Algorithm Analysis 2

Recursion

```
long factorial( int n ){
        if(n <= 1)
                return 1;
        else
                return n * factorial(n - 1);
// f(n) = 1 + t(n-1)
}
```

Expanding recurrence form results in a simulation of the function.

$$T(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = \ldots$$

Which leads to...

$$T(n) = n - 1 + T(1) = O(n)$$

$$T(n) > 2^{n-1/n} * T(1), T(n) = \Omega(2^{n-1/n})$$

Binary Search

```
int binary_search(const vector<int> &a, int X){
        unsigned int low=0, high = s.size()-1;
        while (low <= high){
                int mid = (low + high)/2;
                if(a[mid] < x)
                        low = mid + 1;
                else
                        return mid;
        }
```

```
        return NOT_FOUND
}
// log(n) time complexity
```

Euclid's Algorithm

- Find the GCD (greatest common divisor) between n and m - Given M >= N - Why is it O(logN)

```
long gcd(long m, long n){
        while(n!=0){
                long rem = m %n;
                m = n;
                n = rem;
        }
        return m;
}
// LogN time complexity
```

Exponential

- Calculate x^0 - Complexity O(logN)

```
long pow(long x, int n){
        if(n == 0){
                return 1;
        }
        if(n == 1){
                return x;
        }
        if (isEven(n)){
                return pow(x * x, n/2);
        }
```

```
        else
                return pow(x * x, n/2) * x;
}
// O(logN)
```