1 Magic Penny

1.1 Closed Form: Magic Penny

day	value
0	\$0.01
1	\$0.02
2	\$0.04
3	\$0.08
4	\$0.16
5	\$0.32
6	\$0.64
...	...
31	\$21,474,836.48

Let's start with a topic we are already familiar with. The magic doubling penny. It will double in value every day. After 31 days it will be worth several million dollars as partially shown in the table above. The astute developer, might recognize a mathematical pattern of powers of 2. Specific to this problem of pennies, we solve this in closed form as $\frac{2^n}{100}$ for some n number of days. In the interest of time, that code is provided below:

```
#include<cmath>
double magicPennyClosed( unsigned short days )
{
    return pow( 2, days ) / 100;
}
```

1.2 Loop: Magic Penny

Now of course, we might not recognize these powers of 2 immediately. Instead, we might solve this using a loop. In the interest of time, that code is provided below

```
double magicPennyLoop( unsigned short days )
{
    double result = 0.01;
    for ( int i = 0 ; i < days ; i++ )
    {
        result *= 2;
    }
    return result;
}
```

1.3 Recursion: Magic Penny

Now, let us try to solve this same problem a different way. Let us use recursion. We will work toward a function named *magicPennyRec*. The type signature shouldn't need to change:

```
double magicPennyRec( unsigned short days );
```

1. What is the base case? Put another way, for what values of *days* do we trivially know the final value of the magical doubling penny?

2. What can we do to a large value of *days* to guarantee that we arrive at the base case?

3. Together, let us fill in the provided scaffolding:

```
double magicPennyRec( unsigned short days )
{
        // BASE CASE

        if (                                    )
        {


        }

        // RECURSIVE CASE
        else
        {






        }
}
```

2 Famous recursive problems

A lecture on recursion is not complete without discussion of factorial and Fibonacci.

2.1 Factorial

Factorial is often easier to understand so we'll start there. Formally, the factorial of a natural number $n \in \mathbb{N}_0$, is denoted "$n\,!$" and is defined as follows for all

$$n\,! = \begin{cases} 1 & n \leq 1 \\ n \ * \ (\,(n-1)\,!\,) & n \geq 2 \end{cases}$$

4. What is the value of $0\,!$?
5. What is the value of $1\,!$?
6. What is the value of $2\,!$?
7. What is the value of $3\,!$?
8. What is the value of $-7\,!$?
9. let's turn it into code

```
unsigned int fact( unsigned short n )
{
        // BASE CASE




        // RECURSIVE CASE




}
```

10. let's walk through the code. To make the walkthrough consistent, I have pre-written one version of the solution. Let us construct and fill in a value table together to demonstrate that $fact(5)$ will evaluate to 120

```
unsigned int fact( unsigned short n )
{
    unsigned int fn;
    // BASE CASE
    if ( n <= 1 )
    {
        fn = 1;
    }
    // RECURSIVE CASE
    else
    {
        unsigned int nm1, fnm1;
        nm1 = n - 1;
        fnm1 = fact( nm1 );
        fn = n * fnm1;
    }
    return fn;
}
```

2.2 Fibonacci

It would appear that we have already solved this for recitation this week, regardless, let us review. The Fibonacci Sequence is a famous sequence that occurs in the natural world. Much like the factorial value of a natural number, the Fibonacci sequence is easily described using recursion. Below is a mathematic definition for the Fibonacci sequence value for any $x \in \mathbb{N}_0$ (note that some prefer to use $y \in \mathbb{N}_1$)

$$fib(x) = \begin{cases} x & x \leq 1 \\ fib(x-1) + fib(x-2) & n \geq 2 \end{cases}$$

11. What is the value of $fib(-5)$?
12. What is the value of $fib(0)$?
13. What is the value of $fib(1)$?
14. What is the value of $fib(2)$?
15. What is the value of $fib(3)$?
16. let's turn it into code

```
// your code here
```

3 Merge Sort

Let us end our discussion of recursion with a re-visiting of the merge-sort algorithm. We will write a few functions.

17. first, describe a function named foo given 2 vectors of numbers that we assume are sorted in ascending order, return a single vector of all of these numbers in ascending order. You don't need to write full code, just describe it.

18. Is the above function recursive?

19. What is the complexity of the above function in terms of the result vectors length n? Why?

20. Review the following code and then describe in your own words what the insert function does for us here.

```
vector< int > x = { 0, 100 };
vector< int > y = { 5, 12, 2, 18, 7, 9 };
x.insert( x.begin() + 1, y.begin() + 2 , y.begin() + 5);
// x is now { 0, 2, 18, 7, 100 }
```

21. finally, we will use the provided scaffolding to implement mergeSort. We should assume that the *foo* function has been written for us and we can call it as we see fit. We can tackle this in any order we want.

```
vector<int> mergeSort( vector<int> v )
{
    // base case




    // split




    // recurse




    // merge




}
```