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

   English Statement

   For all x ,there exists some y making $\mathrm{x} + \mathrm{y} = 0$ true.

   Decide

   This statement is True.

   Argument

   Every integer has its negative number. There always has a $\mathrm{y} = -x$

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

   English Statement

   There exists a value of y making $x + y = x$ always true whatever the value of x .

   Decide

   This statement is True.

   Argument

   While y = 0, x could be any integer $0 + x = x$.

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

   English Statement

   There is a x making $x + y = x$ always true whatever the value of y.

   Decide

   This statement is False.

   Argument

   The statement $x + y = x$ equals to $y = x - x = 0$, which means y can not be any integer. This statement can be true only when y is zero.

Name: Cuiting Huang Homework 7 - Analysis

Problem 2: Growth of Functions

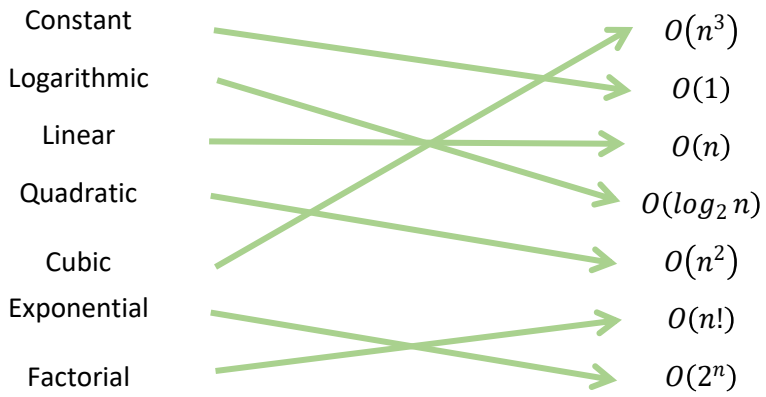
Organize the following functions into six (6) columns. Items in the same column should have the same asymptotic growth rates (they are big-Oh and big-θ of each other. If a column is to the left of another column, all of its growth rates should be slower than those of the column to its right.

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

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

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

Formal Definition

$f(n)$ is $O(g(n))$ if there exists constants 'c' and 'k' such that $|f(n)| \leq c\,|g(n)|$ wherever $n > k$

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

   Show

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

   $100n + 5 \; = \leq 50 * 2\,n + 5 * 2n, \; n > 1;$

   $\Rightarrow 100n + 5 \leq 55 * 2n, n > 1;$

   While $n \; = \; 2, (100 * 2 + 5 \; \leq 105 * 2)$ is true

   $c = 55, k \; = 1$

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

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

   Show

   $n^3 + n^2 + n + 42 \in O(n^3)$

   $n^3 + n^2 + n + 42 \; \leq n^3 + n^3 + n^3 + 42n^3, \; n > 1$

   $\Rightarrow n^3 + n^2 + n + 42 \leq 45n^3$

   While $n = 2, (8 + 4 + 2 + 42 \; \leq \; 45 * \; 8)$ is true

   $c = 45, k \; = 1$

   $n^3 + n^2 + n + 42 \in O(n^3)$ is true.

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

   Show

   $n^{42} + 1{,}000{,}000 \in O(n^{42})$

   $n^{42} + 1{,}000{,}000 \leq n^{42} + 1{,}000{,}000n^{42}, \; n > 1$

   $\Rightarrow n^{42} + 1{,}000{,}000 \leq 1{,}000{,}001n^{42}$

   While $n = 2, (2^{42} + 1{,}000{,}000 \leq 1{,}000{,}001 * 2^{42})$ is true

   $c = 1000001, k \; = 1$

   $n^{42} + 1{,}000{,}000 \in O(n^{42})$ is true.

Name: Cuiting Huang 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?

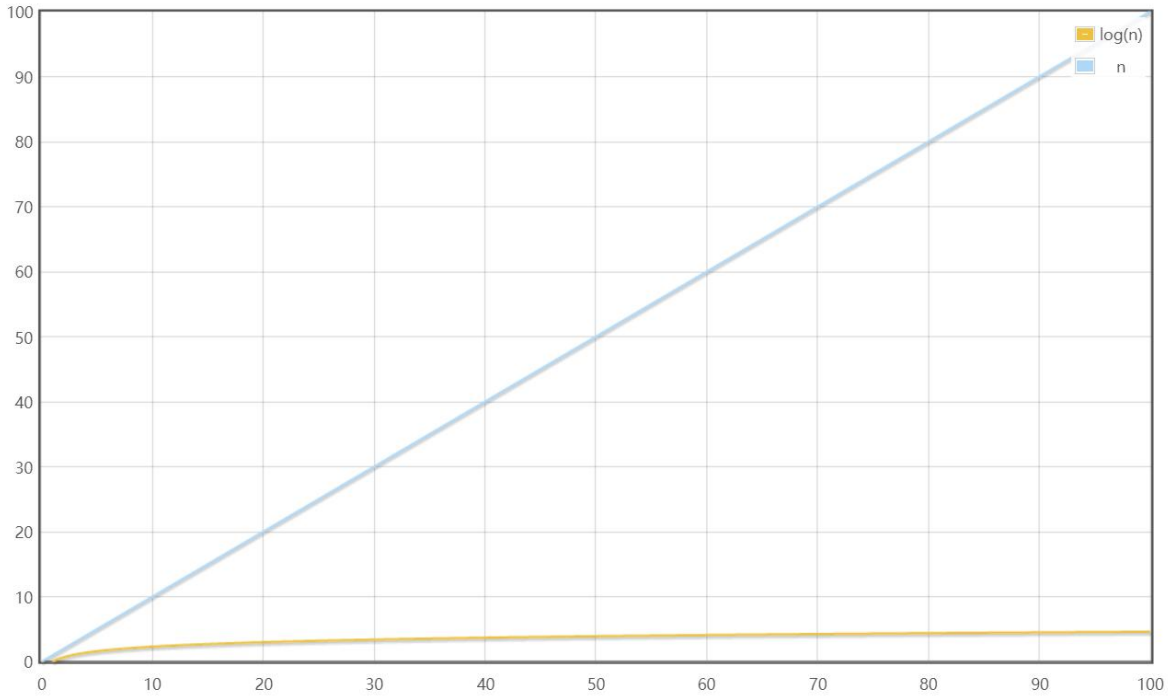
   In the worst possible case, it takes 2048 times to search for a given element in the unordered array.

   We use the *simple search(linear search)*, for each time we get a wrong element, and we just traverse all elements.

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.

   In an ordered array, we will use binary search. The big-Oh is O(logn).
   Step one, we will pick up the element at the middle of the array, named A.
   Step two,comparing the given element named N with A, we will get a information of N,bigger of smaller than A.
   Step three, we will pick another element in the middle of the remaining set of elements.
   Then we will repeat Step three until we find the given elements.

   The big-Oh of *linear search* is O(n), The big-Oh of *binary search* is O(logn). It is obvious that binary search in an ordered array run more faster.



Name: Cuiting Huang Homework 7 - Analysis

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.

   The total of elements is 2097152, in the worst case we have to check log(2097152) elements.

   $2^{10} = 1024$, 2^{20}=1048576, 2^{21} = 2097152
   log(2097152) = 21.

   So it takes 21 steps when we use the faster Search algorithm.

Name: Cuiting Huang 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.

   Step one, divide the coins into two parts, each part has 50 coins and put them on the balancing scale, take the heavier side and go next step.

   Step two, divide the heavier side it into two parts again, each part has 25 coins, take the heavier side and go next step.

   Step three, put 1 coin aside and divide the rest 24 coins in to two parts, each part have 12 coins. If both sides of the scale have the same weight. The coin we put aside is the golden coin. Otherwise, take the heavier side and go next step.

   Step four, repeat the step two and each part has 6 coins.

   Step five, repeat the step two and each part has 3 coins.

   Step six, now we've got 3 coins in total. Put a coin on each side of the balancing scale, if both sides of the scale have the same weight, the remaining coin is the golden one. Other wise, the coin which is heavier is the golden coin.

   The algorithm's time complexity is $O(logn)$.

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

   In the worst case it takes 6 times to find the real golden coin.

   And in the best case it takes 3 times.

Name: Cuiting Huang 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?

   Best-case: We have a sequential array that all the elements are already sorted. And the big-Oh complexity of this case is $O(n)$.

   For every elements have already been sorted, we will never walk through the while loop, We only need to compare $(n - 1)$ times to confirm that all data have been sorted. So the big-Oh complexity $O(n)$.

   Worst-case: We have a reversed array that every single element needs to be sorted. And the big-Oh complexity of this case is $O(n^2)$.

   we could only know which element is the smallest after the iteration of the end of the unsorted portion, we must also iterate until the end.

   For one element, we conduct function find Minimum for one time, which takes n steps. For every elements, it needs n*n steps.

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*)?

   I do not thinks so.
   The big-Oh complexity refers to the amount of computational effort required to perform the algorithm, it has no relation with the storage we could use.

   Even if we could be able to use additional storage, which means we could put every element sorted by every loop time in new space. We still need to iterate the array, the big-Oh complexity will not be changed.