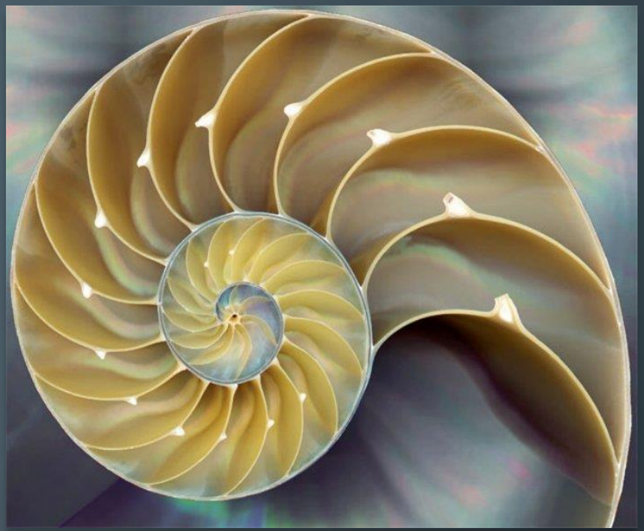
CSEC

WEEK 5: RECURSION

November 11, 2016

COMPUTER SCIENCE ENRICHMENT CLUB

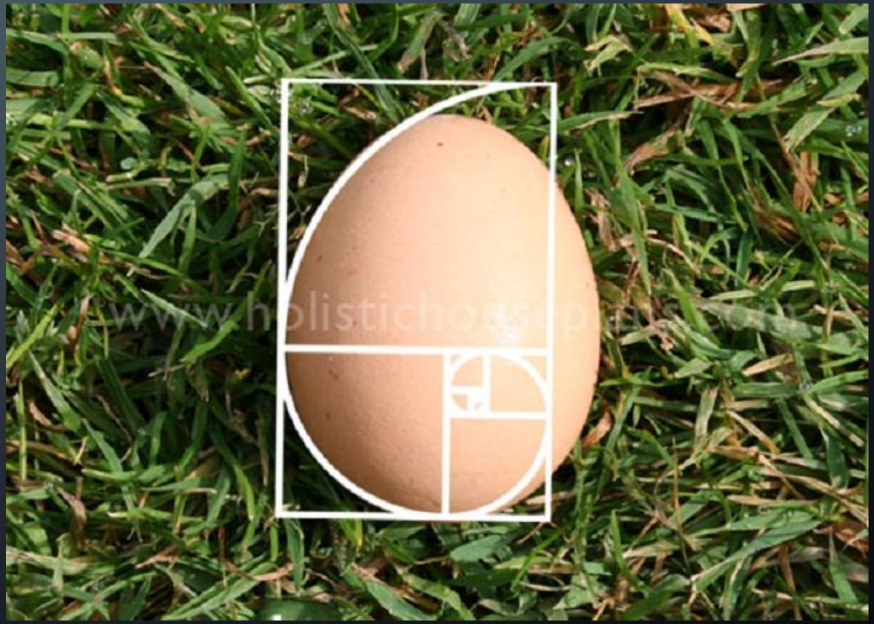
FIBONACCI



- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…. $\mathbf{x_n}$, $\mathbf{x_n = x_{n-1} + x_{n-2}}$
- One of the more famous example of sequences is the Fibonacci sequence, also know as the golden ratio, it is a interesting sequence where the n^{th} number is a sum of the previous two numbers preceding the n^{th} number.

TASK

- Create a function, that takes a integer x such that x>0 and outputs the sequence of Fibonacci numbers up til the xth position.



SOLUTION

```
#include <iostream>

using namespace std;
int main()
{

    int i = 0;
    int k =1;
    int j = 1;
    int sum = 0;
    while(i < 30)
        {
        std::cout << j << " "<<endl;
        sum=j+k;
        j=k;
        k=sum;
        i++;
    }
}

```

IS THERE ANOTHER WAY ?

- Yes, with recursion another solution can be found

```cpp
#include <iostream>

using namespace std;

int fib(int x) {
    if (x == 0)
        return 0;

    if (x == 1)
        return 1;

    return fib(x-1)+fib(x-2);
}
int main() {
    for(int i = 0; i<50;i++){
        cout << fib(i) << " ";
    }
}
```

RECURSION

- What is it?

  Recursion, see Recursion.

- In simple words, if you have a problem, you break it into smaller problems of the same type.

- There are 3 common patterns to recursive decomposition
  - N-1 Approach: Deal with one item (often the first or last), and call the recursion on the remaining N-1
  - Divide and Conquer: Split the problem into 2 or more smaller problems
  - Indirect Recursion: Function 1 calls function 2, which in turn calls function 1

IS THERE A BETTER WAY ?

```cpp
#include <iostream>

using namespace std;

int fib(int x) {
    if (x == 0)
        return 0;

    if (x == 1)
        return 1;

    return fib(x-1)+fib(x-2);
}
int main() {
    for(int i = 0; i<50;i++){
        cout << fib(i) << " ";
    }
}
```

Hmmm…. This looks Familiar

Base Case

Base Case

Recursive Step

RECURSION AND INDUCTION

- Induction:

  -Base Case

  -Induction Hypothesis

  -Induction Step
- Recursion:

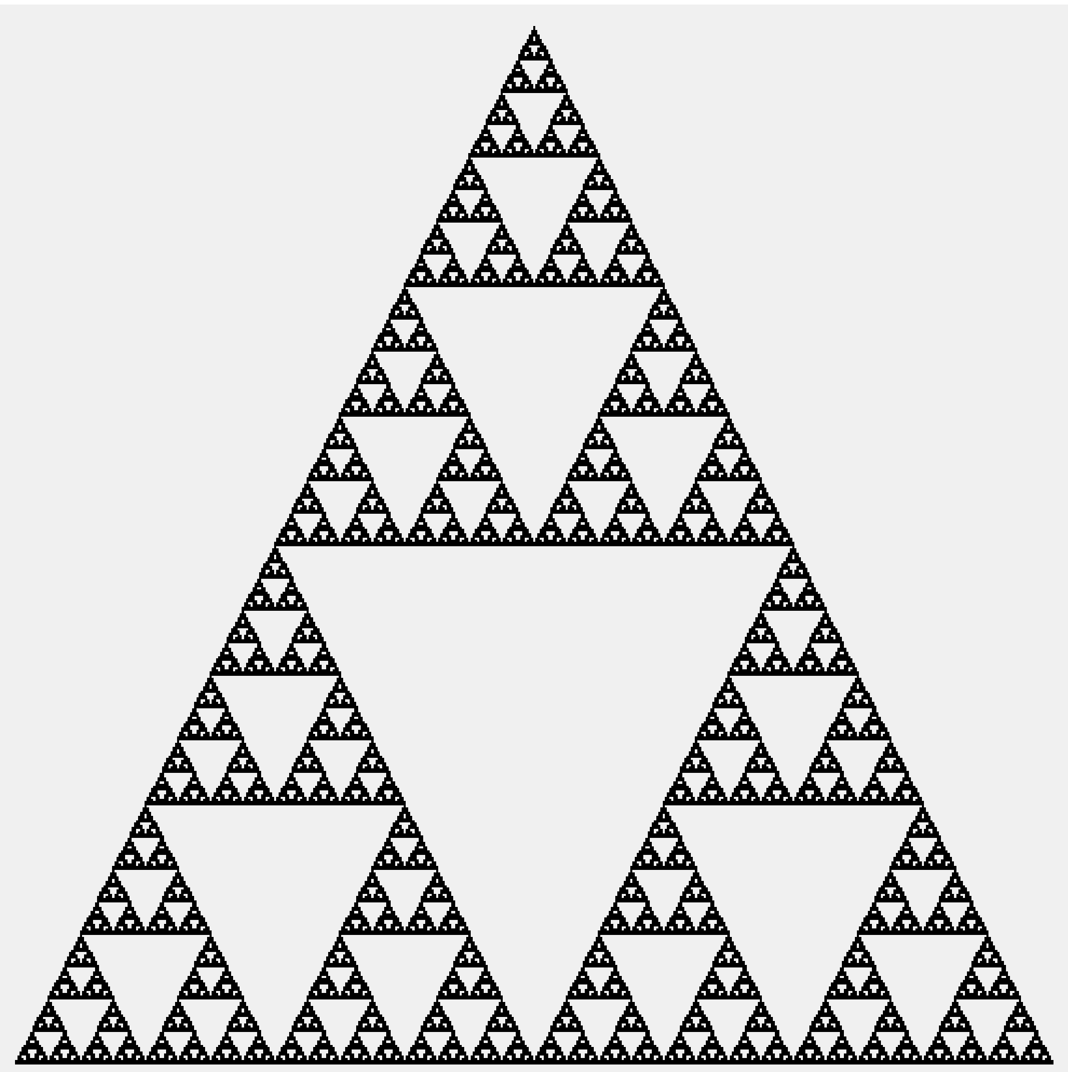
  -Base Case

  -Recursive Step
- While induction is about proving if n exists then surely n+1 exists, recursion is the concept that if a certain task can be done on n elements, then the same task can be done on n-1 elements.

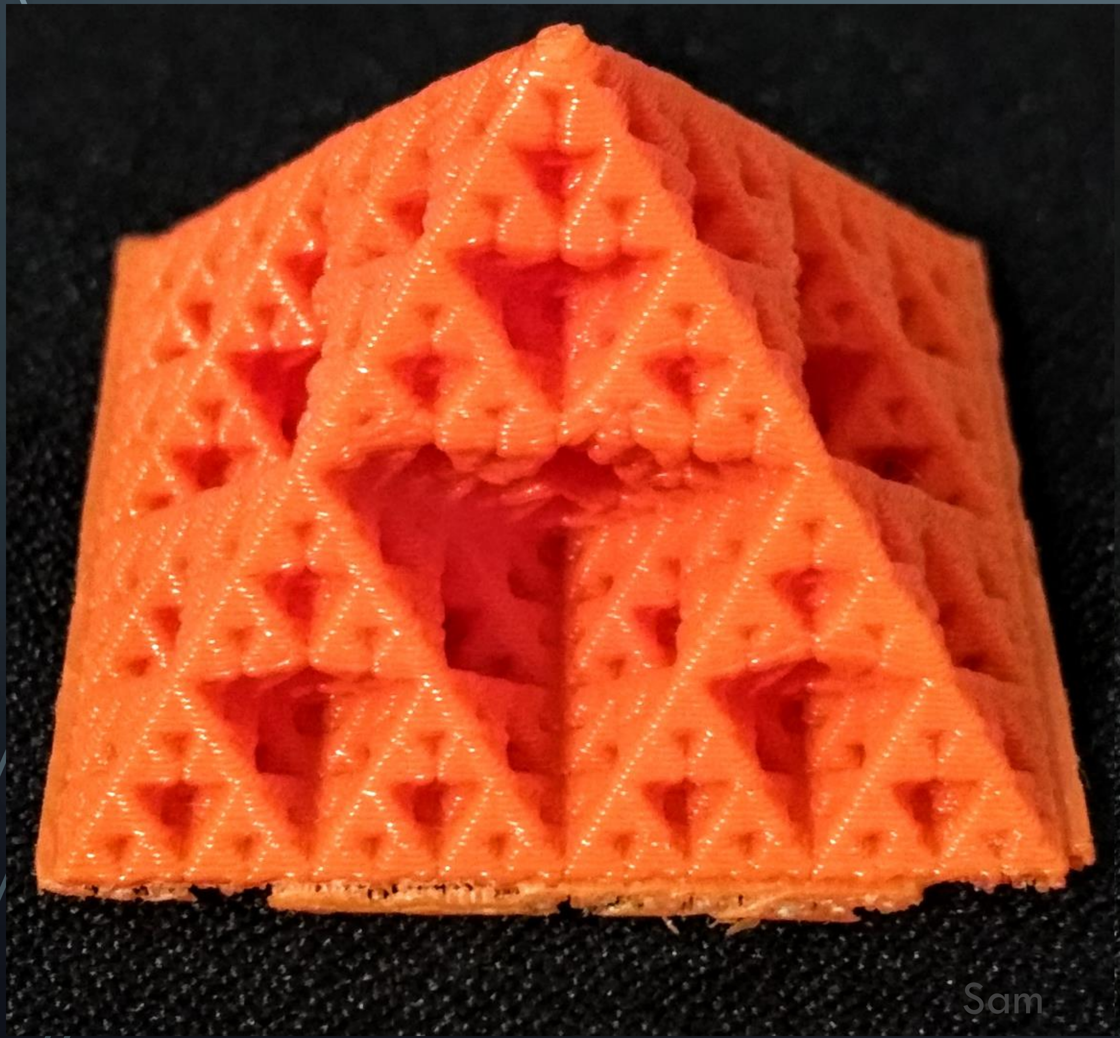
RULES OF RECURSION

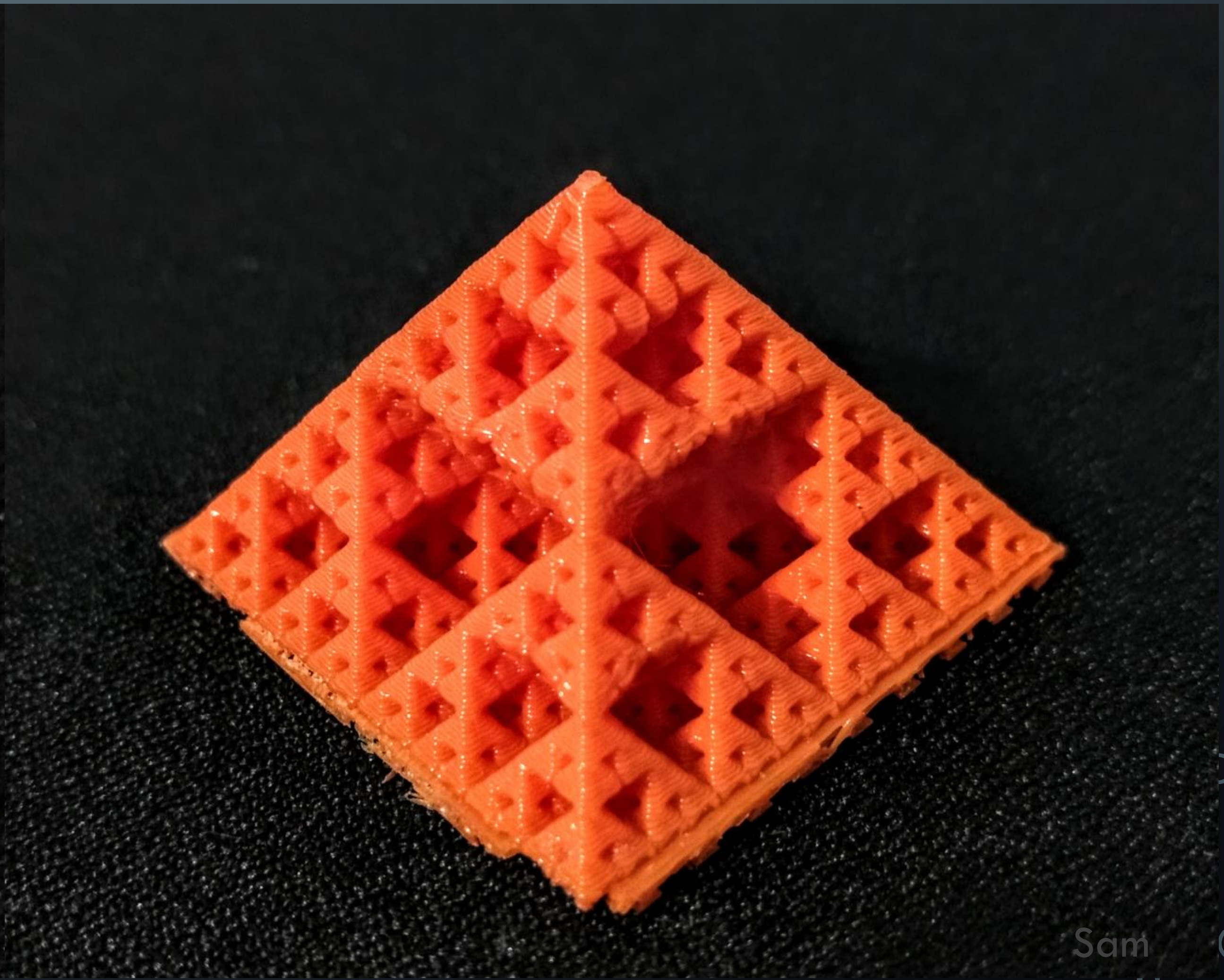
- Recursion consists of 2 phases:
  - Base Case
  - Recursive Decomposition
- Base case should be trivial
- Decomposition must make problems smaller/simpler
- Decomposed problems must be self-similar to original
- Decomposition must eventually lead to base case

DRAWING A SIERPINSKI TRIANGLE USING RECURSION



SOME 3D PRINTED EXAMPLES OF RECURSION (SIERPINSKI PYRAMID)





SO WHY USE RECURSION

- Anything that is recursive can be done iteratively, so why do it recursively?
- Less error prone – less code to go wrong
- Makes some specific problems easier, in particular tree traversal is a lot easier with it
- Reduce time complexity of specific problems

TASK

- Write the following program:

INPUT: an integer x

OUTPUT: x factorial ie 5 factorial = 5! = 5 x 4 x 3 x 2 x 1

Additional Constrains: Must be recursive

SOLUTION

Finding the factorial of a number with N-1 recursive approach

```
#include <iostream>
using namespace std;

int factorial(int);

int main()
{
    int n;
    n = 10;
    cout << "Factorial of " << n <<" = " << factorial(n)<<endl;
    return 0;
}

int factorial(int n)
{
    if (n > 1)
    {
        return n*factorial(n-1);
    }
    else
    {
        return 1;
    }
}
```

```
Factorial of 10 = 3628800
[Finished in 0.4s]
```

THINGS THAT CAN GO WRONG WITH RECURSION

- Base cases not being properly defined or not reached causing stack overflow error
- Implementation
- Sometimes the problem is inherently iterative
- Stack overflow

ADVANCED PROBLEM

- Given a array, and a element to search for, find the index of that element
- Constraint: Must use recursive Divide and Conquer technique.
- Bonus: Do it in O(nlogn)

SOLUTION

```cpp
#include <cstdlib>
#include <iostream>
using namespace std;

int binary_search(int array[],int first,
        int last, int value);

int main() {

int list[10];

 for (int k=0; k<11; k++)
 list[k]=2*k+1;

 for (int k=0; k<11; k++)
 cout<<list[k] << " ";

 cout<< endl<<"binary search results: "
 << binary_search(list,1,21,11)<<endl;

 return 0;
}

int binary_search(int array[],int first,
    int last, int search_key)
{
 int index;

 if (first > last)
 index = -1;

 else
 {
 int mid = (first + last)/2;

 if (search_key == array[mid])
 index = mid;
 else

 if (search_key < array[mid])
 index = binary_search(array,first, mid-1, search_key);
 else
 index = binary_search(array, mid+1, last, search_key);

 }
 return index;
 }
```

```
1 3 5 7 9 11 13 15 17 19 21
binary search results: 5
```

ANOTHER ONE

INPUT: