1. If \( f, g : \mathbb{N} \rightarrow [0, \infty) \):

   (a) \( f(n) = O(g(n)) \) means \( \exists C > 0, N \) such that \( \forall n \geq N, |f(n)| \leq C \cdot g(n) \)  
      (i.e., \( f(n) \) is eventually no larger than some constant multiple of \( g(n) \))

   (b) There must be some constant multiple of \( g \) \( (\exists C > 0) \)
      that \( f(n) \) is no larger than \( (f(n)) \leq C \cdot g(n) \)
      when \( n \) is sufficiently large \( (\exists N, \forall n \geq N) \)

   (ii) If we plot the graphs of \( f, g, \) and \( C \cdot g \):

   (iii) The point of Big-O notation is that it allows us to think about
        the overall growth rate of a function without worrying about
        little details (see Problem #4!).

   (c) (i) Nothing here equals anything \( (\text{the } = \text{ is used as a description, or better yet, a } \epsilon) \).

   (ii) What this is really about is the whole function, not just
        an individual value \( f(n) \) vs. \( g(n) \).

        (All in all, this should be written \( f \in O(g) \)
        to better reflect the actual concept)

   (d) \( f_1(n), f_2(n) = O(g(n)) \Rightarrow f_1(n) \pm f_2(n) = O(g(n)) \)
      (Similarly for constant multiples: \( f(n) = O(g(n)) \Rightarrow A \cdot f(n) \in O(g(n)) \)

   (e) \( f_1(n) = O(g_1(n)) \land f_2(n) = O(g_2(n)) \Rightarrow f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \)
      \( \text{both bounds are used!} \)
2. (a) \( f(n) = \Theta(g(n)) \) means \( \exists c > 0, N \) such that \( \forall n \geq N, f(n) = c \cdot g(n) \)

The opposite inequality! \( \nabla \)

\( f \) is eventually \underbrace{\text{smaller than}}_{\text{some positive multiple of } g}

(b) \( f(n) = \Theta(g(n)) \) means both \( f(n) = \mathcal{O}(g(n)) \) \& \( f(n) = \Omega(g(n)) \)

I.e., \( f \) is eventually between two positive multiples of \( g \), so \( f \& g \) are roughly comparable in size.

3. When \( n \) is large, a few basic categories give us a starting point for understanding \( \mathcal{O}, \Omega, \) and \( \Theta \):

\[
\begin{array}{l|l|l|l|l|l|l|l|l}
\text{(exponentials, base > 1)} & \text{(positive powers of } n) & \text{logs} & \text{logs} & \text{logs} & \text{logs} & \text{logs} & \text{logs} & \text{logs} \\
\hline
\mathcal{O}(n^{10}) & \mathcal{O}(n^{9}) & \ldots n & \ldots \sqrt{n} & \ldots n & n^{2} & n^{3} & \ldots 1.0 & 2.0 & 10.0 & 100.0 & \ldots n! & n^{n} \\
\hline
\mathcal{O}(n) & \mathcal{O}(\log n) & \log \log n & \log n & \log n & \log n & \log n & \log n & \log n & \log n & \log n & \log n & \log n \\
\hline
\mathcal{O}(1) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) & \mathcal{O}(\log n) \\
\hline
\mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) & \mathcal{O}(c) \\
\hline
\mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) & \mathcal{O}(n) \\
\hline
\end{array}
\]

* Anything farther left is \( \mathcal{O}(\text{anything farther right}) \)

* Anything farther right is \( \Omega(\text{anything farther left}) \)

* Other than nonzero constants, none of these are \( \Theta(\text{any other}) \)

\( \mathcal{O} \) due to our rule for \( \text{Big-O} \), "smaller" summands

\( \& \) constant factors are simply absorbed into larger ones!

\( \Theta \) don't forget that our \( \cdot \) rule means that non-constant factors aren't absorbed!
4. (a) \[ \frac{5n^3 + 3n + 1000}{n^3} = O(n) \]

(b) \[ \frac{\log(n^3) + 2n}{n^3} = O(n) \]
\[ \text{NOTE THAT } \log(n^3) = 3 \log n \]

(c) \[ \frac{8n^{100} + 2^n + \log n}{n^{100}} = O(2^n) \]

(d) \[ \frac{100^n + 4n^m + n!}{n^{100}} = O(n!) \]

(e) \[ \frac{100n + n^4}{n^{100}} \left( n^2 + 2^n \right) = O(n^{100}) \]

(f) \[ \left( \frac{\log \log n + 10000}{n^{100}} \right) \left( \frac{\log n + n + n^n}{n^{100}} \right) = O(n \cdot \log \log n) \]

5. (a) CLAIM: \[ 1000 = O(\log n) \], i.e., \( \exists C > 0, N \) such that \( \forall n \geq N, \ 1000 \leq C \log n \)

PROOF: TAKE \( C = 1000 > 0 \)
\( \& \quad N = 2. \)

Let \( n \geq 2 \) be given \( \Rightarrow \) so \( n \geq 2. \)

THEN \( C \cdot \log n \geq 1000 \cdot \log 2 = 1000 \cdot 1 = 1000. \]

(b) CLAIM: \[ 100n = O(n^2) \], i.e., \( \exists C > 0, N \) such that \( \forall n \geq N, \ 100n \leq C \cdot n^2 \)

PROOF: TAKE \( C = 100 > 0 \)
\( \& \quad N = 1. \)

Let \( n \geq 2 \) be given \( \Rightarrow \) so \( n \geq 1. \)

THEN \( n \leq n^2, \) so \( 100n \leq 100n^2 = C \cdot n^2. \)
(c) CLAIM: \( 3^n = \Omega(100 \cdot 2^n) \), i.e., \( \exists C > 0, N \) such that \( \forall n \geq N, \ 3^n \geq C \cdot 100 \cdot 2^n \)

PROOF: TAKE \( C = \frac{1}{100} > 0 \)
\[ \text{& } N = 1. \]

LET \( n \geq N \) BE GIVEN \( \text{& } n \geq 1. \)

THEN \( 3^n \geq 2^n = \frac{1}{100} \cdot 100 \cdot 2^n = C \cdot 100 \cdot 2^n \), \( \square \)

---

(d) \( 100n^2 = \Theta(n^2) \) MEANS \( (\exists C_1 > 0) \) \& \( (\exists C_2 > 0) \) \( \forall n \geq N \), \( C_1 \cdot n^2 \leq 100n^2 \leq C_2 \cdot n^2 \)

So this is two little sub-proofs:

(i) CLAIM: \( 100n^2 = \Theta(n^2) \), i.e., \( \exists C > 0, N \) so that \( \forall n \geq N, \ 100n^2 \leq C \cdot n^2 \)

PROOF: TAKE \( C = 100 > 0 \)

AND \( N = 1. \)

LET \( n \geq N \) BE GIVEN \( \text{& } n \geq 1. \)

THEN \( 100n^2 = C \cdot n^2, \) \( \text{so } 100n^2 \leq C \cdot n^2 \), \( \square \)

(ii) CLAIM: \( 100n^2 = \Omega(n^2) \), i.e., \( \exists C > 0, N \) so that \( \forall n \geq N, \ 100n^2 \geq C \cdot n^2 \)

PROOF: TAKE \( C = 100 > 0 \)

AND \( N = 1. \)

LET \( n \geq N \) BE GIVEN \( \text{& } n \geq 1. \)

THEN \( 100n^2 = C \cdot n^2, \) \( \text{so } 100n^2 \geq C \cdot n^2 \), \( \square \)

\( \square \) NOTE THAT IN GENERAL, THE "\( C \)" IN THESE TWO SECULARS OF A \( \Theta \)-\( \Omega \) PROOF COULD BE DIFFERENT — IN THIS SIMPLE EXAMPLE, THEY END UP THE SAME!
6. (a) Comparing a to each of the 1023 elements and never finding it
will make 1023 comparisons.

(b) Starting with 1023 elements:

\[ \begin{array}{c}
\text{10 comparisons)} \\
\text{This is exactly } \\
\log(1023+1)
\end{array} \]

(i) \( a_{511}, a_{254}, a_{935}, a_{439}, a_{941}, a_{1003}, a_{1024}, a_{1022}, a_{1022} \)

\[ \frac{0 + 1022}{2} \quad \frac{512 + 1022}{2} \quad \frac{743 + 1022}{2} \quad \frac{950 + 1022}{2} \quad \frac{951 + 1022}{2} \quad \frac{1024 + 1022}{2} \quad \frac{1022 + 1022}{2} \]

(MIDDLE ELEMENTS OF THE REST, AT EACH STEP)

(ii) \( a_{511}, a_{255}, a_{127}, a_{63}, a_{31}, a_{15}, a_{7}, a_{3}, a_{1}, 90 \)

\[ \frac{0 + 1022}{2} \quad \frac{510 + 1022}{2} \quad \frac{1034}{2} \quad \frac{272}{2} \quad \frac{42}{2} \quad \frac{14}{2} \quad \frac{6}{2} \quad \frac{3}{2} \quad \frac{1}{2} \]

(c) As in part (a), the linear search will make \( n = 2^{k-1} \) comparisons;
As in part (b), the binary search will make \( \log(n+1) \) comparisons.

\[ \text{\textbf{\( \therefore \) The linear search is } } O(n), \text{ and the binary search is } O(\log n) \]

\[ \text{\( (\log(n+1)) \leq \log(n+n) \)} \]
\[ \text{\( = \log(2n) = 1 + \log n, \)} \]
\[ \text{which is } O(\log n) \]

7. E.G.: 1 3 2 9 1 0 1 5 2 3 1 4 1 6 1 2 0 2 2 3 5 4

Goal: Find \( \gamma \), path from top-left with maximum sum of \( 1 \)'s encountered.

(a) Brute Force Search (of all such paths):

(i) There are \( (9^9) \) = 70 such paths to search.

(ii) Each path encounters 9 numbers, so the sum takes 8 additions.

(iii) In total, this gives 70 x 8 = 560 additions.
(b) **Dynamic Programming**: For the example above:

(i) 3 + 2 + 4
    1 + 2 + 4
    1 + 12

(ii) 4(1 + 2 + 3 + 4) = 40 additions

(iii) **The Dynamic-Programming Algorithm is Quite a Bit More Efficient**!

What makes this possible is that many, many of the 360 additions involved the same numbers (there are only 25 #’s in the grid!), and that knowing the maximum sum possible to each point along the way as we build downward & rightward dictates the rest of the maximum sums to other points past it.

(d) For a general n x n grid:

- **Brute-Force Checks** \( \binom{2(n-1)}{n-1} \) paths, each with \( 2(n-1) \) additions, so \( \frac{(2n-2)!}{(n-1)!2^{n-2}} \) additions.

- **Dynamic Programming** takes \( 4(1 + 2 + \ldots + n-1) = 4 \frac{(n-1)n}{2} = 2(n^2 - n) \) additions.

(e) With a Little Computational Help:

- **Brute-Force** for \( n = 11 \) gives < 1 billion additions, and for \( n = 12 \) gives > 1 billion. \( \therefore n = 12 \).

- **Dynamic Programming** for \( n = 22,361 \) gives < 1 billion additions, and for \( n = 22,362 \) gives > 1 billion. \( \therefore n = 22,362 \).