Homework 7 - 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. $\forall x\, \exists y : x + y = 0$

True. This means for any x, there is a y that x plus y will equal to 0. Y could be -x or 0 if x is 0.

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

True. This means there is a y that any integer of x + y will equal to x. y could be 0 that the any integer plus 0 will be itself.

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

False. This means: there is a x integer that any y plus x will be x. This is impossible because y has to be 0 to meet this equation x + y = x. Y could not be any integer.

Name: Yiyi Wang Homework 7 - Analysis

Problem 2: Growth of Functions

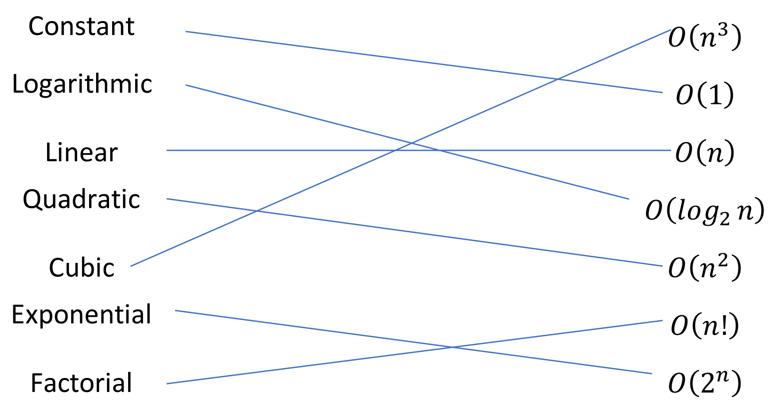
Organize the following functions into six (6) columns. Items in the same column should have the same asymptotic growth rates (they are big-Oh and big-θ of each other. If a column is to the left of another column, all of its growth rates should be slower than those of the column to its right.

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

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

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 $100n + 5 \in O(2n)$

100n + 5 = 50(2n)+5. When the n is large enough, the constant 5 will be ignore. The equation could be 50(2n). And if n is even larger, the 50 is constant and could be ignore. Therefore, the growth rate of 100n+5 will be O(2n).

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

The constant 42 could be ignore since the growth rate is 0. The equation will be $n^3 + n^2 + n$ and when n is getting larger, the growth rate of $n^3$is much larger than $n^2$and n. when n is close to infinite, the growth rate will close to n^3. Therefore, the big-Oh for it is $O(n^3)$

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

When n is two, 2^42 is larger than $1{,}000{,}000$. Constant could be ignored.

Name: Yiyi Wang Homework 7 - 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?

The worst case will be we search every element in the array so we will need to take 2048 steps.

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.

Use the binary search in the ordered array. Take the middle number of the array, if the search element is larger than the middle number, that means the search element is in the right half of the array. We will repeat the steps to take middle number to reduce the range of search until we find the element.

This algorithm is faster because every time we will eliminate half of the searching items.

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.

In the wrost case, we will take 21 times to find the elements. We keep divide the length by 2, we will get the result in 21st time. $\text{Log}_2 2097152$ is 21.

Name: Yiyi Wang Homework 7 - Analysis

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.

Put 50 coins in each side, the heavier side will have the real coin. And then put 25 coins of the heavier 50 to two sides, the heavier side will have the real coins. Divide the coins to 12 and 12 for each side and put one coin away. If the two side have same weight, the 1 coin that is not on the scale is the real one. If one side is heavier, divide these 12 coins to 6 and weigh them. The heavier side will have the real coin, and keep doing it until we find the real one.

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

We need to weigh 6 times to find the real coin. $\text{Log}_2 100 = 6.6$. In the last time of weighting, we will have 3 coins to weighed and we only need to weigh once to know which one is real because the fake coins will have the same weights.

Name: Yiyi Wang Homework 7 - 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?

The time complexity of worst case is O(n^2) because we will need to swap all the elements in the array. In the best case, the array is already sort so the time complexity is O(n).

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*)?

We could use merge sort to have faster time complexity. This sorting method will keep dividing the array into two parts and sort the two halves and merge them together. The space complexity will be O(n) but the time complexity could be O(nlogN).