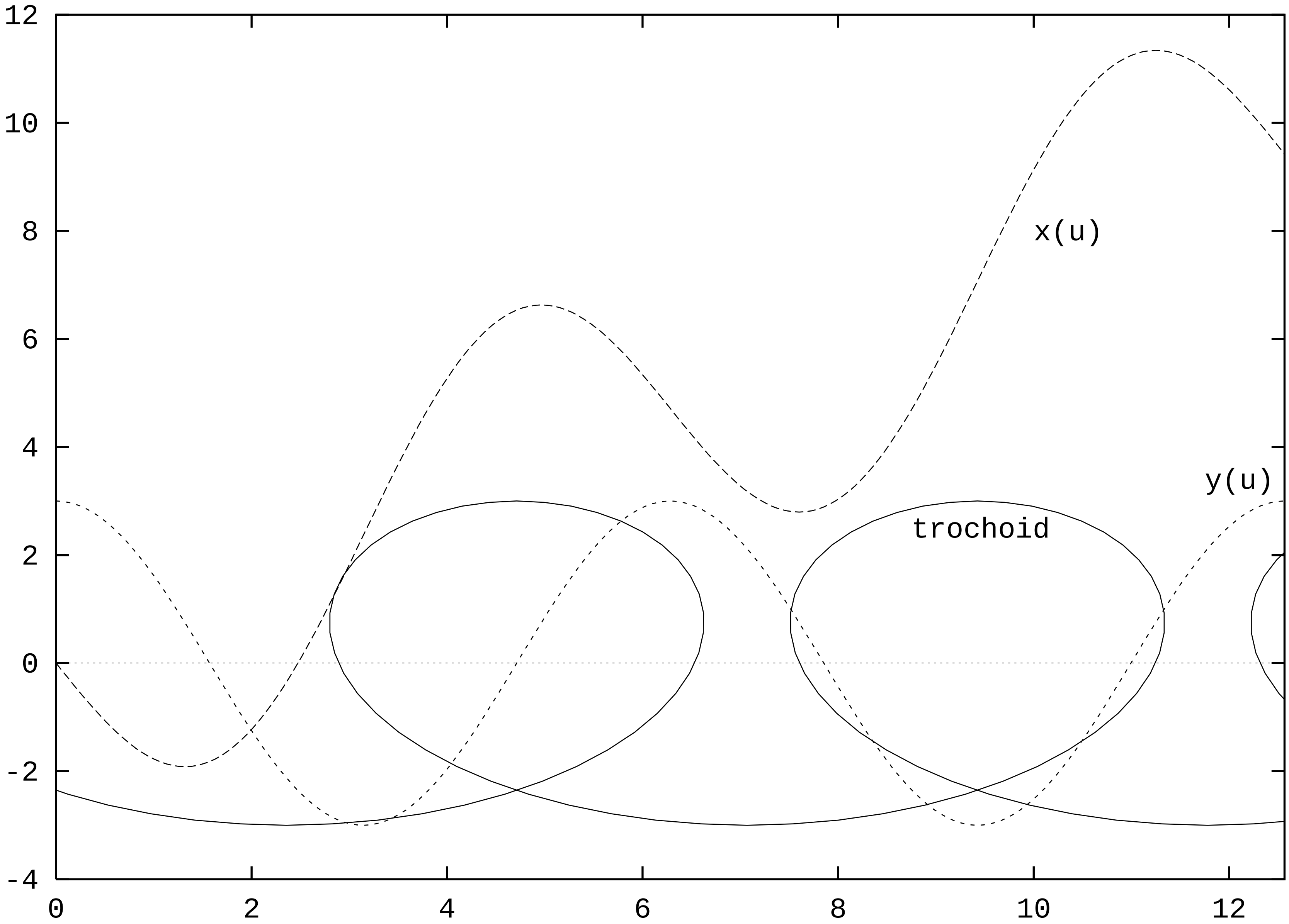
* usually "false" the first time through
* it must be possible to make it "true" in the loop.

```
var done: boolean := false
var s: string

loop
        exit when done
        get s
        done := 1=0
        put s
end loop
```

```
%
% A real example of a case statement.
% Don't worry if you don't understand the code just yet.
%
loop
        exit when t > 1
        case eval of
            label DirectTM:
                ComputeTM(t, basis, TM)
                EvaluateSpline(TM, pt, col)

            label DirectMP:
                px := t*t*t*A(0).x + t*t*A(1).x + t*A(2).x + A(3).x
                py := t*t*t*A(0).y + t*t*A(1).y + t*A(2).y + A(3).y
                PlotPoint(px,py,col)

            label Horner:
                px := (((A(0).x*t) + A(1).x)*t + A(2).x)*t + A(3).x
                py := (((A(0).y*t) + A(1).y)*t + A(2).y)*t + A(3).y
                PlotPoint(px,py,col)

            label ForwardDiff:
                PlotPoint(px,py,col)
                px   += D.x
                py   += D.y
                D.x  += DD.x
                D.y  += DD.y
                DD.x += DDD.x
                DD.y += DDD.y
        end case

        t += dt

end loop
```

```
put "Plotting a trochoid.  Slow way."

const pi: real := 3.141593
var x,y: real
var xn,yn: int
var u, r, a, du: real

setscreen("nocursor")
r  := 100
a  := 25
du := 0.01
u  := 0

x  := 0
y  := r

loop
        exit when u > 10*pi
        xn := round(x)
        yn := round(y)
        drawdot(xn,150+yn,1)
        u  += du
        x  := a*u - r*sin(u)
        y  := r*cos(u)
end loop
```

```
put "Plotting a trochoid. The fast way."

const pi: real := 3.141593
var x,y: real
var xn,yn: int
var u, r, a, du: real

setscreen("nocursor")
r  := 100
a  := 25
du := 0.01
u  := 0

x  := 0
y  := r

loop
        exit when u > 10*pi
        xn := round(x)
        yn := round(y)
        drawdot(xn,150+yn,1)
        u  += du
        x  := x + du*(a - y)
        y  := y + du*(x - a*u)
end loop
```



12
10
8
6
4
2
0
-2
-4
0
2
4
6
8
10
12
x(u)
y(u)
trochoid

```
%           Read and echo names; count them; check for uniqueness

        const N: int := 50        % maximum names
        var i, count, dups : int := 0
        var found : boolean
        var names : array 1 .. N of string

        put "Please type a series of names."
        put "Tell me 'quit' at the end"

        loop
            put "Next name? " ..
            var person : string
            get person
            exit when person = "quit" or i > N

        %   OK, so a name has been entered.  Let's see if it's
        %   already in the list.  This is called a "linear" search,
        %   because we're looking at each item in sequence.

            found := false
            i := 0;
            loop  % is person on the list already?
                exit when found or i = count
                i := i + 1
                found :=  (person = names(i))
            end loop

            if found then
                put "You know, there's already a ",
                    person, " on this list,"
                put "so I won't say hello to you again."
                dups := dups + 1
            else
                count := count + 1
                names(count) := person
                put "Hi, ", person, ".  I've added you to ",
                  "my list at position ", count
            end if
        end loop
        put "I greeted ", count, " people,"
        put "and there were ", dups, " duplicate names entered."
```

Let’s check for duplicate names

Please type a series of names.
Tell me ’quit’ at the end

Next name? **`John`**
Hi, John. I’ve added you to my list at position 1

Next name? **`Jim`**
Hi, Jim. I’ve added you to my list at position 2

Next name? **`Joan`**
Hi, Joan. I’ve added you to my list at position 3

Next name? **`John`**
You know, there’s already a John on this list,
so I won’t say hello to you again.

Next name? **`Joan`**
You know, there’s already a Joan on this list,
so I won’t say hello to you again.

Next name? **`Susan`**
Hi, Susan. I’ve added you to my list at position 4

Next name? **`quit`**
I greeted 4 people,
and there were 2 duplicate names entered.

```
% Strings.

var hi: string := "hello world"

% This puts a string.

put hi


% We can determine its length:

put "The string ‘"', hi, "’ contains ", length(hi), " characters."
% Read and echo names; count them.

var count : int := 0
var person : string

put "Please type a series of names."
put "Tell me ’quit’ at the end"

loop
	put "Next name? " ..
	get person
	exit when person = "quit"
	put "The string ‘"', person,
	     "’ contains ", length(person), " characters."
end loop
```

`turingx string1.t`

hello world
The string ‘hello world’ contains 11 characters.
Please type a series of names.
Tell me ’quit’ at the end

Next name? **`John`**
The string ‘John’ contains 4 characters.

Next name? **`"Eugenio Luciano Fiume"`**
The string ‘Eugenio Luciano Fiume’ contains 21 characters.

Next name? **`hello`**
The string ‘hello’ contains 5 characters.

Next name? **`quit`**

```
% Strings.  Concatenation

var sentence: string := ""
var word: string

put "Please type a sequence of words followed by ‘quit’"

%
% Why have I written the code this way?
%

loop
        get word
        exit when word = "quit"
        if sentence = "" then
                sentence := word
        else
                sentence := sentence + " " + word
        end if
end loop

if sentence = "" then
        put "Hey, you didn’t type a sentence!"
else
        sentence := sentence + "."
        put "The sentence you typed is:"
        put sentence
end if
```

`turingx string2.t`
Please type a sequence of words followed by ‘quit’
`quit`
Hey, you didn’t type a sentence!

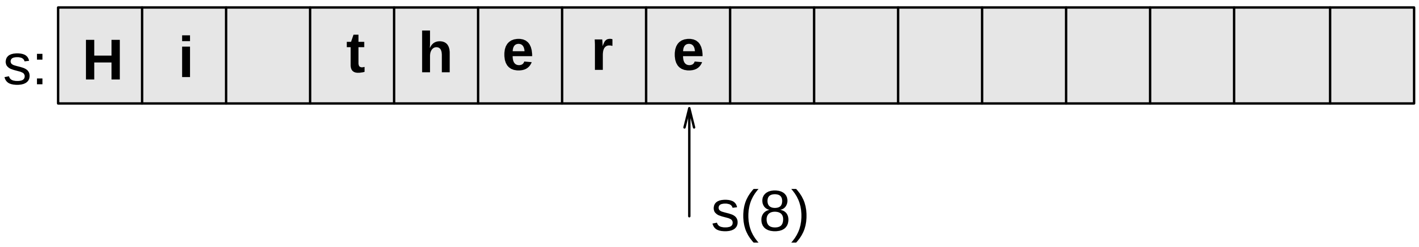
`turingx string2.t`
Please type a sequence of words followed by ‘quit’
`Hi there,`
`my name is`
`Eugene`
`quit`
The sentence you typed is:
Hi there, my name is Eugene.

`turingx string2.t`
Please type a sequence of words followed by ‘quit’
`I am`
`a happy`
`camper`
`quit`
The sentence you typed is:
I am a happy camper.

```
var i: int
var s: string


i := 23
s := "23"

%
% What’s the output?
%
put i
put s


%
% OK, smartie, so now what’s the output?
%
i := 20 + 3
s := "20" + "3"
```

```
%
% Substrings
%

var s: string := "Hello world"

put s
s (2) := "h"
put s
```

Strings: an array of characters




```
%
% Pieces of strings
%

var s: string := "hi there        "
var i: int := 4

put s(2)

put s(1..3)

put s(i..*)

put s(*-i .. *)

%
% Note:  you cannot do this:
%

s(2) = "no way"

%
% You can't even do this:
%

s(2) = "o"

%
% How would you change "hi there" to "ho there"?
% I.e., you want a string, say s2, that looks like s
% except in position 2.
%
```

```
%
%     Conditions evaluate to either true or false
%
%     They are used in if statements and for statements.
%
%
%     Some we know about already.
%

var i: int := 0;
var name :string := "foo"

var b: boolean


if i = 3 then
      % lotsa stuff here
endif

or

b :=  name = "Jim"

if b then ...

%
% Note: the "=" is a test for equality.
% It doesn't assign the value on the RHS to the LHS.
%

% Other boolean operations:

      i > 0
      i > 0 and i < 7
      i <= 8 or i >= 16
      i >= 8 and i >= 16              **

      (name = "Jim") and (age <=30 or age >= 20)
```

%

% ***String Comparisons***

%

% Just as numbers have an order, we can order strings.

% The order is ***lexicographic***, i.e., dictionary order.

%

if s < "n" then ...

%

% The following conditions are ***true***

%

"a" < "b" ... "y" < "z"

Is: "e" < "o" ????

"ab" < "ac"

"ab" < "aba"

"2" < "4"

Is: "2" < 4 ????

" " < "a"

"a " < "aa"

Ordering rules:

The way characters are encoded can sometimes depend on the machine and certainly depends on the continent you're on.

Some essential rules about the ordering of (English) characters:

- The letters "A", "B", ..., "Z" are together and in that order.
- The letters "a", "b", ..., "z" are together and in that order.
- The numerals "0", "1", ..., "9" are together and in that order.
- The ***blank*** character " " comes before all other printable characters.

What is not known:

- Is "a" before or after "A"?
- Is "0" before or after "0"?
- Is "@", or "%" or any other special character before or after "0"?

Nifty Operations:

Have a look at ***index***, ***ord*** and ***chr***.

The last word on for loops

Index variables

```
for i: 1..3
	put i
end for
```

- The name i is an ***index variable***.
- It should not be declared as a "var" before using it,
- and it does not exist after the "end for".

Turing gets angry if you

```
	for i: 1..3
		stuff
		i := 78
		stuff
	end for
```

within the for loop.

The last word on loops: eof and lines

If you don’t know how much input there is, you can test for a special ***end of file*** condition.

```
var foo: string

loop
        exit when eof
        get foo
        do something
end loop
```

This looks prety clean, but something messy can happen.

The semantics of "eof" in Turing is a bit confusing:

eof is true iff current file position (cfp) is at end of file

Thus

```
var v: int

if not eof then
    get v
end if
```

May still abort if the cfp is not at eof and if byte positions cfp..eof in the file are blanks.

See the "get skip" command in the book.

Line Oriented Input

```
var line: string

loop
        exit when eof
        get line : *
        do something
end loop


%
% Fixed length "tokens"
%
var token: string
loop
        exit when eof
        get token : 6
        put token
end loop
```

```
turingx eof.t
Hi there I am going to get confusing output.
Hi the
re I a
m goin
g to g
et con
fusing
 outpu
t.
```

```
abcdefghijklmnopqrstuvwxyz
abcdef
ghijkl
mnopqr
stuvwx
yz
```

```
%
%     Interchange a(i) and a(j)
%
var a: string:= "kcmgj"
var b: string
var i,j: int

put "enter a string"
get a

put "enter i and j"
get i, j

b := a(1..i-1) + a(j) + a(i+1..j-1) + a(i) + a(j+1..*)

put "original: ", a, ".  Exchanged i and j: ", b
```

```
turingx exchange.t
enter a string
hi there
enter i and j
Line 12 of exchange.t: Invalid integer input.

turingx exchange.t
enter a string
"hi there"
enter i and j
2 5
original: hi there.  Exchanged i and j: hh tiere


turingx exchange.t
enter a string
"Hi there"
enter i and j
1 2
original: Hi there.  Exchanged i and j: iH there
```

```
%
%   Array declarations
%
%   We want a fixed sequence of elements such as strings, reals or ints.
%
%   We could just define a long list of names such as:

var foo1, foo2, foo3, foo4, foo5, foo6: string

%
%   Then we have to explicitly refer to each "foo" to use it.
%   Programming languages instead allows them to be grouped into
%   an array.
%

var names: array 1..100 of string
var someReals: array -4..6 of real
var someInts: array 0..9 of int

%
%   Although it’s "natural" to use a subrange of 1..n
%   for an array, it is not necessary.
%

for i: -4..6
    get someReals(i)
end for

for i: 0..9
    get someInts(i)
end for
```

```
% Read and echo names; count them; check for uniqueness

const N: int := 50        % maximum names
var i, count, dups : int := 0
var found : boolean
var names : array 1 .. N of string

put "Please type a series of names."
put "Tell me 'quit' at the end"

loop
        put "Next name? " ..
        var person : string
        get person
        exit when person = "quit" or i > N

%       OK, so a name has been entered.  Let's see if it's
%       already in the list.  This is called a "linear" search,
%       because we're looking at each item in sequence.

        found := false
        i := 0;
        loop  % is person on the list already?
                exit when found or i = count
                i := i + 1
                found :=  (person = names(i))
        end loop

        if found then
                put "You know, there's already a ",
                        person, " on this list,"
                put "so I won't say hello to you again."
                dups := dups + 1
        else
                count := count + 1
                names(count) := person
                put "Hi, ", person, ".  I've added you to ",
                    "my list at position ", count
        end if
end loop
put "I greeted ", count, " people,"
put "and there were ", dups, " duplicate names entered."
```

Let’s check for duplicate names

Please type a series of names.
Tell me ’quit’ at the end

Next name? **`John`**
Hi, John. I’ve added you to my list at position 1

Next name? **`Jim`**
Hi, Jim. I’ve added you to my list at position 2

Next name? **`Joan`**
Hi, Joan. I’ve added you to my list at position 3

Next name? **`John`**
You know, there’s already a John on this list,
so I won’t say hello to you again.

Next name? **`Joan`**
You know, there’s already a Joan on this list,
so I won’t say hello to you again.

Next name? **`Susan`**
Hi, Susan. I’ve added you to my list at position 4

Next name? **`quit`**
I greeted 4 people,
and there were 2 duplicate names entered.

```
% Find the largest number given as input.

const small := -1000000
const N: int := 100             % maximum names

var i:      int  := 0
var maximum: real := small  % start max off very small

var numbers : array 1 .. N of real

put "Enter a list of numbers and I'll find the largest number"
loop
        get skip
        exit when eof or i >= N
        i := i + 1
        get numbers(i)
        if numbers(i) > maximum then
                maximum := numbers(i)
        end if
end loop

if i = 0 then
        put "Hey!  You didn't enter anything."
else
        if i >= N then
                put "We ran out of room to store all the numbers."
                put "The maximum is only based on the first ", N, " of them."
        end if
        put "The maximum is ", maximum
end if
```

```
% Exchange the maximum and minimum in a list.
const small := -1000000
const large :=  1000000
const     N := 10
var i: int  := 0
var maximum: real := small
var minimum: real := large
var imin, imax: int := 1 % indexes of min and max
var foo: real                  % a dummy variable
var numbers : array 1 .. N of real

put "Enter a list of numbers and I'll find the largest number"
loop
        get skip
        exit when eof or i >= N
        i := i + 1
        get numbers(i)
        if numbers(i) > maximum then
                maximum := numbers(i)
                imax    := i
        end if
        if numbers(i) < minimum then
                minimum := numbers(i)
                imin    := i
        end if
end loop

if i = 0 then
        put "Hey!  You didn't enter anything."
else
        if i >= N then
                put "We ran out of room to store all the numbers."
                put "The result is only based on the first ", N, " of them."
        end if
        put "The maximum is ", maximum
        put "The minimum is ", minimum
        foo := numbers(imin)
        numbers(imin) := numbers(imax)
        numbers(imax) := foo
        for j: 1..i
                put numbers(j)
        end for
end if
```

```
turingx array4.t
Enter a list of numbers and I’ll find the largest number
<cntl-D>
Hey!  You didn’t enter anything.

turingx array4.t
Enter a list of numbers and I’ll find the largest number
1
The maximum is 1
The minimum is 1
1

turingx array4.t
Enter a list of numbers and I’ll find the largest number
1 2 3 4 5
The maximum is 5
The minimum is 1
5
2
3
4
1

turingx array4.t
Enter a list of numbers and I’ll find the largest number
13 12 11 10 9 8 7 6 5 4 3 2 1 0
We ran out of room to store all the numbers.
The result is only based on the first 10 of them.
The maximum is 13
The minimum is 4
4
12
11
10
9
8
7
6
5
13
```

```
% Sorting: can be done by iterated maximum finding
%
% As a first step, find the maximum in a list and put it at
% the head of the list.

const small := -1000000
const N: int := 100                    % maximum names

var i: int  := 0
var imax: int  := 1                    % index of maximum
var maximum: real := small  % start max off very small
var foo: real                          % a dummy variable
var numbers : array 1 .. N of real

%
% First get the numbers.
%




%
% Find the maximum
%




%
% We have our maximum, so now put it at the head of the list.
%




%
% Output the (partially-sorted) list
%
put "The partially sorted list is now"
for j: 1..i
        put numbers(j)
end for
```

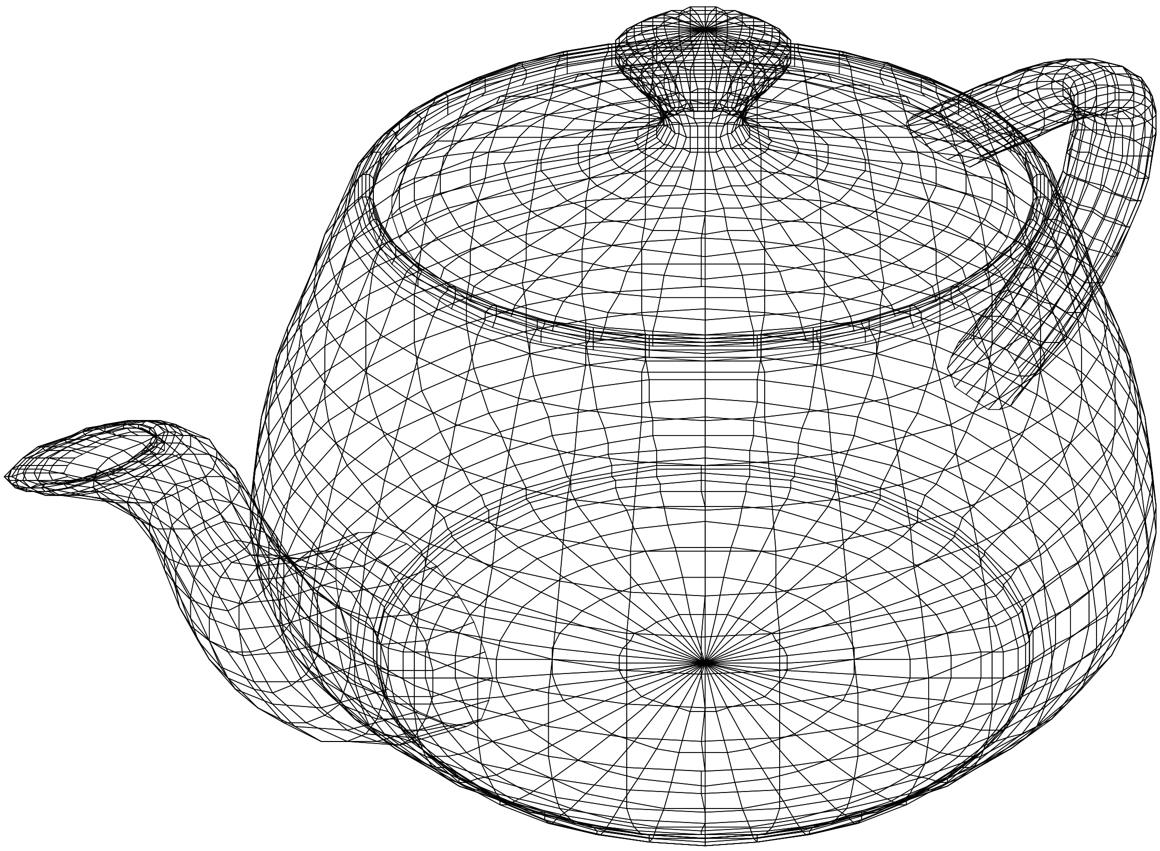
```
% Sorting: can be done by iterated maximum finding
% As a first step, find the maximum in a list and put it at
% the head of the list.

const small := -1000000
const N: int := 100                    % maximum names
var i: int  := 0
var imax: int  := 1                    % index of maximum
var maximum: real := small  % start max off very small
var foo: real                          % a dummy variable
var numbers : array 1 .. N of real

% First get the numbers.

put "Enter a list of numbers and I'll find the largest number"
loop
        get skip
        exit when eof or i >= N
        i := i + 1
        get numbers(i)
end loop

% Find the maximum

for j: 1..i
        if numbers(j) > maximum then
                maximum := numbers(j)
                imax    := j
        end if
end for

% We have our maximum, so now put it at the head of the list.

assert (i>0)
foo := numbers(1)
numbers(1) := numbers(imax)
numbers(imax) := foo

put "The partially sorted list is now"
for j: 1..i
        put numbers(j)
end for
```

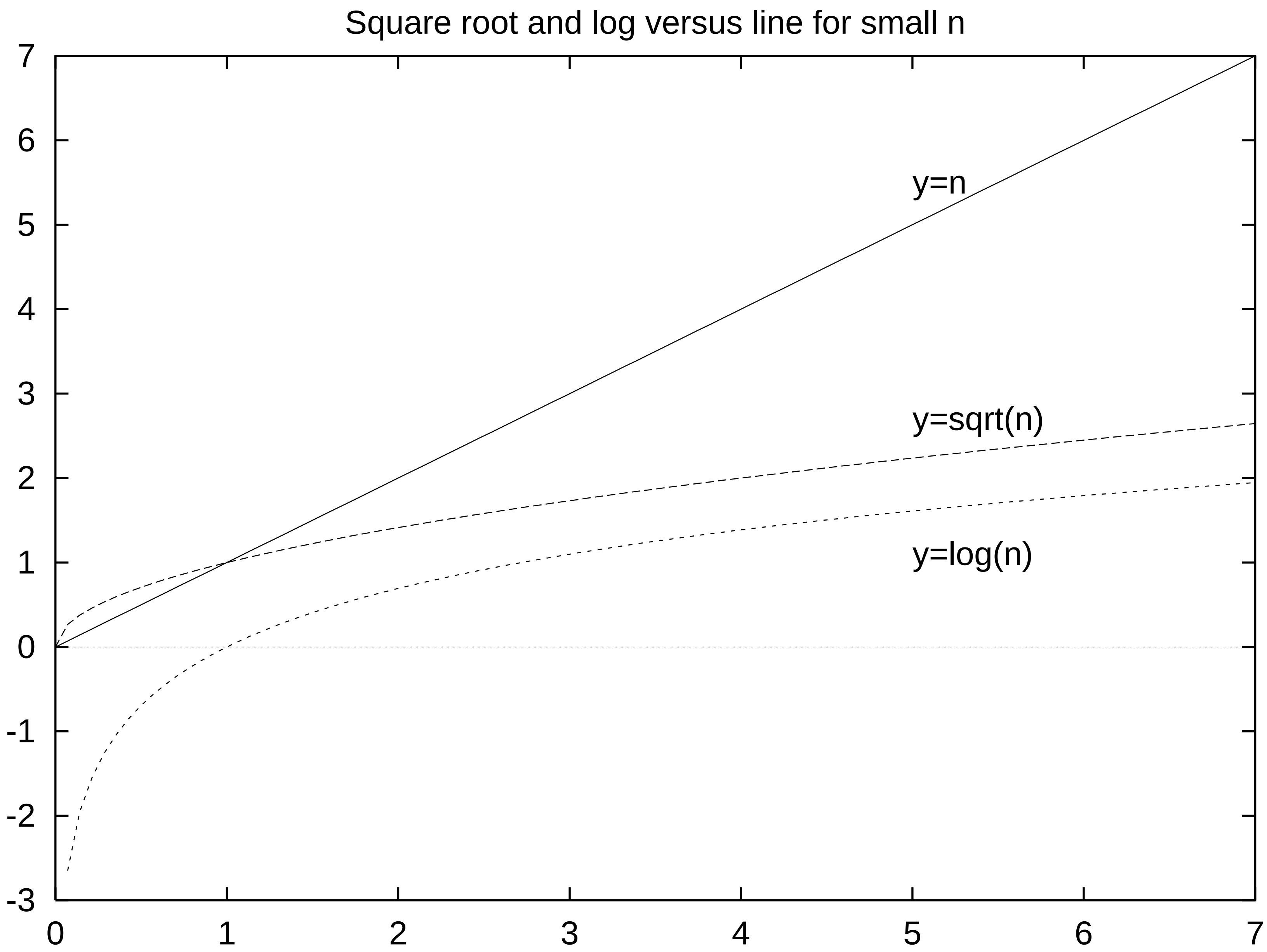
```
turingx max2.t
Enter a list of numbers and I'll find the largest number
<cntl-D>
Line 34 of max2.t: Assert condition is false.
```

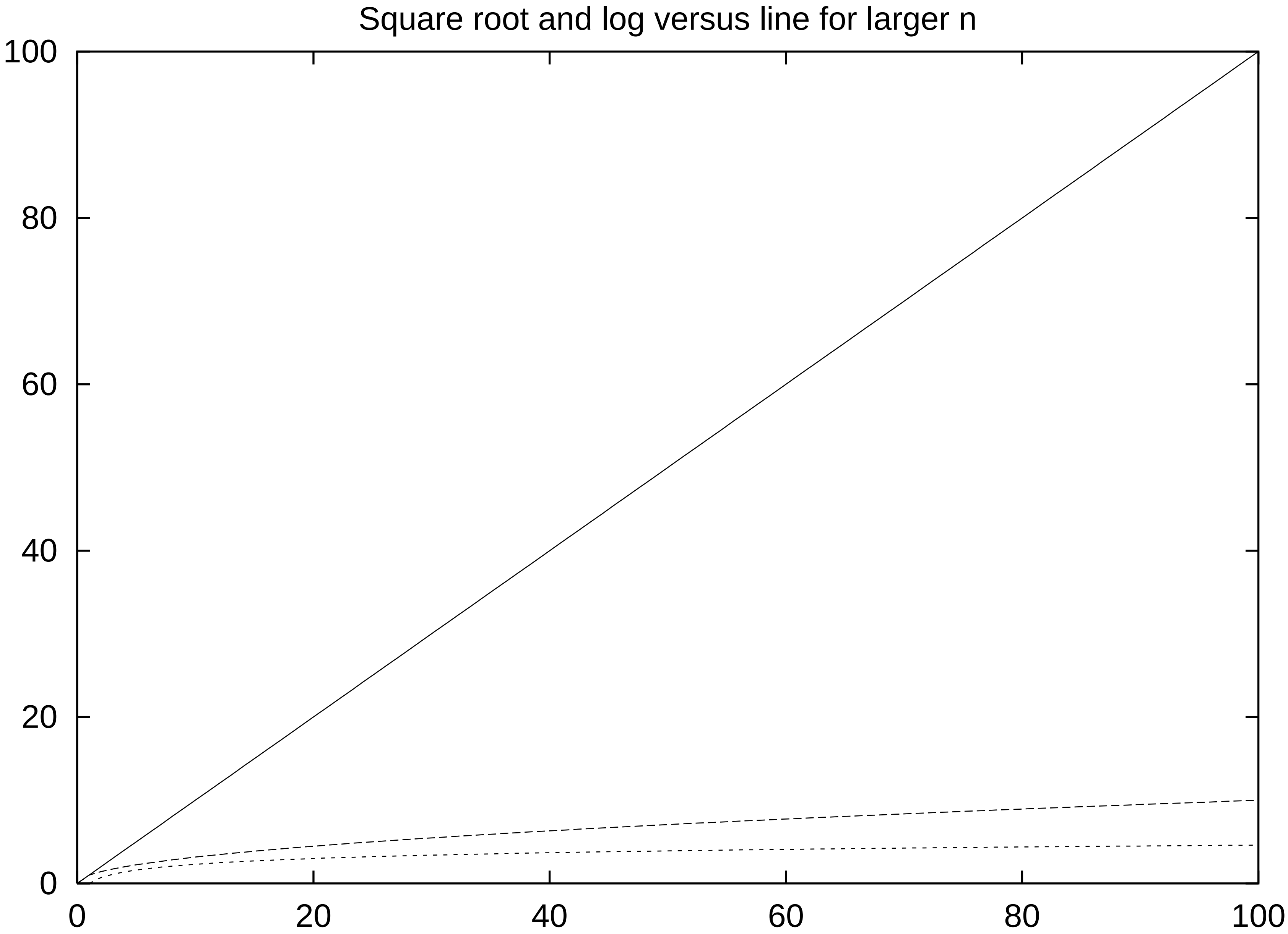
```
turingx max2.t
Enter a list of numbers and I'll find the largest number
8 4 3 7 4 3 12 34 1 2 34 4 3
The partially sorted list is now
34
4
3
7
4
3
12
8
1
2
34
4
3
```

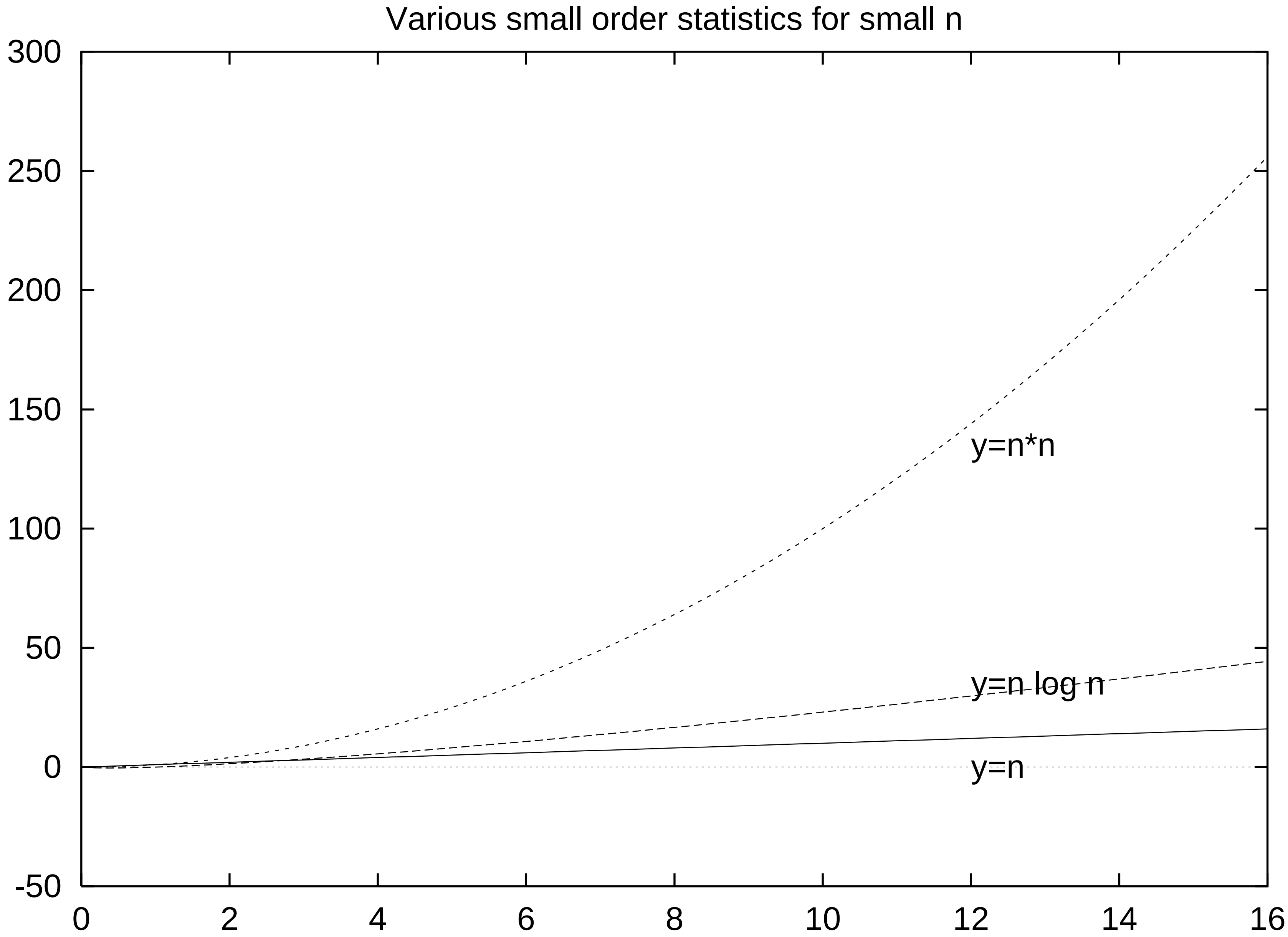
```
import var Plot in "Plot.t"
setscreen("nocursor")
setscreen("graphics:800;800")
setscreen("title: Bicubic Spline Plot")
setscreen("position: -300;-200");

var file:   int;                            % file number
var filename: string := "teapot.data"       % our data file name containing
                                            % matrix information
type matrix:
        array 0..3, 0..3 of real

var xmatrix, ymatrix, zmatrix: matrix

procedure GetMatrix(var mat: matrix)
        for i: 0..3
                for j: 0..3
                        get :file, mat(i,j)
                end for
        end for
end GetMatrix

%       Compute [sˆ3  sˆ2  s  1]M[tˆ3 tˆ2 t 1]+, where M is
%       BPB+ for basis B and observations P.
%       s,t are specific values in [0,1].

function poly(s,t: real, mat: matrix): real
        var T: array 0..3 of real
        var S: array 0..3 of real
        var sums: array 0..3 of real
        var sum: real

        T(3) := 1.0
        T(2) := t
        T(1) := t*t
        T(0) := T(1)*t
        S(3) := 1.0
        S(2) := s
        S(1) := s*s
        S(0) := S(1)*s
        for i: 0..3                     % first evaluate MT+
```

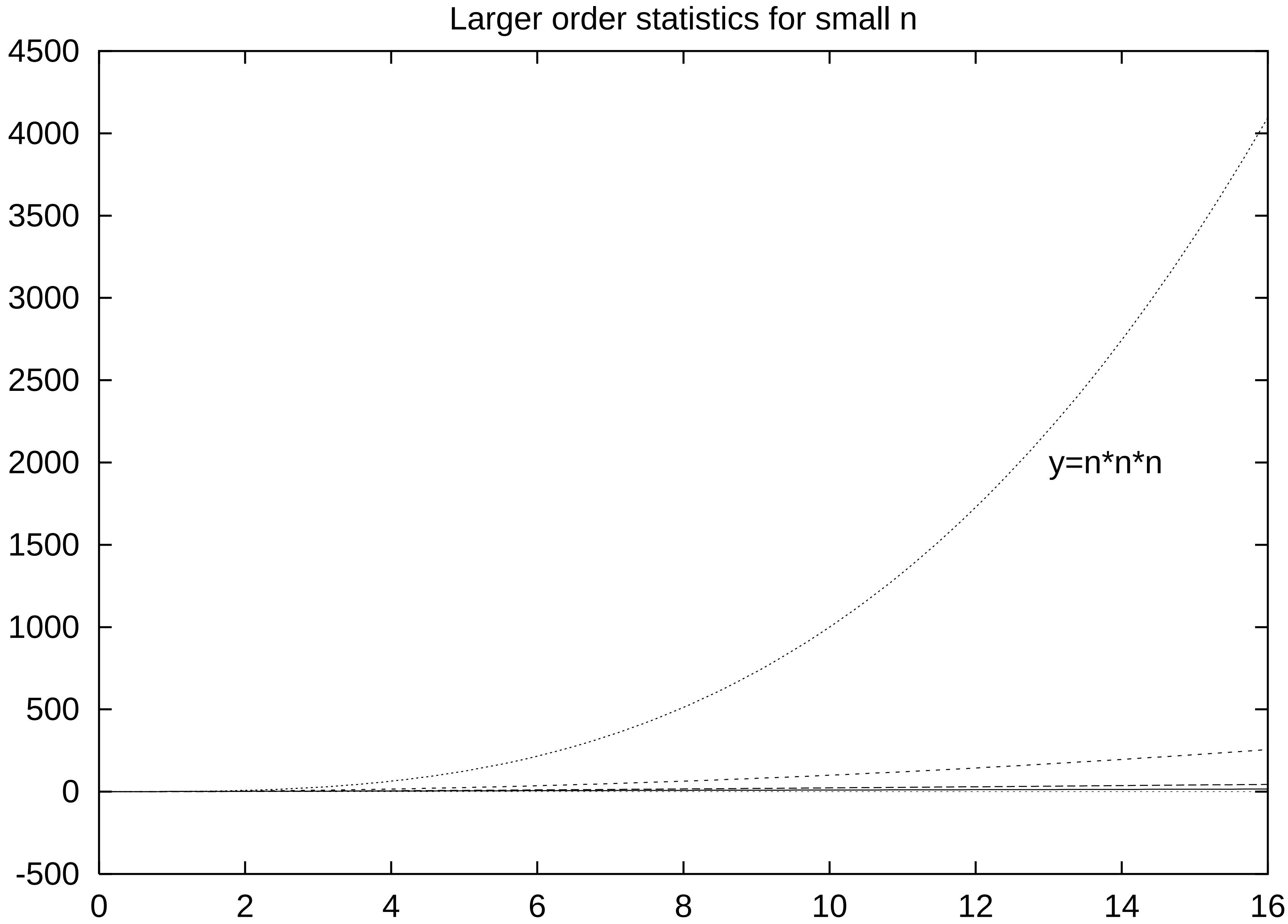
```
        sums(i) := 0.0
                for j: 0..3
                        sums(i) += T(j)*mat(j,i)
                end for
        end for
        sum := 0
        for i: 0..3                                     % now compute S*sums
                sum += S(i)*sums(i)
        end for
        result(sum)
end poly

function polyx(s,t: real): real
        result(poly(s,t,xmatrix))
end polyx

function polyy(s,t: real): real
        result(poly(s,t,ymatrix))
end polyy

function polyz(s,t: real): real
        result(poly(s,t,zmatrix))
end polyz

var number: int;

Plot.SetScale(100,100)
Plot.SetOffset(500,500)
Plot.SetColour(2)

open : file, filename, get
get : file, number
put "Plotting ", number," spline segments for each of x,y,z"

for i: 1..number
        GetMatrix(xmatrix)
        GetMatrix(ymatrix)
        GetMatrix(zmatrix)
        Plot.Plot3D(polyx,polyy,polyz, 0,1, 8, 0,1, 8)
end for
```

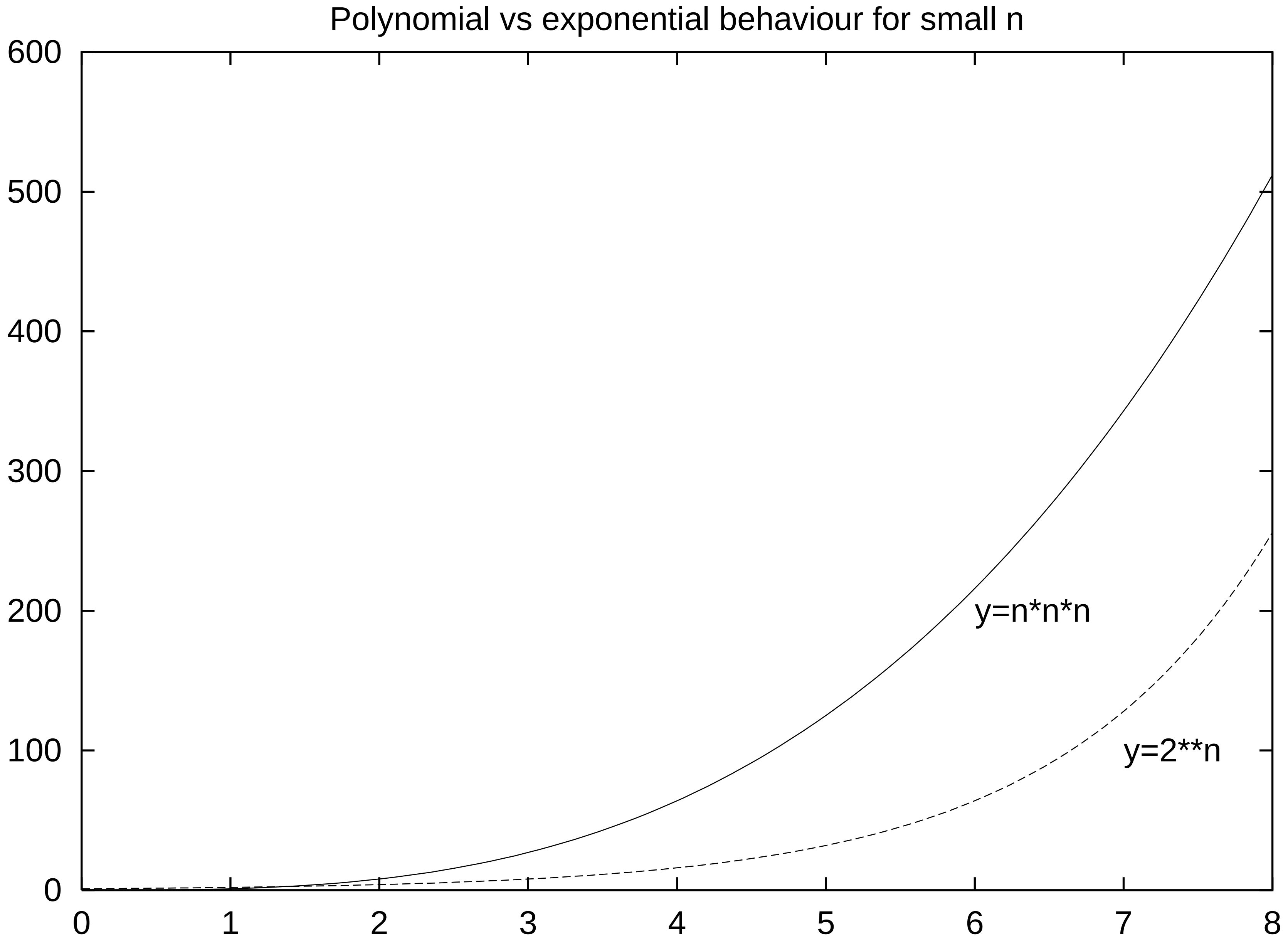



Square root and log versus line for small n
y=n
y=sqrt(n)
y=log(n)
7
6
5
4
3
2
1
0
-1
-2
-3
0
1
2
3
4
5
6
7

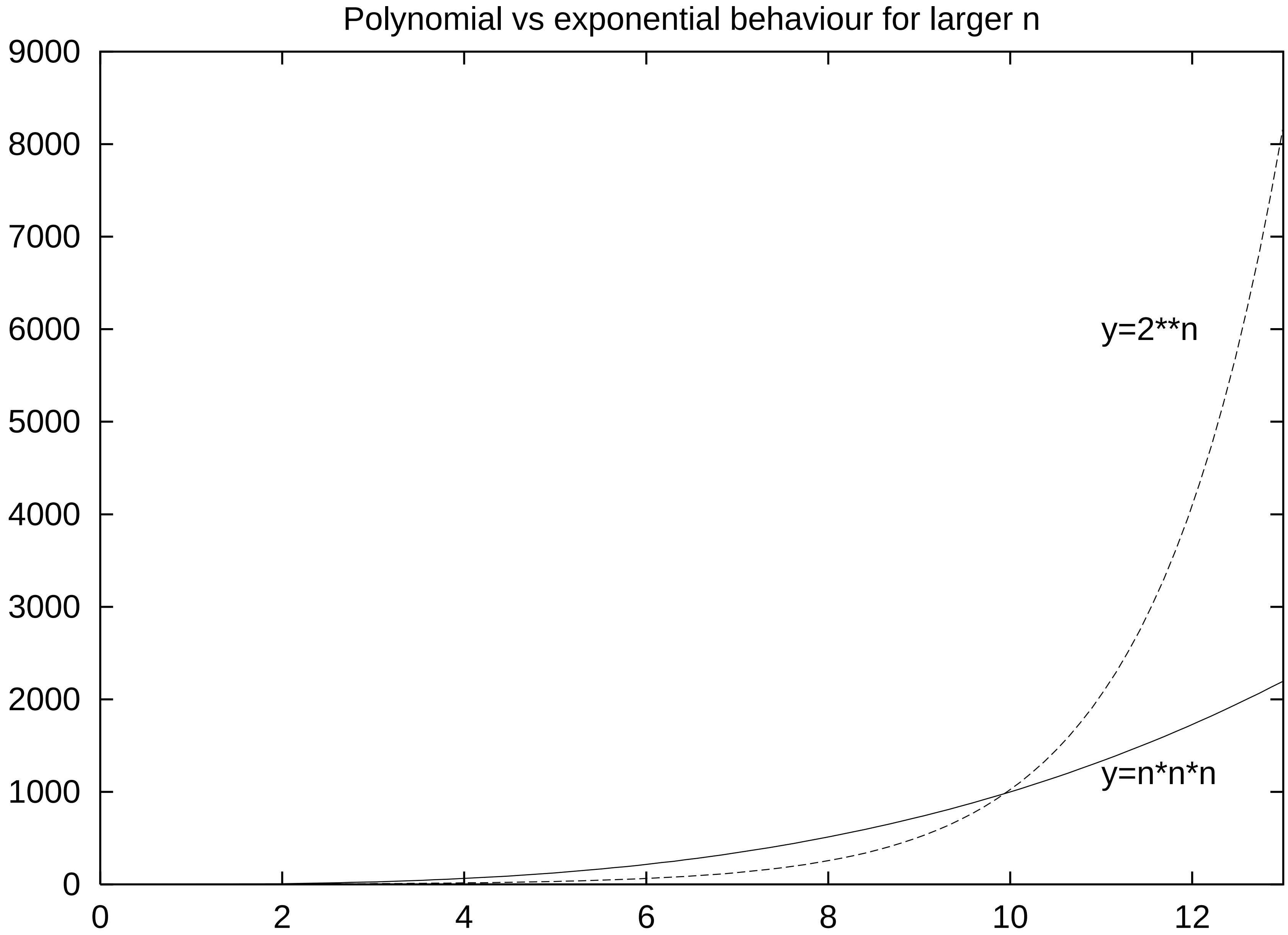

Square root and log versus line for larger n
100
80
60
40
20
0
0
20
40
60
80
100

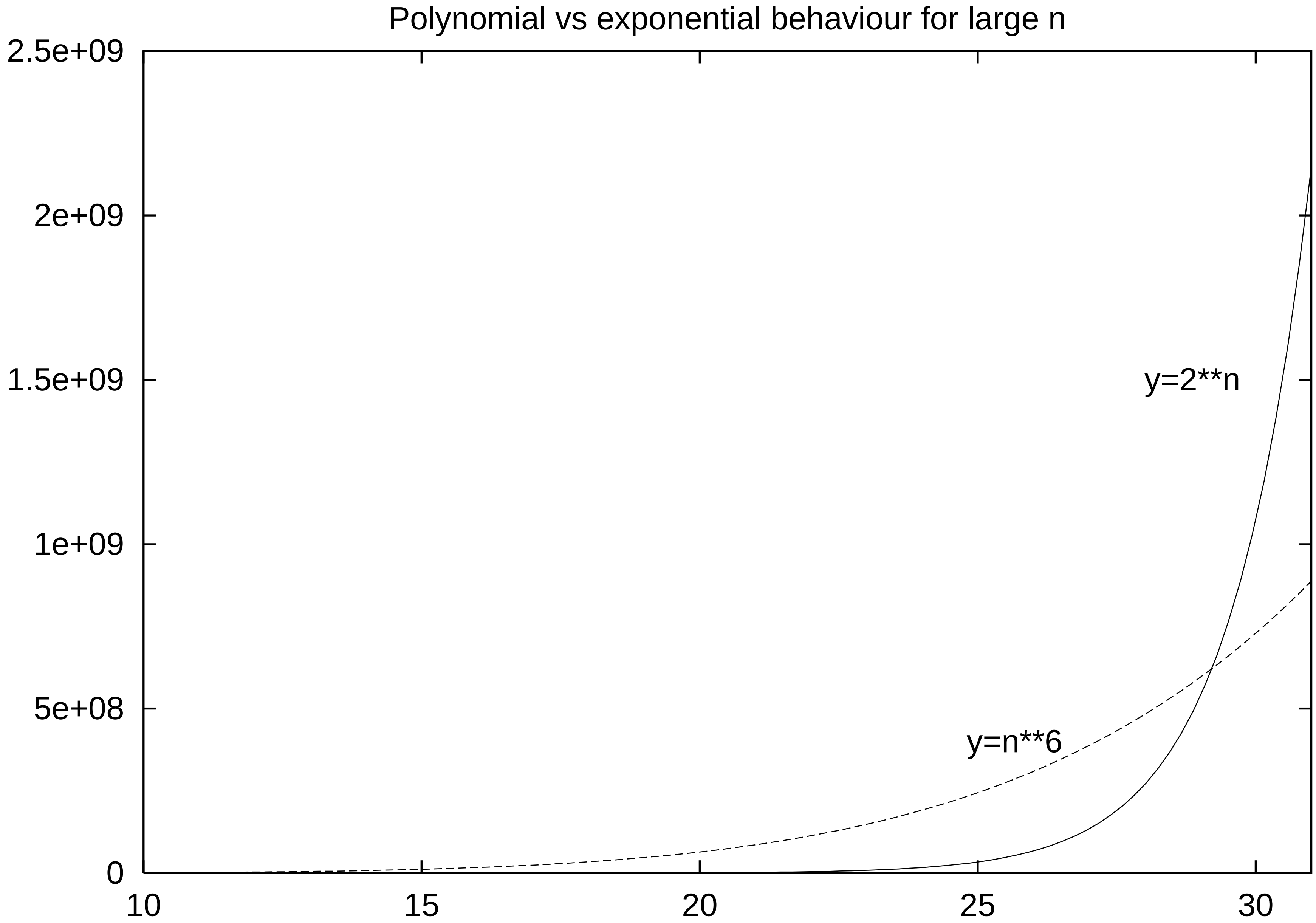

Various small order statistics for small n
300
250
200
150
100
50
0
-50
0
2
4
6
8
10
12
14
16
y=n*n
y=n log n
y=n


Larger order statistics for small n
4500
4000
3500
3000
2500
2000
1500
1000
500
0
-500
0
2
4
6
8
10
12
14
16
y=n*n*n



Polynomial vs exponential behaviour for small n
600
500
400
300
200
100
0
0
1
2
3
4
5
6
7
8
y=n*n*n
y=2**n


Polynomial vs exponential behaviour for larger n
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
12
y=2**n
y=n*n*n



Polynomial vs exponential behaviour for large n
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
10
15
20
25
30
y=2**n
y=n**6

%

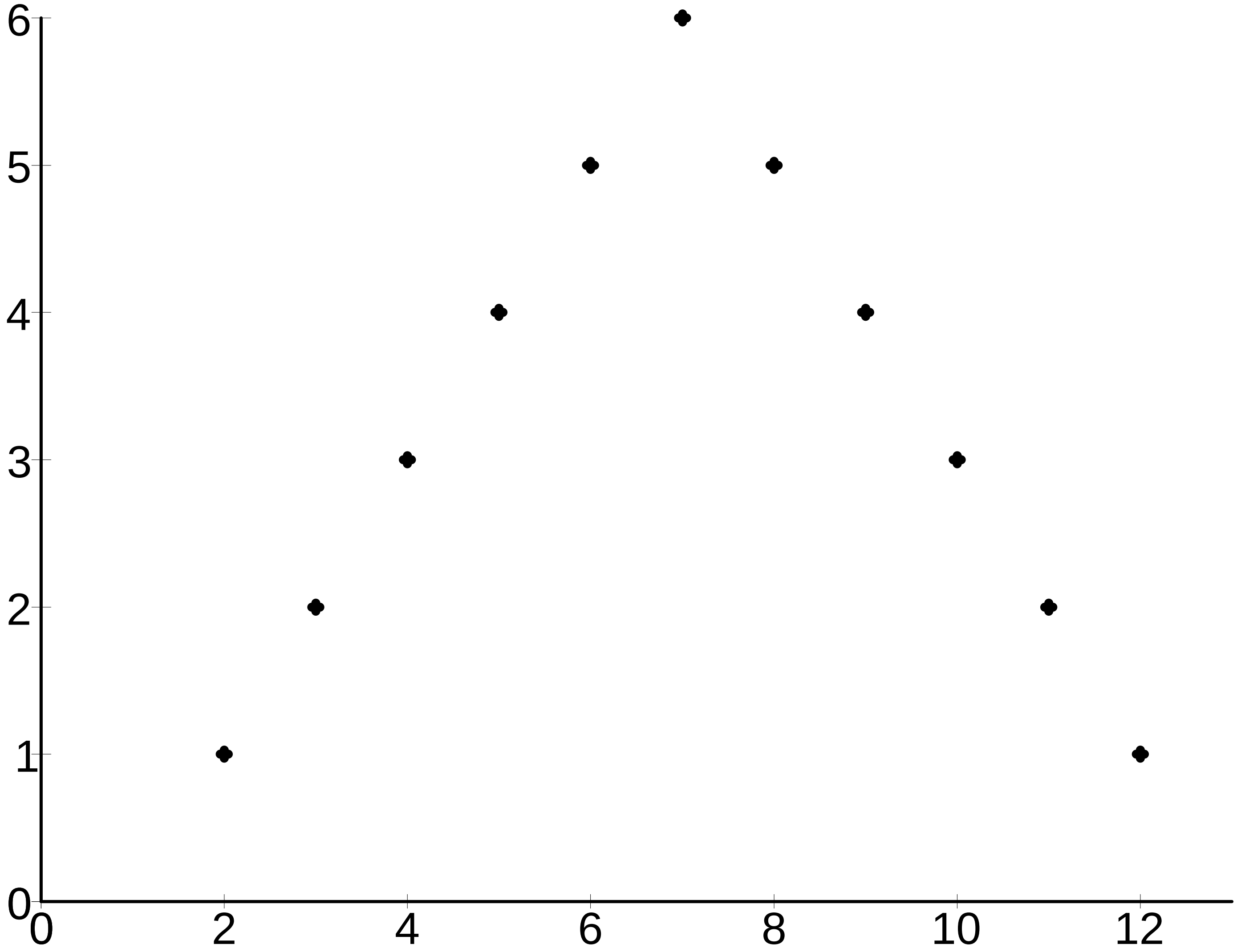
```
%       var parameters
%
% Note: file is a global variable.
%
var file:   int;                        % file number

type matrix:
        array 0..3, 0..3 of real

var xmatrix, ymatrix, zmatrix: matrix

%
% A procedure to get a 4x4 matrix from standard input.
%
procedure GetMatrix(var mat: matrix)
        for i: 0..3
                for j: 0..3
                        get :file, mat(i,j)
                end for
        end for
end GetMatrix

%
% A function to get a 4x4 matrix from standard input.
%
function GetMatrixF: matrix
        var mat: matrix
        for i: 0..3
                for j: 0..3
                        get :file, mat(i,j)
                end for
        end for
        result mat
end GetMatrix1


var matrix1, matrix2: matrix

GetMatrix(matrix1)
matrix2 := GetMatrixF
```

```
% Read and echo names; count them; check for uniqueness

const N: int := 50        % maximum names
var i, count, dups : int := 0
var found : boolean
var names : array 1 .. N of string

put "Please type a series of names."
put "Tell me ’quit’ at the end"

loop
        put "Next name? " ..
        var person : string
        get person
        exit when person = "quit" or i > N

%       OK, so a name has been entered.  Let’s see if it’s
%       already in the list.  This is called a "linear" search,
%       because we’re looking at each item in sequence.

        found := false
        i := 0;
        loop  % is person on the list already?
                exit when found or i = count
                i := i + 1
                found :=  (person = names(i))
        end loop

        if found then
                put "You know, there’s already a ",
                        person, " on this list,"
                put "so I won’t say hello to you again."
                dups := dups + 1
        else
                count := count + 1
                names(count) := person
                put "Hi, ", person, ".  I’ve added you to ",
                    "my list at position ", count
        end if
end loop
put "I greeted ", count, " people,"
put "and there were ", dups, " duplicate names entered."
```

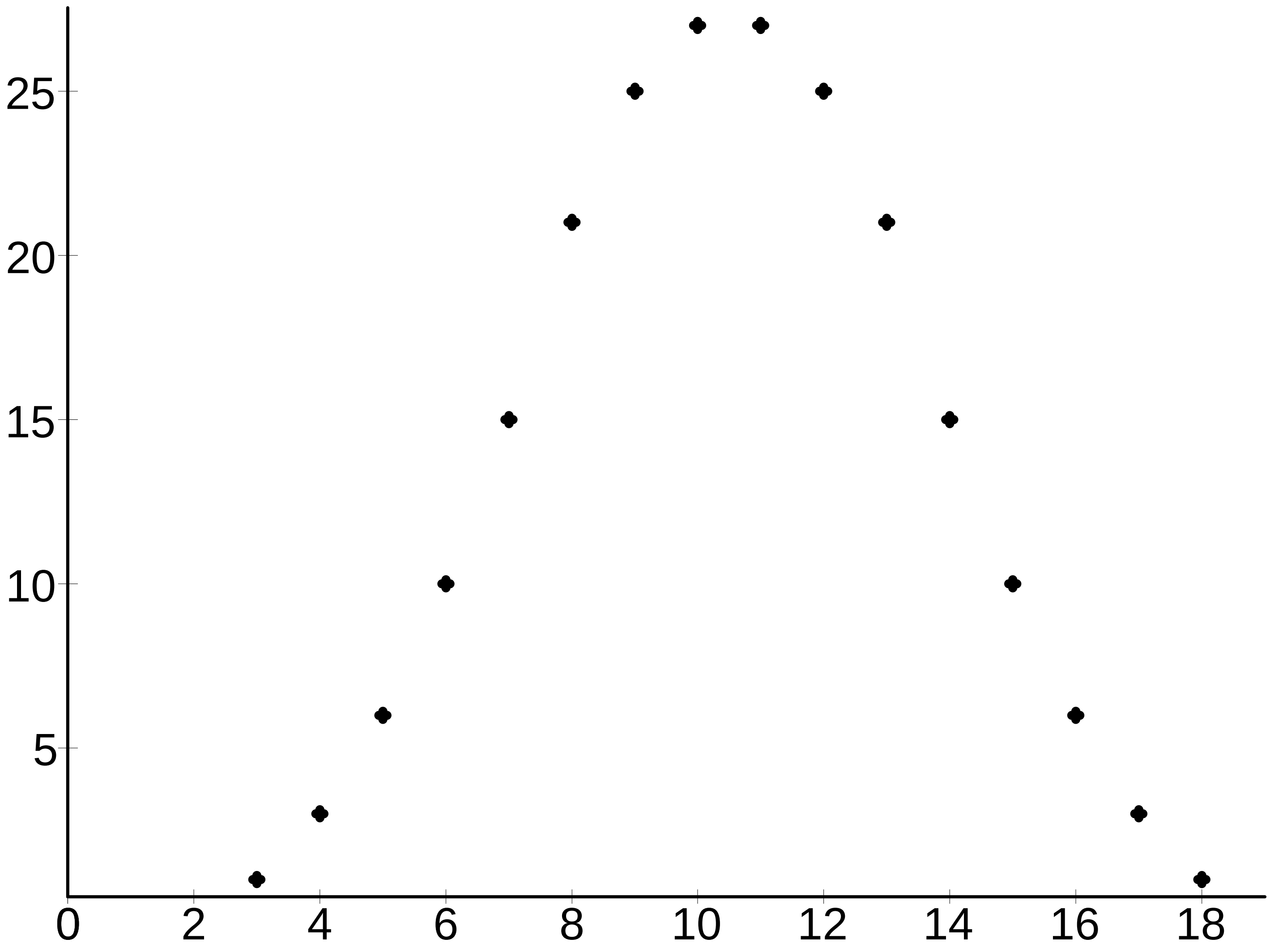
```
% Read and echo names; count them; check for uniqueness
const N: int := 50        % maximum names

type namelist:
        array 1 .. N of string


%
% We can define a linear search procedure separately like so.
%
function found(list: namelist, name: string, n: int): boolean
        var done: boolean := false
        var i: int
        i := 0;
        loop  % is name on the list already?
                exit when done or i = n
                i := i + 1
                done :=  (name = list(i))
        end loop
        result done
end found
```

```
%
% Now we can do things a little more cleanly in the main code.
%

var i, count, dups : int := 0
var names : namelist

put "Please type a series of names."
put "Tell me 'quit' at the end"

loop
        put "Next name? " ..
        var person : string
        get person
        exit when person = "quit" or i > N

%       OK, so a name has been entered.  Is it already on the list?

        if found(names, person, count) then
                put "You know, there's already a ",
                        person, " on this list,"
                put "so I won't say hello to you again."
                dups := dups + 1
        else
                count := count + 1
                names(count) := person
                put "Hi, ", person, ".  I've added you to ",
                    "my list at position ", count
        end if
end loop
put "I greeted ", count, " people,"
put "and there were ", dups, " duplicate names entered."
```

Frequency distribution of the sum of two rolled dice



Frequency distribution of the sum of three rolled dice



```
%
%	These are data types and constants that are global to
%	all procedures in our search program.
%
const N := 1000

type namelist:
	array 1..N of string
```

```
%
%     Get a file and put it into an array specified by the user.
%     Also return number of elements read.
%

procedure GetFile( var A: array 1..* of string, var num: int)
      var stream: int
      var filename: string
      var i: int := 0

      put "Enter a file name containing the list:  "..
      get filename

%     The file name should exist, etc.  These checks are omitted, but
%     you shouldn't.

      put "OK.  Taking data from ", filename
      open :stream, filename, get
      assert stream > 0

      loop
            get: stream, skip
            exit when eof(stream) or i >= N
            i := i+1
            get :stream, A(i):*
      end loop

      close :stream
      num := i

end GetFile
```

```
%
%     Search for an element in the list and return its index.
%

function search(list: array 1..* of string, key: string, N: int): int

%
%     Linear search.  Find element i in range 1..N such that
%     list(i) = key, if key exists in list.
%
%     If so, return i, otherwise return 0.

   var i := N

   loop
       exit when i <= 0 or list(i) = key
       i := i-1
   end loop

   result i
end search
```

```
%
%     Search for an element in the list and return its index.
%

function search(list: array 1..* of string, key: string, N: int): int

%
%     Binary search.  Find element i in range 1..N such that
%     list(i) = key, if key exists in list.
%
%     If so, return i, otherwise return 0.

   var first := 1
   var last  := N

   loop
       const middle := (first + last) div 2
       if list(middle) > key then
               last := middle - 1
       elsif list(middle) < key then
               first := middle + 1
       else
               result middle
       end if
       exit when first > last
   end loop

   result 0
end search
```

```
%
%     Search for an element in the list and return its index.
%

function search(list: array 1..* of string, key: string, N: int): int

%
%     A little more sophsticated search.
%     Binary search.  Find element i in range 1..N such that
%     list(i) = key, if key exists in list.
%
%     If so, return i, otherwise return 0.

   var first := 1
   var last  := N

   loop
       const middle := (first + last) div 2
       if list(middle) >= key then
              last := middle
       elsif list(middle) < key then
              first := middle + 1
       end if
       exit when first >= last
   end loop

   if list(first) = key then
       result first
   else
       result 0
   end if

% But what if list(middle) = key in the first place (or thereafter?)

end search
```

```
%
%     A general "driver" program for searching
%
%
include "types.t"
include "getfile.t"
include  "search.t"

var A:   namelist
var key: string
var n:   int

GetFile(A,n)

loop
        put "Item to search for? " ..
        get key:*
        exit when key = "quit"
        put search(A,key,n)
end loop
```

```
turingx driversearch1.t
Enter a file name containing the list:  data
OK.  Taking data from data
Item to search for? John
5
Item to search for? Joan
4
Item to search for? Jim
3
Item to search for? Eugene
0
Item to search for? Fred
0
Item to search for? Janet
2
Item to search for? Janett
0
Item to search for? quit
```

```
%
%       A general "driver" program for searching that does
%       no prompting.
%
%
include "types.t"
include "getfile.t"
include  "search.t"

var A:   namelist
var key: string
var n:   int
var foo: int

GetFile(A,n)

loop
%       put "Item to search for? " ..
        get key:*
        exit when key = "quit"
        foo := search(A,key,n)
end loop
```

I’ve created a file which contains names of all students
in CSC104/108/148. In UNIX, I can use "wc" for word count
which tells me that it contains...

wc names
```
  745    2771   23095 names
```

Next I constructed a script to exercise the driver that contains "names" followed by a set of queries. There are exactly 745*3 queries. Exactly 745 of them will fail.

Let’s time the results for the different searches.

LINEAR Search:

time turingx driversearch2.t <script.search

```
Enter a file name containing the list:
OK.  Taking data from names
16.010u 0.040s 0:16.15 99.3% 0+399k 1+0io 0pf+0w
```

BINARY Search 1 :

time turingx driversearch2.t < script.search
```
Enter a file name containing the list:
OK.  Taking data from names
0.850u 0.050s 0:00.90 100.0% 0+386k 0+0io 0pf+0w
```

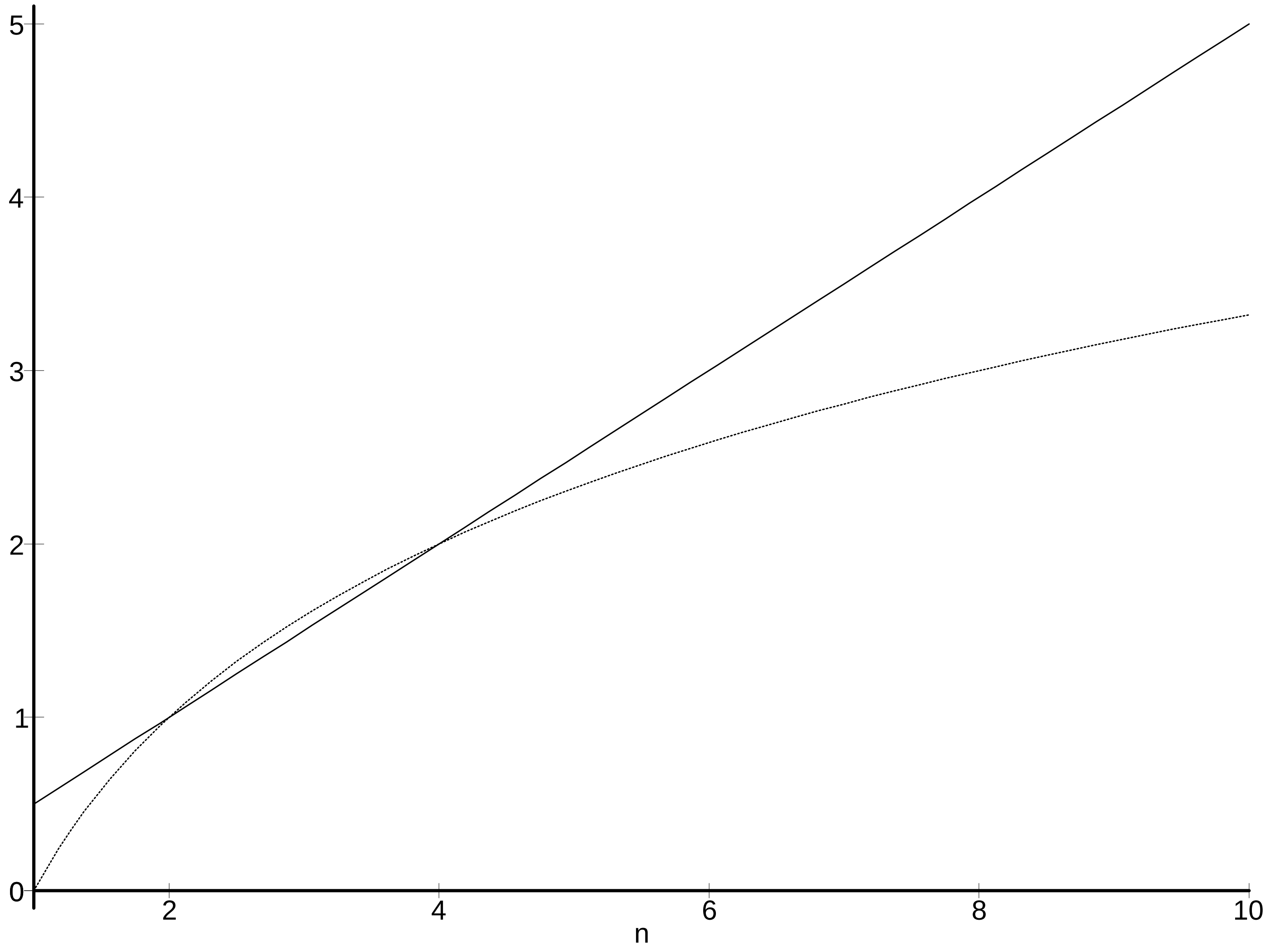
BINARY Search 2 :

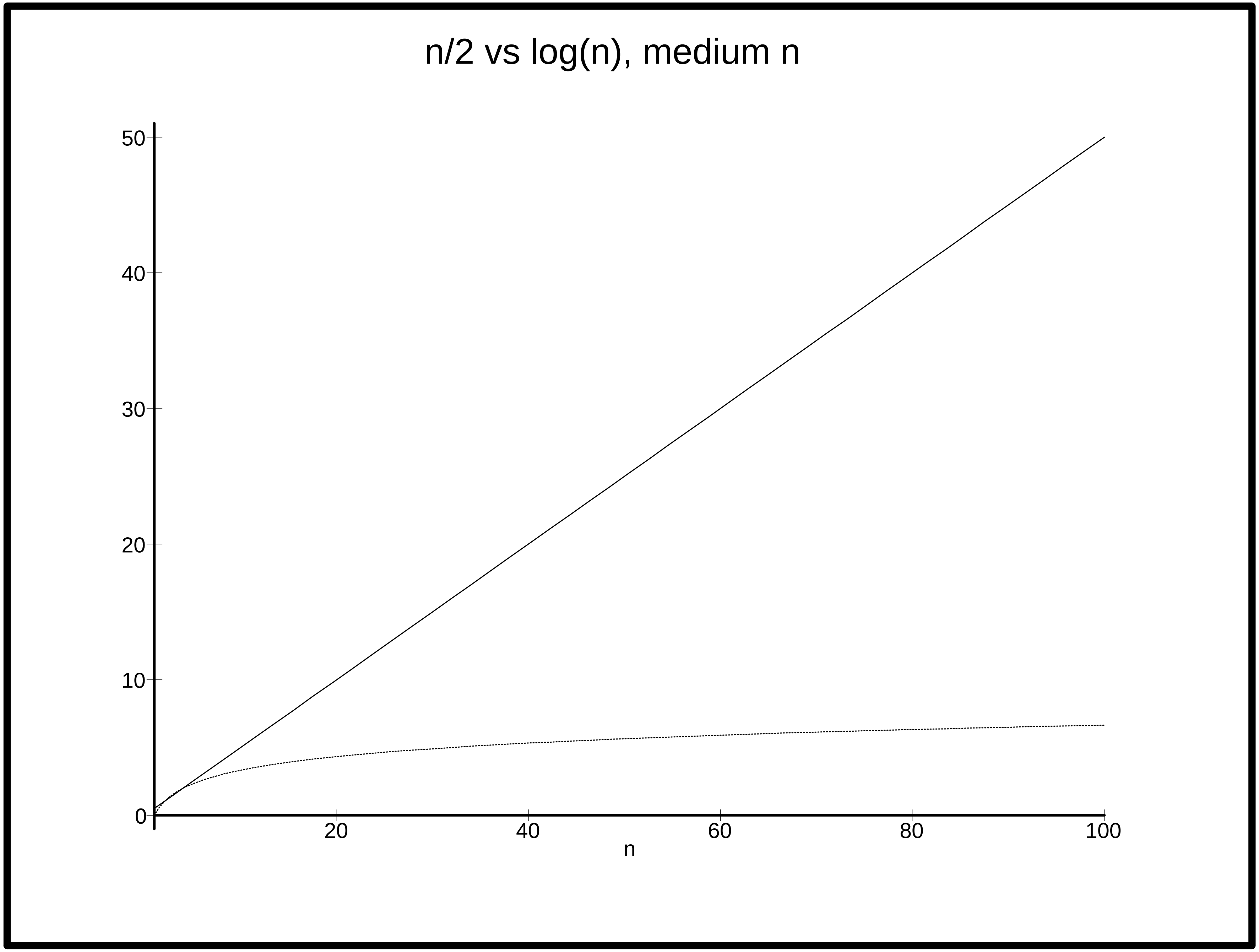
```
Enter a file name containing the list:
OK.  Taking data from names
0.850u 0.050s 0:00.90 100.0% 0+386k 0+0io 0pf+0w
```

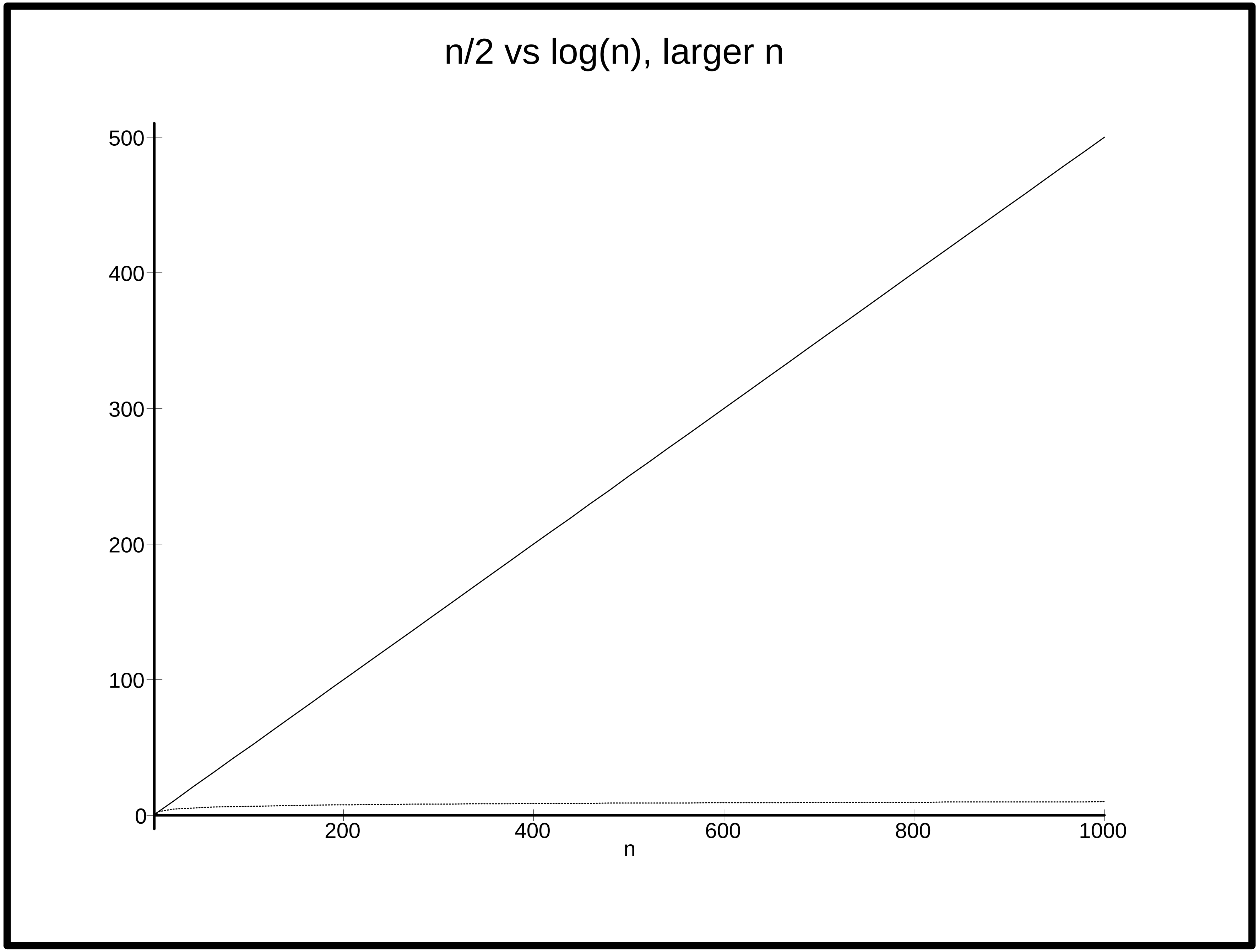
hoc
16.010/0.850
```
    18.835294
```

So, binary search is about 19 times faster in this case.
But how do we get a list in sorted order? ... slowly

n/2 vs log(n), small n




n/2 vs log(n), medium n
50
40
30
20
10
0
20
40
60
80
100
n


n/2 vs log(n), larger n
500
400
300
200
100
0
200
400
600
800
1000
n

%

```
%     Insertion Sort
%
procedure sort (var A: array 1..* of string, N:int)

   var inserted: boolean
   var next: string

   for s : 2..N                      % A(1..s-1) are sorted
       next := A(s)
       inserted := false
       for decreasing pos: s..2       % find place to insert
          if A(pos-1) <= next then
              A(pos)   := next
              inserted := true
              exit
          else
              A(pos) := A(pos-1)      % shift element one down
          end if
       end for

       if not inserted then
          A(1) := next
       end if
   end for
end sort
```

```
%
%    Bubble Sort
%
procedure sort (var A: array 1..* of string, N:int)

var sorted: boolean

   for decreasing last: N..2
       sorted := true                    % any interchanges this time?
       for i: 1..last-1
           if A(i) > A(i+1) then  % interchange needed?
               sorted := false
               const foo := A(i)  % yes, do interchange
               A(i)   := A(i+1)
               A(i+1) := foo
           end if
       end for
       exit when sorted
   end for
end sort
```

```
%
%     A general "driver" program for sorting.
%
%
include "types.t"
include "getfile.t"
include  "sort.t"

var A:   namelist
var n:   int

GetFile(A,n)

put "Unsorted list:"
for i:1..n
        put A(i)
end for

sort(A,n)

put " "
put "Sorted list:"
for i:1..n
        put A(i)
end for
```

turingx driversort1.t

```
Enter a file name containing the list:  data1
OK.  Taking data from data1

Unsorted list:
Spike
Joan
Ed
Lance
Sammy
Elizabeth
Pinky

Sorted list:
Ed
Elizabeth
Joan
Lance
Pinky
Sammy
Spike
```

```
%
%     A general "driver" program for sorting without prompting.
%
%
include "types.t"
include "getfile.t"
include  "sort.t"

var A:   namelist
var n:   int

GetFile(A,n)

% put "Unsorted list:"
% for i:1..n
        % put A(i)
% end for
%
sort(A,n)

put " "
put "Sorted list:"
for i:1..n
        put A(i)
end for
```

Bubble sort on a sorted list

0.2 real 0.1 user 0.0 sys

Insertion sort on a sorted list

0.3 real 0.2 user 0.0 sys

Bubble sort on a reversed list

22.0 real 19.4 user 0.0 sys

Insertion sort on a reversed list

8.9 real 8.5 user 0.1 sys

Bubble sort on a nearly sorted list

0.3 real 0.1 user 0.0 sys

Insertion sort on a nearly sorted list

0.3 real 0.1 user 0.0 sys

```
%
%     General Insertion Sort:
%             the comparison operation is a parameter.
%
procedure generalsort (var A: array 1..* of string, N:int,
              function greaterthan(a,b:string): boolean)

   var inserted: boolean
   var next: string

   for s : 2..N                    % A(1..s-1) are sorted
       next := A(s)
       inserted := false
       for decreasing pos: s..2       % find place to insert
          if greaterthan(next,A(pos-1)) then
              A(pos)   := next
              inserted := true
              exit
          else
              A(pos) := A(pos-1)      % shift element one down
          end if
       end for

       if not inserted then
          A(1) := next
       end if
   end for
end generalsort
```

```
%
%     General Bubble Sort:
%             the comparison operation is a parameter.
%
procedure generalsort (var A: array 1..* of string, N:int,
              function greaterthan(a,b:string): boolean)
   var sorted: boolean

   for decreasing last: N..2
       sorted := true                    % any interchanges this time?
       for i: 1..last-1
          if greaterthan(A(i),A(i+1)) then  % interchange needed?
              sorted := false
              const foo := A(i)  % yes, do interchange
              A(i)   := A(i+1)
              A(i+1) := foo
          end if
       end for
       exit when sorted
   end for
end generalsort
```

```
%     A general "driver" program for sorting.

include "types.t"
include "getfile.t"
include  "sortgen.t"

function less(a,b:string): boolean
        result a < b
end less

function greater(a,b:string): boolean
        result a > b
end greater

var A,B:   namelist
var n:   int

GetFile(A,n)
B := A                              % save A in B (N/A in all languages)

put "Unsorted list:"
for i:1..n
        put A(i)
end for

generalsort(A,n,less)
put " "
put "Sorted list in descending order:"
for i:1..n
        put A(i)
end for

generalsort(B,n,greater)
put " "
put "Sorted list in ascending order:"
for i:1..n
        put B(i)
end for
```

turingx driversort3.t

```
Enter a file name containing the list:  data1
OK.  Taking data from list

Unsorted list:
Spike
Joan
Ed
Zeke
Lance
Sammy
Carlotta
Elizabeth
Pinky

Sorted list in descending order:
Zeke
Spike
Sammy
Pinky
Lance
Joan
Elizabeth
Ed
Carlotta

Sorted list in ascending order:
Carlotta
Ed
Elizabeth
Joan
Lance
Pinky
Sammy
Spike
Zeke
```

```
%	A general "driver" program for sorting.

include "types.t"
include "getfile.t"
include  "sortgen.t"

var compares:int := 0

function less(a,b:string): boolean
	compares := compares + 1
	result a < b
end less

function greater(a,b:string): boolean
	compares := compares + 1
	result a > b
end greater

var A,B:   namelist
var n:   int

GetFile(A,n)
B := A				% save A in B (N/A in all languages)

put "Unsorted list:"
for i:1..n
	put A(i)
end for

compares := 0
generalsort(A,n,less)
put " "
put "Sorted list in descending order:"
for i:1..n
	put A(i)
end for

put "Number of comparisons: ", compares
compares := 0

generalsort(B,n,greater)
```

```
put " "
put "Sorted list in ascending order:"
for i:1..n
        put B(i)
end for

put "Number of comparisons: ", compares
```

Number of comparisons taken to sort a list

Bubble sort on a sorted list: 744

Insertion sort on a sorted list: 744

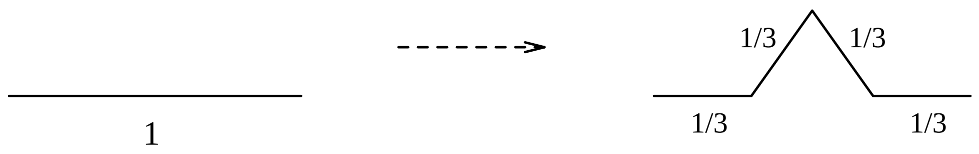
Bubble sort on a reversed list: 277140

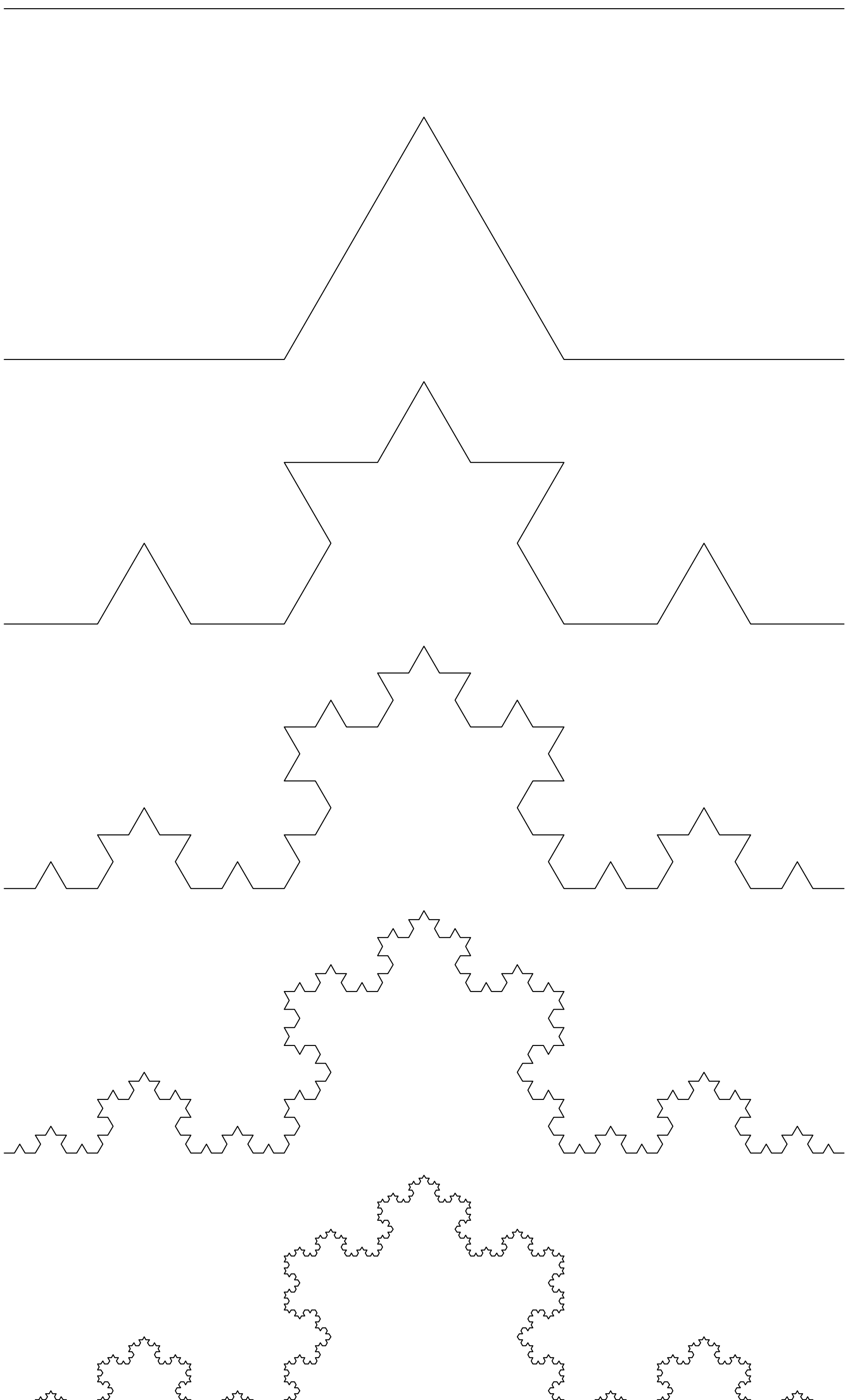
Insertion sort on a reversed list: 277140

Bubble sort on a nearly sorted list: 798

Insertion sort on a nearly sorted list: 2229

Replacement Rule for von Koch curve





```
%
% fractaltypes.t
%
% Elementary types and constants for deterministic fractal program
%

const N := 2048                         % max line segments in fractal
const Pi := 3.1415927
const scale := 512                      % scale factor for image
const Xres := 800                       % X and Y resolutions
const Yres := 800
const xoffset := 100                    % display offset in X, Y
const yoffset := -Yres + 100

type realpoint:
        record
                x: real
                y: real
        end record

type intpoint:
        record
                x: int
                y: int
        end record


type fractal:
        array 1..N of realpoint
```

```
%
%     vondriver.t
%
%     Driver to create von Koch fractals
%
include "fractaltypes.t"
include "vonkoch.t"
include "init.t"

var newfractal, oldfractal: fractal
var ans: string
var n, nnew: int

setscreen("nocursor")
setscreen("graphics:800;800")

line(oldfractal,n)              % init oldfractal to a line
triangle(oldfractal,n)          % init oldfractal to a triangle
square(oldfractal,n)            % init oldfractal to a square

% PutFractal(oldfractal,n)

put "Plotting von Koch curve"

loop
      put "Hit return to continue or 'quit' to stop " ..
      get ans: *
      exit when ans = "quit"
      Replace(oldfractal, n, newfractal, nnew, RepVonKoch)
      DrawFractal(newfractal, nnew)
      oldfractal := newfractal
      n := nnew
end loop
```

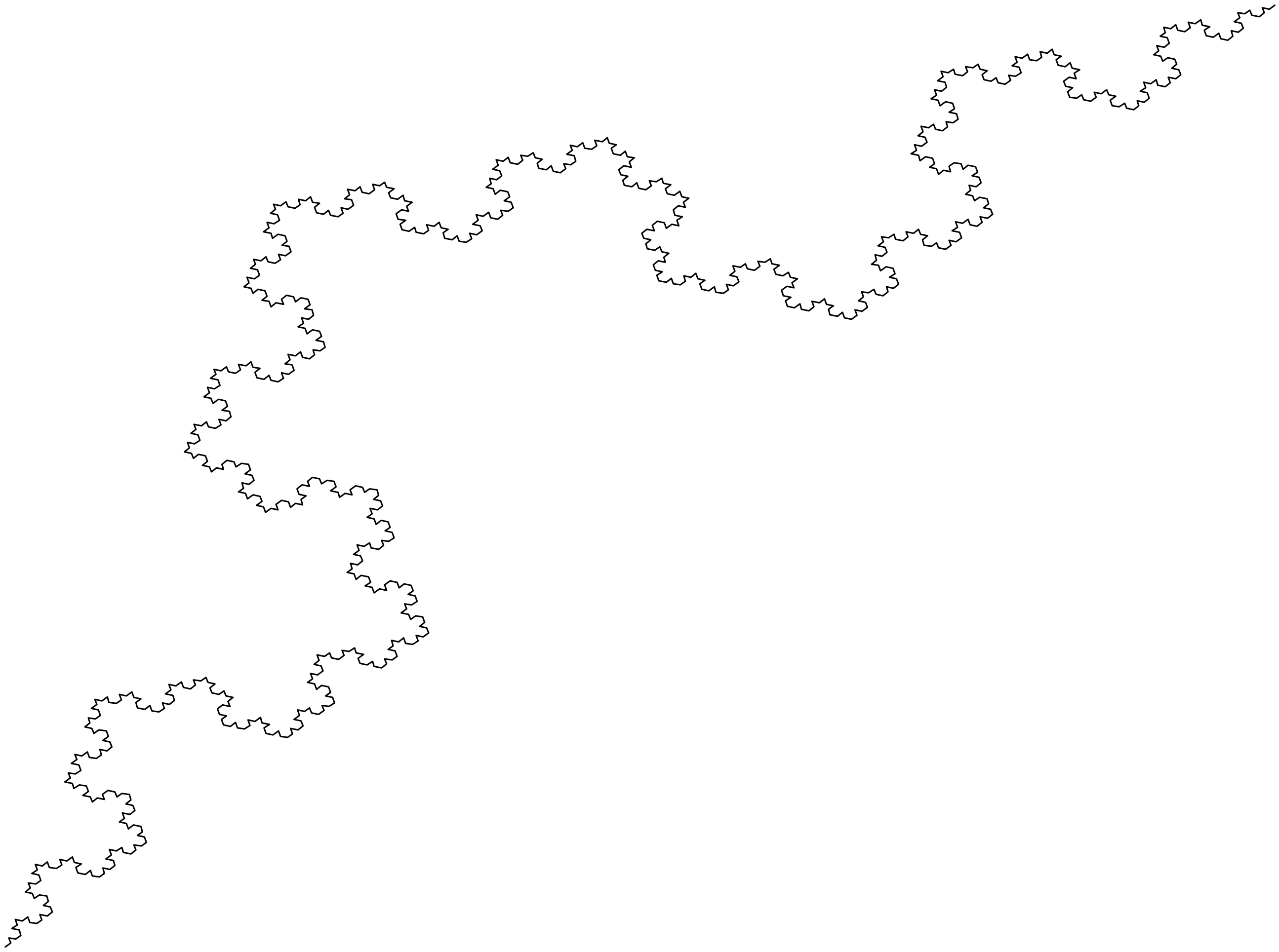
```
%
%     init.t:
%
%     Routines that create some initial shapes to transform using
%     geometric replacement rules.
%
procedure line(var pts: fractal, var n: int)
        pts(1).x := 0
        pts(1).y := 0
        pts(2).x := 1.0
        pts(2).y := 0
        n := 2
end line


procedure triangle(var pts: fractal, var n: int)
        pts(1).x := 0
        pts(1).y := 0
        pts(3).x := 1.0
        pts(3).y := 0
        pts(2) := Rotate(pts(1), pts(3), Pi/3)
        n := 3
end triangle

procedure square(var pts: fractal, var n: int)
        pts(1).x := 0
        pts(1).y := 0
        pts(2).x := 0
        pts(2).y := -1
        pts(3).x := 1
        pts(3).y := -1
        pts(4).x := 1
        pts(4).y := 0
        pts(5).x := 0
        pts(5).y := 0
        n := 5
end square
```

```
%
%	vonkoch.t
%
%	routines that implement geometric replacement schemes
%
%	Return point that is the fraction t between P and Q.
%	"Lirp" stands for linear interpolation.

function Lirp(P,Q: realpoint, t: real): realpoint
	var newpt: realpoint

	newpt.x := P.x + t*(Q.x-P.x)
	newpt.y := P.y + t*(Q.y-P.y)
	result(newpt)
end Lirp


function ConvertToScreen(p: realpoint): intpoint
	var q: intpoint
	q.x := round(p.x*scale) + xoffset
	q.y := Yres - round(p.y*scale) + yoffset
	result(q)
end ConvertToScreen


%	Rotate P about Q through angle ang.

function Rotate(P, Q: realpoint, ang: real): realpoint
	var s,c,x,y: real
	var pt: realpoint

	s := sin(ang)
	c := cos(ang)
	x := P.x-Q.x;
	y := P.y-Q.y;
	pt.x :=  c*x - s*y + Q.x
	pt.y :=  s*x + c*y + Q.y
	result(pt)
end Rotate
```

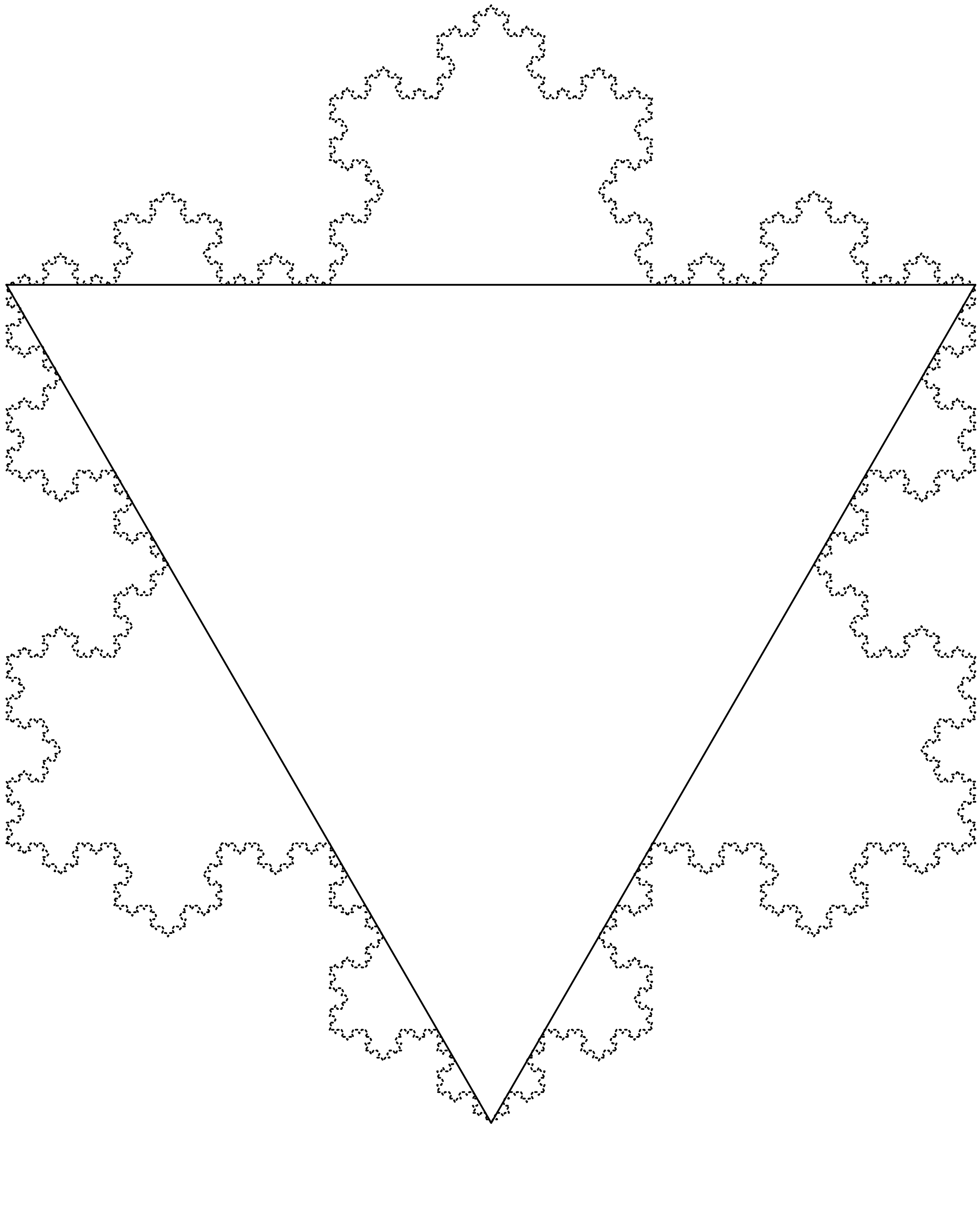
```
procedure DrawFractal(pts: fractal, n: int)
	var p0,p1: intpoint

	cls
	for i: 1..n-1
		p0 := ConvertToScreen(pts(i))
		p1 := ConvertToScreen(pts(i+1))
		drawline(p0.x, p0.y, p1.x, p1.y, 1)
	end for
end DrawFractal


procedure PutFractal(pts: fractal, n: int)
	var p0,p1: intpoint

	for i: 1..n
		p0 := ConvertToScreen(pts(i))
		put "(", pts(i).x, ",", pts(i).y, "), [", p0.x, ",", p0.y, "]"
	end for
end PutFractal

%
%	Replace the sequence of line segments in current, putting the
%	result in new using the replacement rule called rule.
%
procedure Replace(current: fractal, n: int,
		var next: fractal, var nnext: int,
		function rule(p0,p1: realpoint, var n: int): fractal)

	var newfractal: fractal
	var nfrac: int
	nnext := 0
	for i: 1..n-1
		newfractal := rule(current(i), current(i+1), nfrac)
		% Append newfractal to next
		for j: 1..nfrac
			nnext := nnext + 1
			next(nnext) := newfractal(j)
		end for
	end for
end Replace
```

```
%	Replacement rule for von Koch curve.

function RepVonKoch(P,Q: realpoint, var n: int): fractal
	var P1, Q1, C: realpoint
	var newfractal: fractal

	newfractal(1) := P
	newfractal(2) := Lirp(P,Q,1/3)
	newfractal(4) := Lirp(P,Q,2/3)
	newfractal(3) := Rotate(newfractal(2),newfractal(4),Pi/3)
	newfractal(5) := Q
	n := 5
	result(newfractal)
end RepVonKoch
```

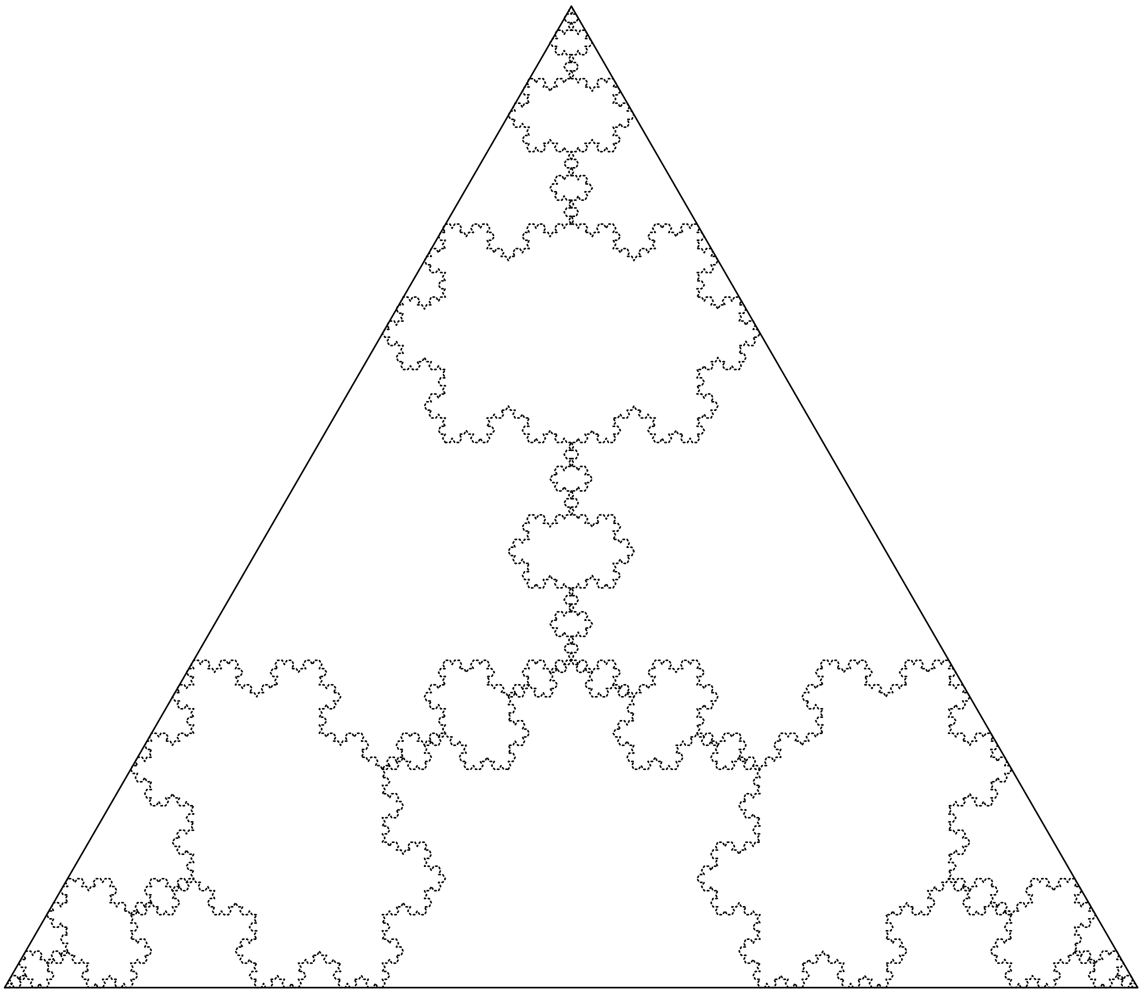
An oblique initial line segment



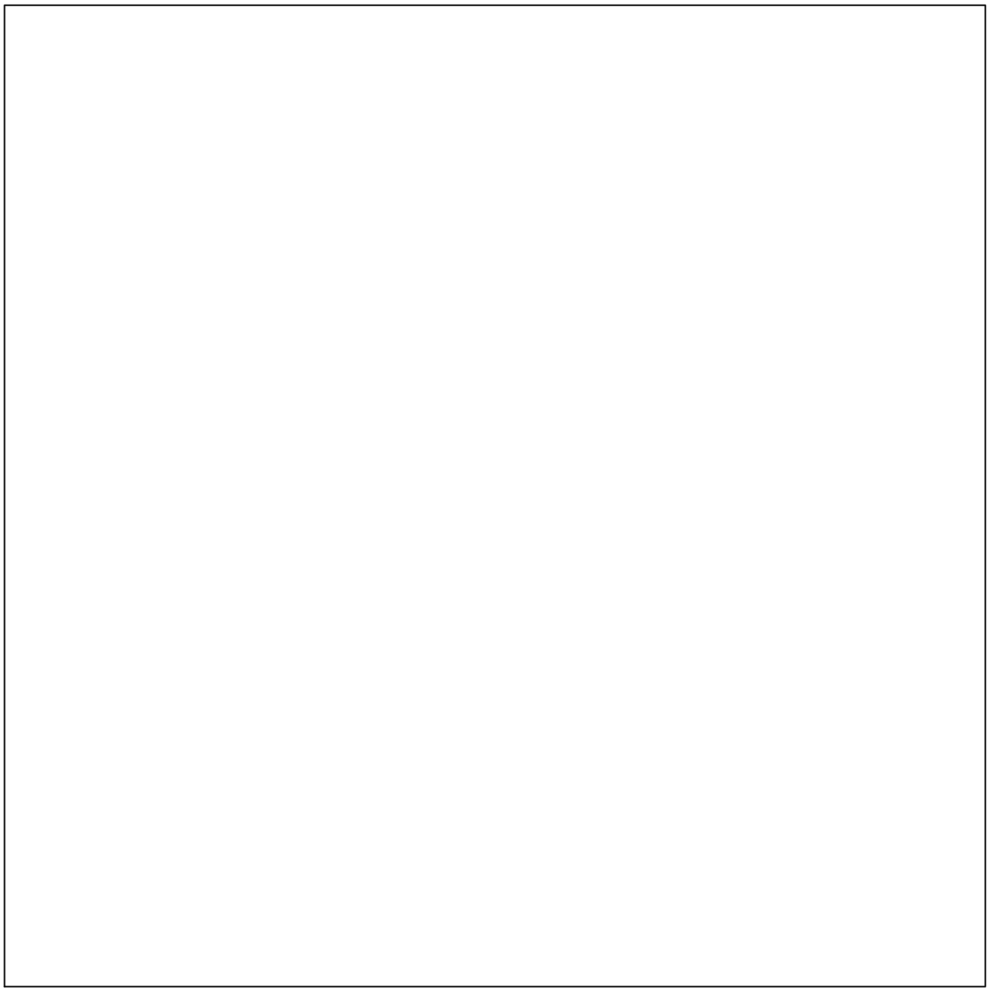
The von Koch Snowflake

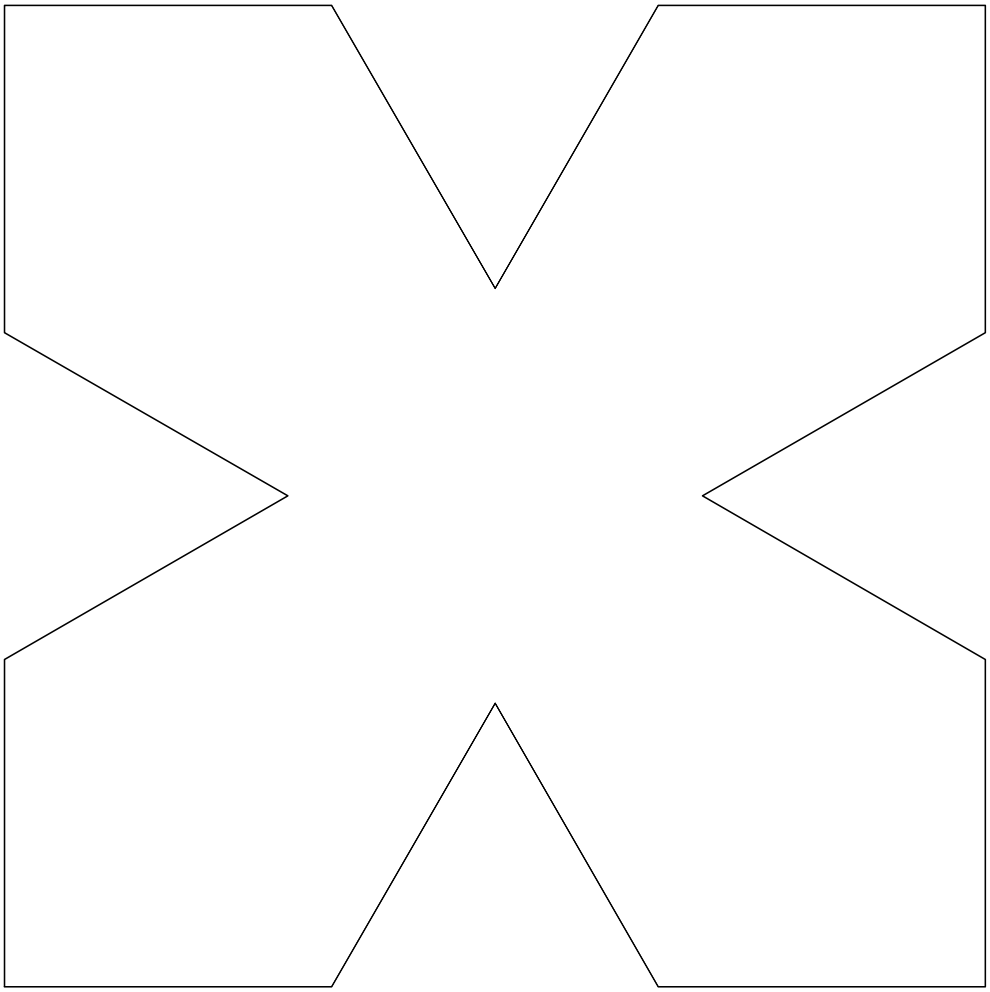


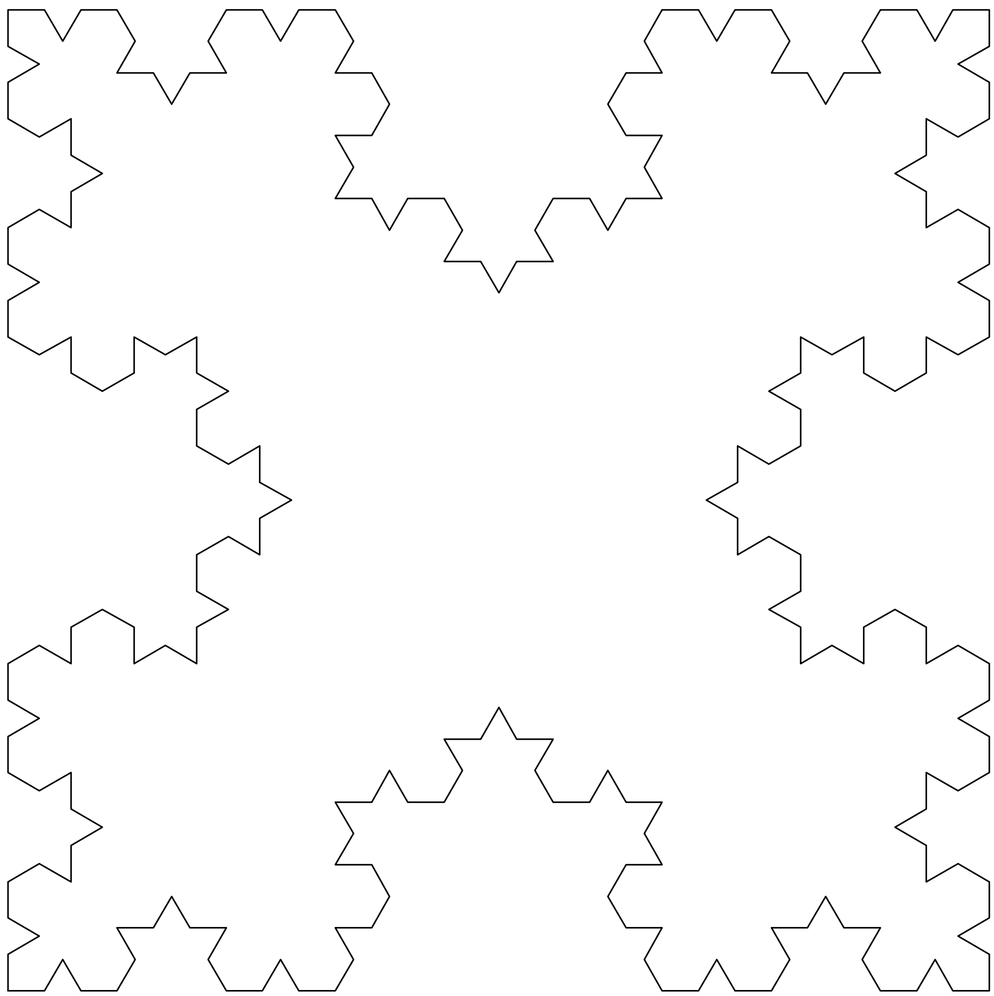
Change the rule to rotate by -Pi/3

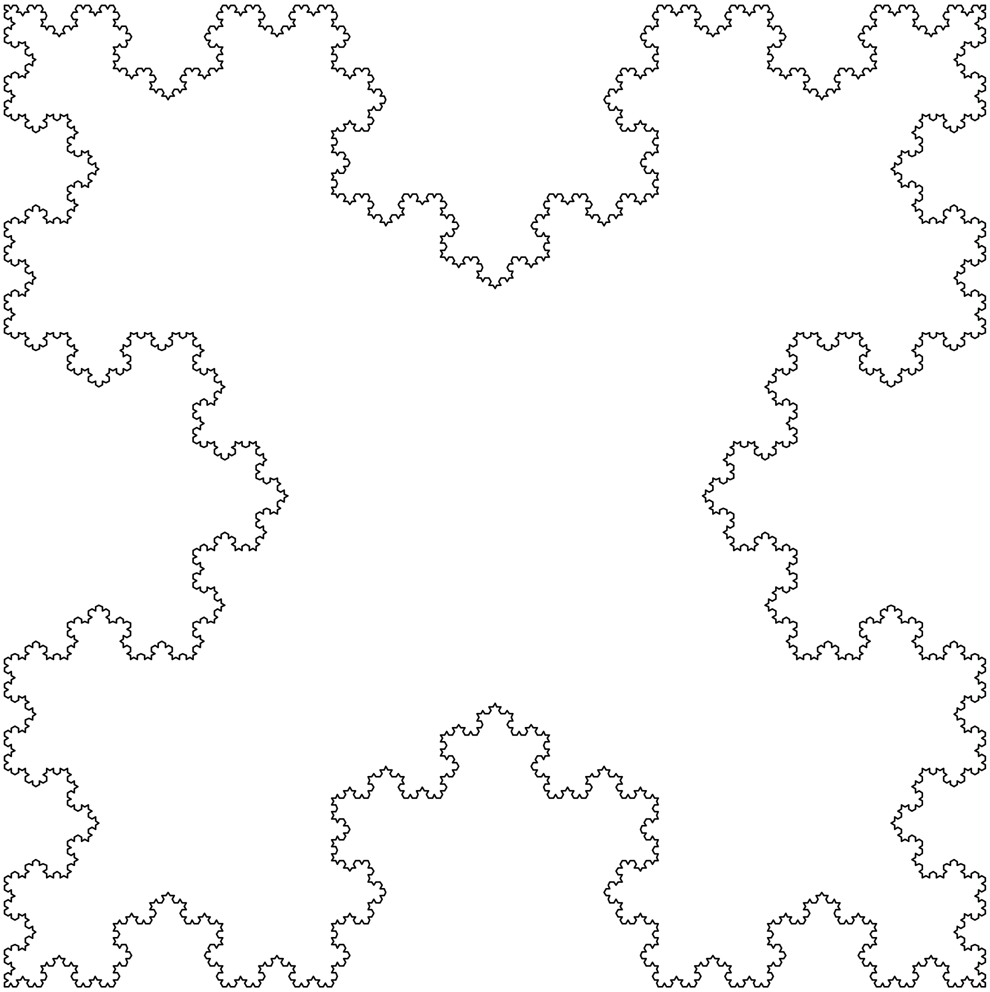


How about a square?









```
%
% fractal1d.t
%
% Recursive algorithm to generate "smoothed" 1-D fractional Brownian motion
%
const maxsize := 1024
const norands := 5                        % no of random variables

type output: array 0..maxsize of real
var Fh: output                            % global var for efficiency


function gauss(s: real) : real
        var r : real
        var sum : real := 0

        for i: 1..norands
                rand(r)
                sum := sum + (r-0.5)*2
        end for
        r := s*sum/norands
        result r
end gauss


proc subdivide( f1,f2: int, std, ratio: real)
        var fmid:  int
        var stdmid: real

        fmid := (f1 + f2) div 2
        if fmid not= f1 and fmid not= f2 then
                Fh(fmid) := (Fh(f1)+Fh(f2))/2.0 + gauss(std);
                stdmid := std*ratio
                subdivide(f1,fmid,stdmid,ratio)
                subdivide(fmid,f2,stdmid,ratio)
        end if
end subdivide



procedure fractal(maxlevel: int, h, scale: real)
```

```
  var first, last: int
        var ratio, std:  real

        first := 0
        last  := 2**maxlevel
        Fh(first) := gauss(scale)
        Fh(last)  := gauss(scale)
        ratio     := 2**(-h)
        std       := scale*ratio
        subdivide(first,last, std,ratio)
end fractal



var scale, h: real

const level := 10
setscreen("graphics:1024;800")
put "Fractal demonstration."
put "Enter smoothness value [0,1]: " ..
get h
setscreen("nocursor")
fractal(level,h,256)
randomize
for i : 0..2**level-2
        drawline(i*1,round(400+Fh(i)), (i+1)*1,round(400+Fh(i+1)),1);
end for
```

get skip is important

Consider the following code.

```
var s: string(20)
var t: string(10)

get s:20, t:10
put s:20, " ", t:10

get s:20, t:10
put s:20, " ", t:10
```

Suppose our data is:

```
01234567890123456789xxxxxxxxxx
0123456789 123456789 yyyyyyyyy
```

Then our output will be

```
01234567890123456789 xxxxxxxxxx

                    0123456789
```

However, if our code uses a get skip:

```
var s: string(20)
var t: string(10)

get s:20, t:10
put s:20, " ", t:10

get skip
get s:20, t:10
put s:20, " ", t:10
```

Our output is

```
01234567890123456789 xxxxxxxxxx
0123456789 123456789  yyyyyyyyy
```

But there’s another possibility

I’ve dressed it up a little using records, but the idea is simply to read the whole line into a variable, and then pick off the parts you want:

```
type datarecord:
   record
      s: string
      t: string                % keep lengths if desired
   end record

function GetRecord: datarecord
      var line: string
      var data: datarecord
      get line:*
      data.s := line(1..20)
      data.t := line(21..30)
      result(data)
end GetRecord

var rec: datarecord

rec := GetRecord
put rec.s:20, " ", rec.t:10

rec := GetRecord
put rec.s:20, " ", rec.t:10
```

This gives the correct results.