Final on Thursday
Statements are sentences that claim certain things. Can be either true or false, but not both.
Logical Implications

- “If-Then” statements
All men are mortal. Socrates is a man. Socrates is mortal.
Quantifiers and Limits

- Universal
- Existential
Direct Proofs

- The method of the proof is to take an original statement $p$, which we assume to be true, and use it to show directly that another statement $q$ is true.
An irrational number can be written as a \textit{decimal}, but not as a fraction. An irrational number has endless \textit{non-repeating} digits to the right of the \textit{decimal} point.
It is a simple idea that comes directly from **long division**.

The Quotient Remainder theorem says:
Given any integer $A$, and a positive integer $B$, there exist unique integers $Q$ and $R$ such that

$$A = B \times Q + R$$

where $0 \leq R < B$

We can see that this comes directly from long division. When we divide $A$ by $B$ in long division, $Q$ is the quotient and $R$ is the remainder.
If we can write a number in this form then $A \mod B = R$
Any integer larger than 1 can be factorized into primes
Greatest Common Divisor / Euclid’s Algo

- **GCD**: Greatest number that divides two numbers without leaving a remainder

- **Euclid’s Algorithm**:
  - if \( m < n \), swap \((m,n)\)
  - while \( n \) does not equal 0
    - \( r = m \mod n \)
    - \( m = n \)
    - \( n = r \)
  - endwhile
  - output \( m \)
Asymptotic Notation

- Helps with questions like:
  - How long will a program run on an input?
  - How much space will it take?
  - Is the problem solvable?

- **Big O of n**: $O(g(n))$ is an upper-bound on the growth of a function, $f(n)$
- **Big Omega of n**: $\Omega(g(n))$ is lower-bound on the growth of a function, $f(n)$
- **Theta of n**: $\Theta(g(n))$ is tight bound on the growth of a function, $f(n)$.

Asymptotic Notation

- 1 (constant running time):
  - Instructions are executed once or a few times

- $\log N$ (logarithmic)
  - A big problem is solved by cutting the original problem in smaller sizes, by a constant fraction at each step

- $N$ (linear)
  - A small amount of processing is done on each input element

- $N \log N$
  - A problem is solved by dividing it into smaller problems, solving them independently and combining the solution
Asymptotic Notation

- $N^2$ (quadratic)
  - Typical for algorithms that process all pairs of data items (double nested loops)
- $N^3$ (cubic)
  - Processing of triples of data (triple nested loops)
- $N^K$ (polynomial) and $2^N$ (exponential)
  - Few exponential algorithms are appropriate for practical use

University of Arizona
Sequences and Series

- **Sequences**: an ordered list of numbers
- **Series**: is the value you get when you add up all the terms of a sequence
Mathematical Induction

- **Mathematical induction** is a method of mathematical proof typically used to establish a given statement for all natural numbers. It is a form of direct proof, and it is done in two steps. The first step, known as the base case, is to prove the given statement for the first natural number.

1. Give the negation of “if I hit my thumb with a hammer, then my thumb will hurt”
   1. Hint, it is not if I hit my thumb with a hammer my thumb won’t hurt
2. Of the following, which are rational?
   a) $\frac{16}{9}$
   b) $\frac{2}{\sqrt{5}}$
3. For the following, give the GCD:
   a) (42, 56)
4. Give the prime factorization for the following:
   a) 48
   b) 180
1. Directly prove that if \( n \) is an odd integer then \( n^2 \) is also an odd integer

2. Prove by Induction that for \( n \geq 1 \),
   \[
   1 \times 2 + 2 \times 3 + 3 \times 4 + \ldots + (n)(n+1) = (n)(n+1)(n+2)/3
   \]
1. The following is code for linear (non recursive) Fibonacci sequence at position n given below:

```python
def LinearFibonacci(n):
    fn = f1 = f2 = 1
    for x in xrange(2, n):
        fn = f1 + f2
        f2, f1 = f1, fn
    return fn
```

2. What is the Big – O notation of the time complexity for Linear Fibonacci?
1. The following is code for Recursive Fibonacci sequence at position n given below:
   ```python
   def fibonacci(n):
       if n < 2:
           return n
       else:
           return fibonacci(n - 1) + fibonacci(n - 2)
   ```

2. What is the Big-O notation/time complexity of Recursive Fibonacci?
1. The following is code for Bubblesort is given to the right:

2. What is the Big – O notation of Bubblesort?

```plaintext
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
        until not swapped
    end procedure
```