Fractal Cryptology

New Mexico High School

Supercomputing Challenge
Final Report
April 2, 2003

Team #053
Moriarty High School

Team Members:
Brandi Howell
Anna Reese
Michael Basile

Team Sponsor:
Paula Avery

Project Mentor:
Garth Reese

Table of Contents

Executive Summery

Secret messages need something special to keep them secret. Fractals are relatively new to the world of math. Some fractals are of nature and others look completely chaotic. Our team decided to combine these two elements, secret messages and chaotic fractals, to create a unique way of encryption.

We chose to use a two-dimensional array filled with a set of random numbers. The height represented the length of an inputted message. The width allowed for another random element, for the specific column was decided by another random number generator. Rint, the random number filling the chosen slot, was then sent to a mapper. This mapper used rint in two ways. First, we modulus rint by 5 to determine the number of iterations run for each summation. After about 5 iterations, the fractal tends towards infinity, which was unusable in the code. In appendix c, there are five separate lines because of this. Second, we modulus rint by 1000 because numbers over 1000 tended to become flat. This is especially noticeable in the bottom curve on appendix c. This second form of rint was used to determine how many different c values were used in determining z=z^2+c, the Mandelbrot fractal equation. The final value of each iteration was summed and returned to be used to manipulate the inputted message. We also modulus this number, this time by 50, to ensure it was small enough to remain in the set of ASCII values between 0-255. Then the original character of the message was added to the value returned from the mapper. Each letter was done likewise until the whole message was encrypted.

We were successful in making a program that used fractals to encrypt a message. However, we found, as we learned, that the way we used the fractal did not utilize the fractals chaotic nature to it's full potential. In addition, we found that the mapper returned some of the same values. We were not able to create a decoder and we do not believe anyone, even the recipient of the message, would have much luck decoding the message because of the repeats with the mapper. This code obviously has some troubles because if the recipient cannot decode the message, the encryption is worthless. Despite this, we have learned a lot about encryption, fractals, and mostly, about programming on our compiler with c++.

We conclude that it is possible to encrypt code using fractals. There are many ways this could be done. Our version is one with need of improvement, but it has been a source of learning for our team. Sometimes the only way to learn something will not work the way you want it to is by trial and error. Although this code does not include a decoder, it does encode. We feel success at using the fractals, although not in the ways we had previously imagined. Fractals and encryption will doubtless be combined again.

Introduction

Problem Definition

The purpose of this project is to encrypt a message in a unique way using fractals. In the program we incorporated a two dimensional array, which we filled with random numbers through the use of the random number generator. We used the random numbers in the array to change the ASCII value for each letter of the message. We also used fractals in our formula to encrypt the message. The random number was mapped using the fractal algorithm that gave us the new number for encryption.

Background

Cryptology is the art of encoding and decoding messages and has been important throughout history. In ancient days people used concealment ciphers; the message was there, but somehow hidden. For example, one king shaved his servant's head, tattooed the royal message on the shaved head, and then waited for the servant's hair to grow back. Next, the king sent the servant off to deliver the message. The receiver had to shave the servant's head once again to read the message. Obviously, this is not a very efficient way of encrypting messages, for many reasons. Another way of encrypting, in the early days, was transposition ciphers. The message was either written in columns and read diagonally or it was written in paragraph form and only every third letter read. This was a good way of encrypting messages because if someone you didn't want to receive the letter read it, they wouldn't know what you were talking about. Another way to encrypt messages is by substitution. In substitution ciphers, numbers and symbols replace the letters. Instead of a regular alphabet, you use a cryptobet. For example, the letter 'a' would really be the 'z', or a patter such as "plus three" would be used, which would make

‘a’ become ‘d.’ Today, there are many encryption methods. With super-computers, the complexity of the method is increased.

Fractals tend to be very complex, chaotic, and self similar, which can make a simple encryption method very complex. A fractal is a set of results from an equation, which is usually graphed. Although based on simple equations like the Mandelbrot z=z^2 +c, the results tend to be chaotic. For example, in z=z^+c, when z begins at 0 and c is also 0, no matter how many iterations you perform, the result is always 0. However, when z begins at 0 but c = –1.9, the results are chaotic (see appendix b).

Method of Solution

We began our project by studying the process of substitution encryption and created an encryption code. Our group decided to work with the substitution method to encrypt messages because it was easy to work with and could be hard to decipher. We started with a constant message to encrypt. We decided that a creative way to encrypt this message was with a randomly filled two-dimensional array. We used a random number generator to fill the array with non consecutive. We pictured our message written vertically with each row representing a letter of our message. We then created a second random number generator, which randomly picked a column for each row. The random number filling the decided position on the array was sent to a specific function designated for that spot on the array, i.e. column 4 goes to function 4. This function contained a fractal simulation that changed the value of the random number. In this fractal function, we used the Mandelbrot equation, z=z^2+c, where z=0 and c is a variable constant. C began at –2. We iterated the formula with c = -2 between 0-4 times, based on modulus 5 plus 1 of the original random number. Then, we added 4.0/((rint%1000)+1), where rint is the random number. This number is significant because we did not want c to be more

than 2. As it starts at –2, we used 4 as the numerator. Then, we had to modulus the random number by 1000 because we found that the results were fairly useless if the random number was larger than 1000. Finally, we added one to the denominator because the modulus could be near 0, and 0 couldn't divide any number. For each change in c until c became 2, we ran through modulus 5 plus 1 of the random number iterations. We added the last value of each set of iterations to a sum value. This was sent to our encryption function where the ASCII value of the original was changed, resulting in a new letter.

Results

To begin, we used a random number generator provided by our teacher. However, that program failed to give us sufficient random numbers so we added our own formulas, which created what seemed to be random numbers. At the last moment, we found that our random number generator was not in-fact producing a set of random numbers, but creating a new seed value each time. We learned of the popular rand() function to obtain all the random numbers to full our array. Once the array was filled, we learned to write our program to a file so that we could review the results. We began changing a constant message with the results from the 2-D array. We made the message change into only A-Z and a-z characters. Next we made a fractal program that found the Mandelbrot set. We found that fractals have very chaotic elements. However, in our code we used the fractal to map the random numbers (see appendix c). We then incorporated that into our program. When the program is ran you are asked for a message that you want to encrypt, the encryption process is run with no print out of the process, and then the final encrypted messages is displayed.

Conclusions

We were unable to create a decoder. In fact, this may not be able to be decoded because when we mapped the results we found that some of the numbers, especially as they grew larger, mapped to the same number. That seems like a failure, but it is not. We not only learned a lot, but we can imagine a decoder. To decode, we would need to perfect the mapper so that no number came out twice. We did accomplish our main goal: to encrypt a message in a unique way. This code has many limitations; however, it will encrypt a message. We have learned about taking projects one-step at a time. We had expectations for a conclusion that fractals ought to be the new way to encrypt. As we set out to write the code, we found that we had to learn other things before we could even add fractals. With more time, experience, and especially with an experienced guide, this code could do many things. It wrote it to be separate enough as to use parallel processing. We never got near that point. We finally did add the fractal element, but we found that it didn't do what we thought it would. This has still been a good project, though, because we learned about working hard, about learning through the Internet, books, and trial by error, and about taking each step, one at a time.

Acknowledgements

We would like to thank many people for assisting us in our project. First, we would like to thank our teacher, Paula Avery. We would also like to thank our mentor, Michael T, who help us start with a simple method of encryption. We would also like to thank Garth Reese, for helping us decide to use a two dimensional array and assisting us in cleaning our code.

Recommendations

We have learned a lot through this project. We worked daily on little steps. To finish, now, is a little disappointing. We have accomplished what we had first intended, but with more direct help, we could have done much more. For example, we spent many days making all the letters change into regular letters; that is, between A-Z and a-z. As we look back, that was not even needed. Also, we had many problems with the random number generator, but recently we found that all our troubles were mostly syntax. This is disappointing. However, there is much that could be added to this project. Fractals are very chaotic and our mapper does not utilize that quality as well as it could. Of course, decryption could be added. We think that the mapper would have to be changed to incorporate decryptions, because it gives some of the same numbers, which would be very difficult to decipher.

```
/*
  Name: Project Program
  Author: Anna & Michael
  Description: Using a 2D array w/ random #'s to encode a message.
  Date: 13/01/03 09:44
*/
#include <stdlib.h>    // for pause at end of program
#include <iomanip.h>   // so we can cout with spaces
#include <iostream.h>  // for cout and cin
#include <time.h>      // random # generater, uses time
#include <fstream.h>   // so we can write the file
#include <math.h>      // for a tangent
#include <assert.h>    // used when rarely something needs to be
conditional

int stringlength(char *string_in);// our string counter funciton
int i;                              // random number for columns
int rn;                             //random number to fill arrays
char message[2000000];// so it is as big as it can get.
const int  size_2 = 21;// size_2 is the width of our array.

int msglen=0;// going to be the length of our message.
int **xy_axis=0;// declared as 0 so we can change it to being our
"message"'s length within Main.

int xy;   // global to use in message() and array()
int verbose=0;// this allows us to choose to talk or not- used
throughout the program
const int MAXINT=2147483647;// the largest signed int.

int array();                                          //Mike's function
to fill a 2D array with random numbers
int get_num1(); // should have (int i)     // random number generater
to get n.

int print_array();  // this should print the xy axis  I inserted it at
end of array()

int Message();  // this should go through all the letters in the
message
    // it also changes the value

int function_xy_r_xx(int id);


int r = 0;            //row int

int main()
{
  for (int u=0; u!=10; u++){  // for loop to test accuracy
  srand(time(NULL)); // select random seed
  //srand(123456611);  // comment out for production

  cout << "What is the message that you want to encrypt?\n"
       <<"(must be under 2 million characters)\n";// ask user for
message
  cin.get(message, 2000000);// getting the message
  cin.ignore(2000000, '\n');
```

```
  if (verbose)
  {
      cout << message << '\n';  // just used to make sure we can print
it out.
      cout << "The length of the string is " << stringlength(message)
           << " characters long.\n";
       }
  msglen = stringlength(message);        // so we don't keep calling
the function.
  xy_axis=new int*[msglen];      // sets the xy_axis to the string
length.
  xy_axis[0]=new int[msglen*size_2];
  for (int j=1;j<msglen;j++)// for loop to make xy_axis j tall, which
is as tall as our message.
    xy_axis[j]=xy_axis[j-1]+size_2;

  array();// fills array.
 Message();// message function

  delete[] xy_axis[0];// have to double delete because we double
declared.
  delete[] xy_axis;  // needed to close it up.  Don't want to leave
anything hanging.

  cout <<endl;  // so it looks pretty before we end.  I like it better
nice, organize, and clean.
  }//end of for loop to test accuracy.
  system ("PAUSE");
  return 0;
}



/* Take each character in the message. Change it into a crypto letter.

   The crypto letter must be a printable letter.
   The crypto letter replaces the original letter in the message.

   How do we do it....
   Obtain a key for each character.
   Take the modulus of it.
   add that number to the original message letter. If that's a higher
value
   than 'z' let it circle around to 'A'.
   if the value is between 93 and 96, add 5 to make it a letter.
*/
int Message()
{
  // for my first letter/number, I do row 1(or a counter called r),
number n
  static char blank=' ';  // static- stays the same.  Like a const,
only a char.
  get_num1();  //  gets rand number for n
  for ( r=0;r<= msglen-1; r++)// r= letter(the first letter, etca loop
to use all the letters in the message
    {
      int n = i%21;// i is random number, modulus 21 because array is
only 21 wide.
```

```
       if (verbose)
      cout <<"This is the value for n " << n << '\n';      //This is
the value for the y cordinate
      cout << '\n';

//These are all the if statements to call up the functions for when the
n value (y-cordinate) is in each of the 21 places in the array
      if(n >= 0 && n < 20 )
      {
        function_xy_r_xx(n);
        if (verbose)
        cout <<"This is the value for xy function "<< n<<": " << xy <<
'\n';

      }

      xy_axis[r][n];      // defines the xy cordinate we are going to
use
       if (verbose)
    cout << "Right now, the n = "<< n <<endl;        // just tells us
what our y cordinate is


      if (message[r] == blank)                              //This if is
for a space in the message
      {
        message[r] = blank;                                  //This
keeps a space a space in the message
      }
      else                                                 //This is for
the other characters ofther then the space.
      {
        if((message[r] >= 1) && (message[r] <= 64))       //This is for
a character other than 65-122
          {
            message[r] = message[r];                          //This keep
the character what it is in the message
          }
        else
          {
            if(message[r]+(xy%50) <= 122)
//This is for changing the A-Z and a-z to a new letter
            { message[r]=message[r]+(xy%50);
//Reasigns message[r] to a new letter
            if((message[r] >= 91) && (message[r] <= 96))
//This is for if the new letter is between 91-96
              {
                message[r]=((message[r]+(xy%50)) - 90) + 97;
//Changes the letter to a letter after 122
                //  message[r]=message[r]+(xy_axis[r][n]%50);
              }
            }

            else   //This is for changing the value if it is above 122
            {
              message [r]=(message[r]+(xy%50)-122)+65;
              //Reasignes the letter to a letter between 65-122
              if((message[r] >= 91) && (message[r] <= 96))
```

```
                //This changes the letter to another letter that is not
in between 91-96
                {
                  message[r]=(message[r] - 90) + 97;
                  //Reasigns the letter to a letter not in the 91-96
range
                }
            }    //65-90...97-122
          }
      }
      cout << message << endl;                    //prints out what
the message is after each letter is changed

    }                                //closes for loop
  cout << message << endl;         //Prints out what the final message
is


  return 0;
}                                // closes function

int array()
{
   for (int x=0; x <msglen; x++)
          {
            for (int y=0; y <size_2; y++)
             {
                    rn=rand();              // gets a random number.
                    xy_axis[x][y] = rn;
                  }
          }
 if (verbose)
 print_array();
 return 0;
}


int get_num1()        //this is the rng for picking value n.
{
 i = rand()+(2 * msglen);// so i equals the changed random # based on
the length of the string, too. (I deleted (&20 right before the 2)
  return 0;
}

/* z = z*z + c= mandlerot set.  */

int function_xy_r_xx(int id)// id is n in other functions
{

  float z0=1.0 +(id-1)*0.1;// this says which funciton.  ie- if n=1, z
starts as .1, or something.
  if ( id==0)
    z0=0;
  int rint=xy_axis[r][id];// random number
    if (verbose)
    cout << "rint % 5 = " << rint%5 << endl;
  float sum = 0.0;//so we can sum this junk
```

```
  float add=4.0/((rint%1000)+1);//add specifies how many times this
will run.  How much is added each turn.
  assert(fabs(add)>1e-10);// assert says this will be assumed, but if
not
  for (float c = -2.0; c<=2.0+add; c += add)
    {
       float z =z0;
      if (verbose)
      cout  << "c======= " << c << "=======" << endl;

      for (int q=0; q <= (rint%5)+1; q++)
      {
        z=z*z + c;//mandelbrot function
        if (verbose)
          cout <<z<<endl;

      }//ends second for loop
      sum = sum+z;

    }
  if (sum <=0)
    {sum*(-1);}
  if ( sum<MAXINT )
    xy=(int)sum;
  else{
    xy=(int)((sum/MAXINT)/(rint+1));
  }
  return xy;
}


int stringlength(char *string_in) // how does this work?  Mrs. Avery
wrote it and now I can't use it.
{
  int countlength = 0;

  while(string_in[countlength] != '\0')
    countlength++;

  return(countlength);
}

int print_array()
{
int p, q;
 for (p = 0; p <msglen; p++)
    {
      cout << endl;
       for ( q = 0; q <size_2; q++)
      { cout << setw(10) << xy_axis [p] [q];}

    }
  cout <<endl;
  return 0;
}
```

	-2	-1.9	-1.8	-1.7	-1.6	-1.5	-1.4	-1.3
1	-2	-1.9	-1.8	-1.7	-1.6	-1.5	-1.4	-1.3
2	2	1.71	1.44	1.19	0.96	0.75	0.56	0.39
3	2	1.0241	0.2736	-0.2839	-0.6784	-0.9375	-1.0864	-1.1479
4	2	-0.85122	-1.72514	-1.6194	-1.13977	-0.621093	-0.219735	0.0176745
5	2	-1.17542	1.17612	0.922458	-0.300917	-1.11424	-1.35172	-1.29969
6	2	-0.518376	-0.416744	-0.849071	-1.50945	-0.258462	0.427138	0.389187
8	2	-1.63129	-1.62632	-0.979078	0.678435	-1.4332	-1.21755	-1.14853
9	2	0.761093	0.844932	-0.741406	-1.13973	0.554054	0.0824355	0.0191282
10	2	-1.32074	-1.08609	-1.15032	-0.301026	-1.19302	-1.3932	-1.29963
12	2	-0.155654	-0.620407	-0.37677	-1.50938	-0.0766949	0.541018	0.389049
13	2	-1.87577	-1.4151	-1.55804	0.678238	-1.49412	-1.1073	-1.14864
14	2	1.61852	0.202494	0.727503	-1.13999	0.732388	-0.173888	0.0193763

	-1.2	-1.1	-1	-0.9	-0.8	-0.7	-0.6	-0.5
1	-1.2	-1.1	-1	-0.9	-0.8	-0.7	-0.6	-0.5
2	0.24	0.11	-2.38E-07	-0.0900002	-0.16	-0.21	-0.24	-0.25
3	-1.1424	-1.0879	-1	-0.8919	-0.7744	-0.6559	-0.5424	-0.4375
4	0.105078	0.0835262	-2.38E-07	-0.104515	-0.200305	-0.269795	-0.305802	-0.308594
5	-1.18896	-1.09302	-1	-0.889076	-0.759878	-0.62721	-0.506485	-0.40477
6	0.213623	0.0946999	-2.38E-07	-0.109543	-0.222586	-0.306607	-0.343473	-0.336161
8	-1.15437	-1.09103	-1	-0.888	-0.750455	-0.605992	-0.482026	-0.386995
9	0.132559	0.0903503	-2.38E-07	-0.111456	-0.236816	-0.332774	-0.367651	-0.350234
10	-1.18243	-1.09184	-1	-0.887577	-0.743918	-0.589261	-0.464833	-0.377336
12	0.198136	0.0921073	-2.38E-07	-0.112206	-0.246586	-0.352771	-0.38393	-0.357618
13	-1.16074	-1.09152	-1	-0.88741	-0.739195	-0.575553	-0.452597	-0.372109
14	0.147322	0.0914074	-2.38E-07	-0.112504	-0.25359	-0.368739	-0.395155	-0.361534

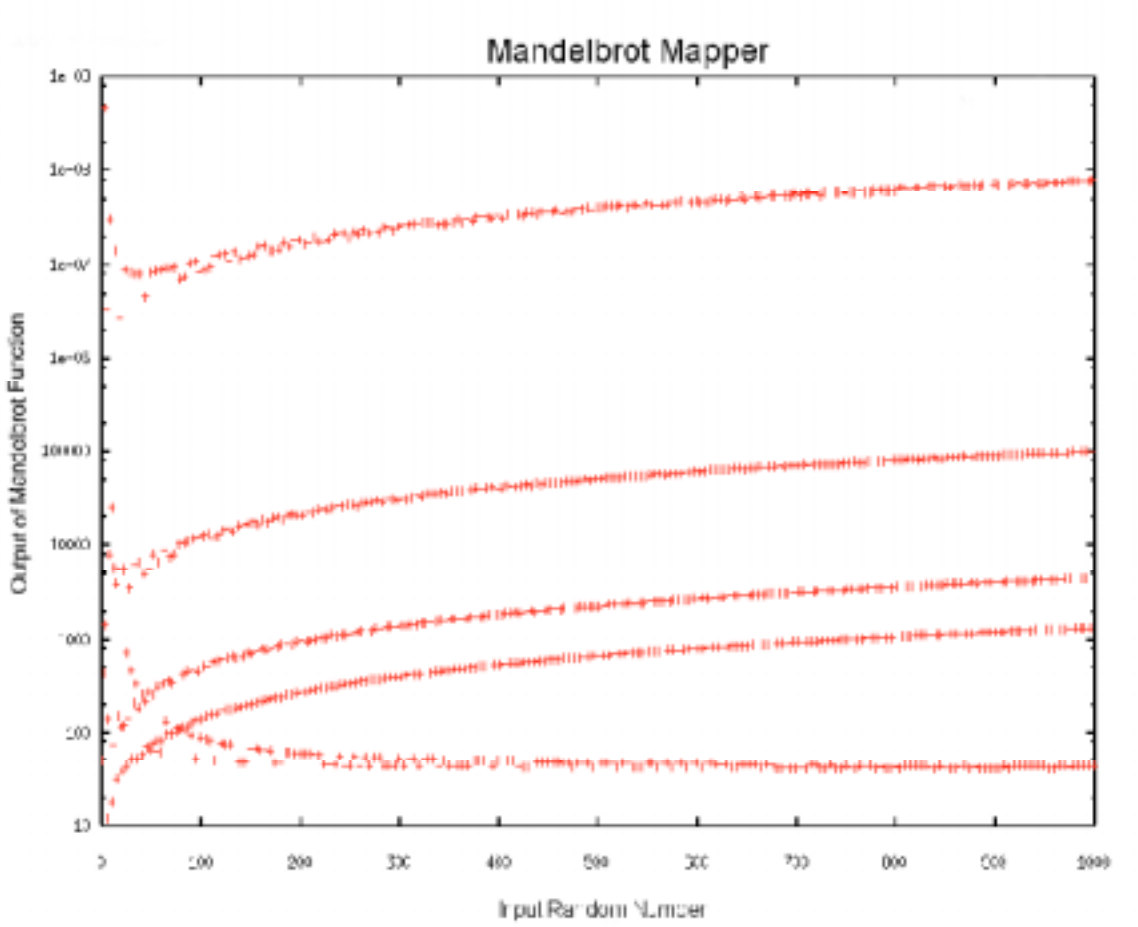
	-0.4	-0.3	-0.2	-0.1	3.11E-07	0.1	0.2	0.3		
1	-0.4	-0.3	-0.2	-0.0999997	3.11E-07	0.1	0.2	0.3		
2	-0.24	-0.21	-0.16	-0.0899998	3.11E-07	0.11	0.24	0.39		
3	-0.3424		-0.2559	-0.1744	-0.0918997	3.11E-07	0.1121	0.257601	0.452101	
4	-0.282762		-0.234515	-0.169584	-0.0915541	3.11E-07	0.112567	0.266358	0.504395	
5	-0.320045		-0.245002	-0.171241	-0.0916175	3.11E-07	0.112672	0.270947	0.554415	
6	-0.297571		-0.239974	-0.170676	-0.0916059	3.11E-07	0.112695	0.273413	0.607376	
8	-0.311451		-0.242412	-0.170869	-0.091608	3.11E-07	0.112701	0.274755	0.668906	
9	-0.302998		-0.241236	-0.170803	-0.0916077	3.11E-07	0.112702	0.275491	0.747436	
10	-0.308192		-0.241805	-0.170826	-0.0916077	3.11E-07	0.112702	0.275895	0.858661	
12	-0.305017		-0.24153	-0.170818	-0.0916077	3.11E-07	0.112702	0.276119	1.0373	
13	-0.306964		-0.241663	-0.170821	-0.0916077	3.11E-07	0.112702	0.276242	1.37599	
14	-0.305773		-0.241599	-0.17082	-0.0916077	3.11E-07	0.112702	0.27631	2.19334	

	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	
1	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	
2	0.560001		0.750001	0.960001	1.19	1.44	1.71	2	2.31
3	0.713601		1.0625	1.5216	2.1161	2.8736	3.8241	5.00001	6.43611
4	0.909227		1.62891	2.91527	5.17789	9.0576	15.5238	26.0001	42.5235
5	1.22669		3.15335	9.09881	27.5105	82.84	241.888	677.003	1809.34
6	1.90478		10.4436	83.3884	757.529	6863.27	58510.5	458334	3.27E+06
8	4.02817		109.568	6954.22	573852	4.71E+07	3.42E+09	2.10E+11	1.07E+13
9	16.6262		12005.8	4.84E+07	3.29E+11	2.22E+15	1.17E+19	4.41E+22	1.15E+26
10	276.83	1.44E+08	2.34E+15	1.08E+23	4.92E+30	1.37E+38	InfinityInfinity		

12	76635.4	2.08E+16	5.47E+30	Infinity	Infinity	Infinity	Infinity	Infinity						
13	5.87E+09	4.32E+32	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity						
14	3.45E+19	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity							

	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9						
1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9						
2	2.64	2.99	3.36	3.75	4.16	4.59	5.04	5.51						
3	8.16961		10.2401		12.6896		15.5625		18.9056					
	22.7681		27.2016		32.2601									
4	67.9425		106.16	162.426		243.692		359.022		520.087				
	741.728		1042.62											
5	4617.38		11271.2		26383.7		59387.2		128899		270493			
	550163		1.09E+06											
6	2.13E+07		1.27E+08		6.96E+08		3.53E+09		1.66E+10					
	7.32E+10		3.03E+11		1.18E+12									
8	4.55E+14		1.61E+16		4.85E+17		1.24E+19		2.76E+20					
	5.35E+21		9.16E+22		1.40E+24									
9	2.07E+29		2.60E+32		2.35E+35		1.55E+38		Infinity	Infinity	Infinity			
	Infinity													
10	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity						
12	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity						
13	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity						
14	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity						

	2
1	2
2	6
3	38
4	1446
5	2.09E+06
6	4.37E+12
8	1.91E+25
9	Infinity
10	Infinity
12	Infinity
13	Infinity
14	Infinity


Mandelbrot Mapper
Output of Mandelbrot Function
Input Random Number