Here is a review of some formulas that you will find useful when doing asymptotic analysis.

- $\sum_{i=1}^{N} i = 1 + 2 + 3 + 4 + \cdots + N = \frac{N(N+1)}{2} = \frac{N^2 + N}{2}$
- $\sum_{i=0}^{N-1} 2^i = 1 + 2 + 4 + 8 + \cdots + 2^{N-1} = 2 \cdot 2^{N-1} - 1 = 2^N - 1$

Intuition

For the following recursive functions, give the worst case and best case running time in the appropriate $O(\cdot)$, $\Omega(\cdot)$, or $\Theta(\cdot)$ notation.

1.1 Give the running time in terms of $N$.

```java
public void andslam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) {
            System.out.println("datboi.jpg");
        }
        andslam(N / 2);
    }
}
```
1.2 Give the running time for `andwelcome(arr, 0, N)` where N is the length of the input array arr.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[");
    for (int i = low; i < high; i += 1) {
        System.out.print("loyal ");
    }
    System.out.println("]");
    if (high - low > 0) {
        double coin = Math.random();
        if (coin > 0.5) {
            andwelcome(arr, low, low + (high - low) / 2);
        } else {
            andwelcome(arr, low, low + (high - low) / 2);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```

1.3 Give the running time in terms of N.

```java
public int tothe(int N) {
    if (N <= 1) {
        return N;
    }
    return tothe(N - 1) + tothe(N - 1);
}
```

1.4 Give the running time in terms of N.

```java
public static void spacejam(int N) {
    if (N <= 1) {
        return;
    }
    for (int i = 0; i < N; i += 1) {
        spacejam(N - 1);
    }
}
```
Hey you watchu gon do

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms guaranteed to be faster? If so, which? And if neither is always faster, explain why.

(a) Algorithm 1: \( \Theta(N) \), Algorithm 2: \( \Theta(N^2) \)

(b) Algorithm 1: \( \Omega(N) \), Algorithm 2: \( \Omega(N^2) \)

(c) Algorithm 1: \( O(N) \), Algorithm 2: \( O(N^2) \)

(d) Algorithm 1: \( \Theta(N^2) \), Algorithm 2: \( O(\log N) \)

(e) Algorithm 1: \( O(N \log N) \), Algorithm 2: \( \Omega(N \log N) \)

Would your answers above change if we did not assume that \( N \) was very large (for example, if there was a maximum value for \( N \), or if \( N \) was constant)?

Asymptotic Notation

Draw the running time graph of an algorithm that is \( O(\sqrt{N}) \) in the best case and \( \Omega(N) \) in the worst case. Assume that the algorithm is also trivially \( \Omega(1) \) in the best case and \( O(\infty) \) in the worst case.

Extra: Following is a question from last week, now that you have properly learned about \( O(\cdot) \), \( \Omega(\cdot) \), or \( \Theta(\cdot) \).
Are the statements in the right column true or false? If false, correct the asymptotic notation ($\Omega(\cdot), \Theta(\cdot), O(\cdot)$). Be sure to give the tightest bound. $\Omega(\cdot)$ is the opposite of $O(\cdot)$, i.e. $f(n) \in \Omega(g(n)) \iff g(n) \in O(f(n))$.

\[
\begin{array}{ccc}
 f(n) = 20501 & g(n) = 1 & f(n) \in O(g(n)) \\
 f(n) = n^2 + n & g(n) = 0.000001n^3 & f(n) \in \Omega(g(n)) \\
 f(n) = 2^{2n} + 1000 & g(n) = 4^n + n^{100} & f(n) \in O(g(n)) \\
 f(n) = \log(n^{100}) & g(n) = n \log n & f(n) \in \Theta(g(n)) \\
 f(n) = n \log n + 3^n + n & g(n) = n^2 + n + \log n & f(n) \in \Omega(g(n)) \\
 f(n) = n \log n + n^2 & g(n) = \log n + n^2 & f(n) \in \Theta(g(n)) \\
 f(n) = n \log n & g(n) = (\log n)^2 & f(n) \in O(g(n))
\end{array}
\]

Fall 2015 Extra

4.1 If you have time, try to answer this challenge question. For each answer true or false. If true, explain why and if false provide a counterexample.

(a) If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions (that is for all $n, f(n), g(n) > 0$), then $\frac{f(n)}{g(n)} \in O(n)$.

(b) If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $\frac{f(n)}{g(n)} \in \Theta(n)$. 