Name: Eileen Chang

	CS5008
Spring 2021	March 18

Assignment 8, Primer on proofs

We want to continue to get more comfortable with the mathematical notation used in Algorithms. In problem 1, you are going to write *in plain English* what the expression is and then solve the statement.

Problem 1 - Quantifiers

Write the following statements as English sentences, then decide whether those statements are true if x and y can be any integers. When deciding if x and y can be any integers, prove your claim with a convincing argument.

1. $\forall x \exists y : x + y = 0$

   For every integer x, there exists integer y, such that $x + y = 0$.

   True - every integer can be added to its opposite sign in order to get a sum of 0. For instance, $5 + (-5) = 0$.

2. $\exists y \forall x : x + y = x$

   There exists an integer y, such that for every integer x, $x + y = x$.

   True - the integer 0 is an integer (y) that can make the statement $x + y = x$ true for any integer x.

3. $\exists x \forall y : x + y = x$

   There exists an integer x, such that for every integer y, $x + y = x$.

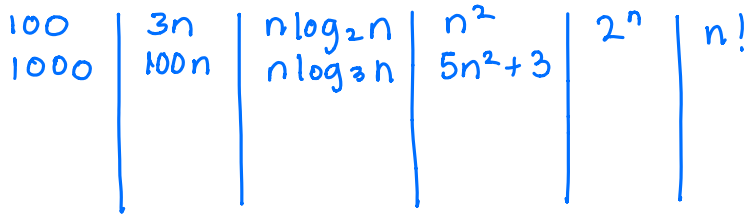
   True - the integer 0 is an integer (x) that can make the statement $x + y = x$ true for any integer y.

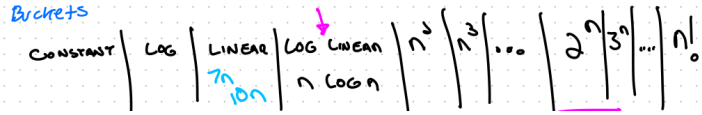
In problem 2 and problem 3, we want to solidify our understanding of Big-O notation. Remember, Big-O notation is about the growth of a function as n grows asymptotically large.

Problem 2 - Growth of Functions

Organize the following functions into six columns. Items in the same column should have the same asymptotic growth rates (they are big-O and big-Θ of each other). If a column is to the left of another column, all its growth rates should be slower than those of the column to its right.

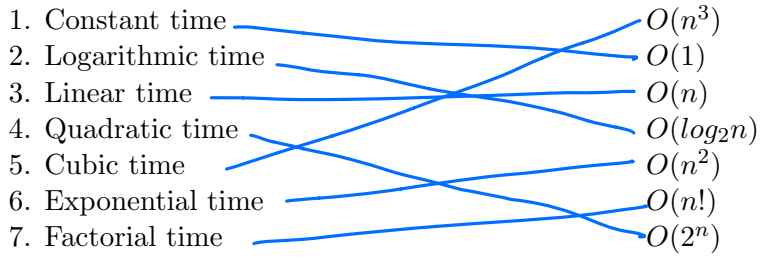
n^2, $n!$, $n \log_2 n$, $3n$, $5n^2 + 3$, 2^n, 10000, $n \log_3 n$, 100, $100n$





Problem 3 - Function Growth Language

Match the following English explanations to the *best* corresponding Big-O function by drawing a line from the left to the right.



Def:
$f(n) = O(g(n))$
if $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
where $c > 0$ and $n_0 > 0$

Problem 4 - Big-O

1. Using the definition of big-O, show $100n + 5 = O(2n)$.

$100n + 5 \leq c \cdot 2n$

$\frac{100n + 5}{2n} \leq c$ Therefore, Big-O holds for $n \geq n_0 = 1$ and

$c \geq \frac{100(1)+5}{2(1)}$ or $c \geq \frac{105}{2}$.

2. Using the definition of big-O, show $n^3 + n^2 + n + 100 = O(n^3)$.

$n^3 + n^2 + n + 100 \leq c \cdot n^3$

$\frac{n^3 + n^2 + n + 100}{n^3} \leq c$ Therefore, Big-O holds for $n \geq n_0 = 1$

and $c \geq \frac{(1)^3 + (1)^2 + 1 + 100}{(1)^3}$ or $c \geq 103$.

3. Using the definition of big-O, show $n^{99} + 10000000 = O(n^{99})$.

$n^{99} + 10000000 \leq c \cdot n^{99}$

$\frac{n^{99} + 10000000}{n^{99}} \leq c$ Therefore, Big-O holds for $n \geq n_0 = 1$

and $c \geq \frac{(1)^{99} + 10000000}{(1)^{99}}$ or $c \geq 10000001$.

Problem 4 - Searching

We will consider the problem of search in ordered and unordered arrays.

1. We are given an algorithm called *search* which can tell us *true* or *false* in one step per search query if we have found our desired element in an unordered array of length 2048. How many steps does it take in the worse possible case to search for a given element in the unordered array?

   2048 steps and the worst case time complexity is O(n).

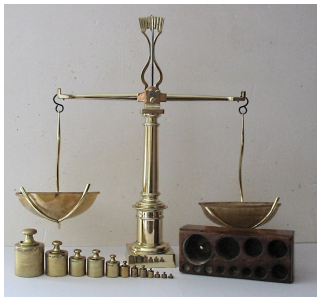
2. Describe a *fasterSearch* algorithm to search for an element in an **ordered array**. In your explanation, include the time complexity using Big-O notation and draw or otherwise explain clearly why this algorithm is able to run faster.

   A binary search is a faster search algorithm. The time complexity is O(log n). This algorithm is faster because it starts at the middle of a list that is already sorted and determines whether the element is in the first or last half of the list based on whether the adjacent elements are greater or less than itself. It then searches from the middle of each subarray until the element is found.

3. How many steps does your fasterSearch algorithm (from the previous part) take to find an element in an ordered array of length 256 in the worse-case? Show the math to support your claim

   $\log_2(256) = 8$ steps

Problem 5 - Another Search Analysis



Imagine it is your lucky day, and you are given 100 golden coins. Unfortunately 99 of the gold coins are fake. The fake gold coins all weigh 1 oz. but the 1 real gold weighs 1.0000000001 oz. You are also given one balancing scale that can precisely weigh each of the two sides. If one side is heavier than the other side, you will see the scale tip.

1. Describe an algorithm for finding the real coin. You must also include the algorithm the time complexity. *Hint* Think carefully–or do this experiment with a roommate and think about how many ways you can prune the maximum amount of fake coins using your scale.

   we can split the coins in half (50/50) and weigh 50 vs. 50. The side that is heavier should contain the real coin. we then split the heavier bunch 25/25 and compare their weights to find the heavier side. We can keep doing this until the coin is found. This algorithm should have a time complexity of O(logn) since it is a binary search.

2. How many weighing must you do to find the real coin given your algorithm?

   $\log_2(100) = 6.644$ so 7 steps worst case.