Homework 4 - Analysis

In this homework, we are going to work to become comfortable with the mathematical notation used in algorithmic analysis.

Problem 1: Quantifiers

For each of the following, write an equivalent *English statement.* Then decide whether those statements are true if x and y are integers (e.g., they can be any integer). Then write a convincing argument to prove your claim.

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

When y = -x, this statement can be proven true.

Proof:

let x be any integer, then y = - x will fulfill x + y = 0 since x – x = 0. Therefore, for any x, there exists a y such that x + y = 0.

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

This statement is True as there does not exist an y that allows x + y = x for any integer x.

Proof:

We can let y = 0, then x + y = x is true for every x since x + 0 = 0 is true for every x.

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

This statement is False as there does not exist an x that can lead every x + y = x.

Proof:
x + y = x is true only for y = 0, but not for any y integers such as 1 or 2. Therefore, there does not exist an x that can fulfill the requirements of the statement.

Name: Chenyi Xiang Homework 4 - Analysis

Problem 2: Growth of Functions

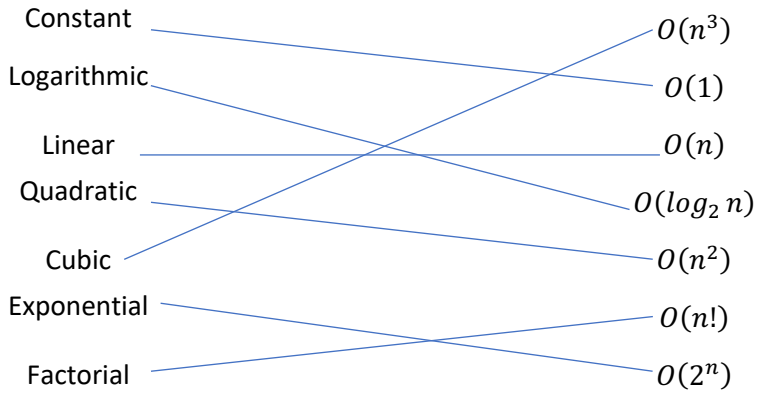
Organize the following functions into six (6) columns. Items in the same column should have the same asymptotic growth rates (they are big-Oh and big-θ of each other. If a column is to the left of another column, all of its growth rates should be slower than those of the column to its right.

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

O(1)	O($n \log n$)	O(N)	O(N^2)	O(2^n)	Exponential
100	$n\ log_3\ n$	$3n$	n^2	2^n	$n!$
10000	$n\ log_2\ n$	$100n$	$5n^2 + 3$		

Problem 3: Function Growth Language

Match the following English explanations to the *best* corresponding big-Oh function by drawing a line from an element in the left column to an element in the right column.



Problem 4: Big-Oh

1. Using the definition of big-Oh, show that $100n + 5 \in O(2n)$

To show that $100n + 5 \in O(2n)$, if there exists a positive integer n0 such that f(n) <= c * g(n) for all n >= n0, then f(n) ∈ O(g(n)). We can set a constant s and n0 such that 100n + 5 <= s* 2n, n >= n0.

Now we can set s = 102 and n0 = 5. Then for any n> n0, the following equation is always true.

100n + 5 <= 102 * 2n

100n + 5 <= 204n

This shows that 100n + 5 grows no faster than 204n, which means that 100n + 5 ∈ O(2n).

2. Using the definition of big-Oh, show that $n^3 + n^2 + n + 42 \in O(n^3)$

We can set a constant c and a positive integer n0 such that $n^3 + n^2 + n + 42 \leq c * (n^3)$ for any n >= n0.

Specifically, we can let c = 2 and n0 = 1. Then for n >= 1. We can conclude:

$$n^3 + n^2 + n + 42 \leq 2 * n^3$$

$n^3 + n^2 + n + 42 <= n^3 + 2* n^3$

then $0 <= n^3$

Since $0 <= n^3$ and for all n >= 1, it obeys that $n^3 + n^2 + n + 42 <= 2 * n^3$, so $n^3 + n^2 + n + 42 \in O(n^3)$

3. Using the definition of big-Oh, show that $n^{42} + 1{,}000{,}000 \in O(n^{42})$

To show that $n^{42} + 1{,}000{,}000 \in O(n^{42})$, we can set a constant c and n0 such that for n >= n0, we can have $n^{42} + 1{,}000{,}000 \leq c * n^{42}$

We can let c = 2 and n0 = 1. Then, for any value of n >= 1, we have $n^{42} + 1{,}000{,}000 \leq 2 * n^{42}$

As n^{42} grows faster than 1000000, therefore, $n^{42} + 1{,}000{,}000 \in O(n^{42})$

Name: Chenyi Xiang

Homework 4 - Analysis

Name: Chenyi Xiang Homework 4 - Analysis

Problem 5: Searching

In this problem, we consider the problem of searching in ordered and unordered arrays:

1. We are given an algorithm called *search* that 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 takes in the worst possible case to search for a given element in the unordered array?

2048. Since in the worst case, this algorithm has to check every element in an unordered array of length 2048 until find the target.

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

Binary search. This algorithm in each search divides evenly between two ordered arrays until find the target number.

Its time complexity is O(log n), in which n is the total number of elements in the array. Since in each step, the size of the array will be reduced into half. Therefore, this leads to a logarithmic relationship between needed steps and number of elements in the array.

As clarified above, the worst situation requires O(n) time complexity which runs to check every element in the worst situation.

3. How many steps does your *fasterSearch* algorithm (from the previous part) take to find an element in an ordered array of length 2,097,152 in the worst case? Show the math to support your claim.

log2(2,097,152) = 21 steps. Since binary search in each search divides evenly between two ordered arrays until find the target number.

And its time complexity is O(log n), in which n is the total number of elements in the array. Since in each step, the size of the array will be reduced into half. Therefore, this leads to a logarithmic relationship between needed steps and number of elements in the array.

Problem 6: 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 weight 1 oz. but the real gold weighs 1.0000001 oz. You are also given one balancing scale that can precisely weight each of the two sides. If one side is heavier than the other the other side, you will see the scale tip.

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

1. Divide the 100 coins into two groups of 50 coins.
2. Weigh the first group against the second group.
3. If the scale is balanced, it means the real coin is in the second group of 50 coins.
4. If the scale is not balanced, it means the real coin is in the first group of 50 coins.
5. Repeat the above steps until two coins left and weigh them against each other to determine which one is the real coin.

The time complexity is O(log n), in which n is the number of coins. In each iteration of the algorithm, I divide the number of coins in half until I find the target. Hence, the algorithm will take log base 2 of n iterations to find the real coin.

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

log(100) = 7

Name: Chenyi Xiang Homework 4 - Analysis

Problem 7 – Insertion Sort

1. Explain what you think the worst case, big-Oh complexity and the best-case, big-Oh complexity of insertion sort is. Why do you think that?

2. Do you think that you could have gotten a better big-Oh complexity if you had been able to use additional storage (i.e., your implementation was not *in-place*)?