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

   Statement: For every integer x, there exists an integer y such that x + y = 0.

   1. Let x be an arbitrary integer.
   2. Suppose an integer y such that x + y = 0.
   3. For this, we choose y = -x.
   4. Now, we add x and y, we get x + (-x) = 0 (which is true by the definition of additive inverses.)
   5. Since we have found an integer y (y = -x) that makes the equation true for any integer x, we have proven that the statement is true.

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

   Statement: There exists an integer y such that for every integer x, x + y = x

   1. First, we will find integer y.
   2. For any integer x, we have x + y = x.
   3. Subtracting x from both sides of the equation: x + y - x = x - x.
   4. This simplifies to y = 0, *because any integer subtracted from itself equals 0.*
   5. So, we have found an integer y (y = 0), such that when added to any integer x, results in x.
   6. This shows that there exists an integer y (in this case, y = 0) that satisfies the statement.

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

   Statement: There exists an integer x such that for every integer y, x + y = x

   1. Let's assume there exists an integer x such that for every integer y, x + y = x.
   2. Now, we will find an integer x.

3. For any integer y, we have x + y = x.
4. Subtracting x from both sides of the equation: x + y - x = x - x.
5. This simplifies to y = 0, *because any integer subtracted from itself equals 0.*
6. So, for this choice of x, we can see that for any integer y, adding y to x results in x.
7. This tells that there exists an integer x (in this case, x = 0) that satisfies the statement.

Problem 2: Growth of Functions

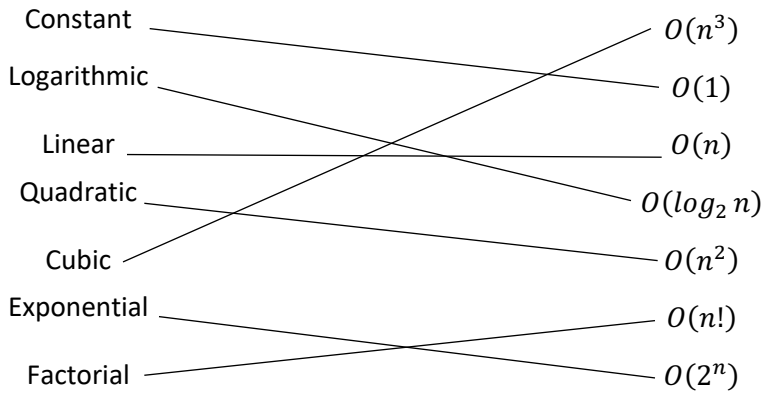
Organize the following functions into six (6) columns. Items in the same column should have the same asymptotic growth rates (they are big-Oh and big-θ of each other. If a column is to the left of another column, all of its growth rates should be slower than those of the column to its right.

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

Slowest Rate	**->**	**->**	**->**	**->**	**Fastest Rate**
10000	$n\ log_3\ n$	n^2	$3n$	$n!$	2^n
100	$n\ log_2\ n$	$5n^2 + 3$	$100n$		

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.



Name: Kevin Chen Homework 4 - Analysis

Problem 4: Big-Oh

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

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

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

   Now, let's choose c = $100c$ (a positive constant greater than 100) and $n_0=1$.

   For $n \geq 1$, the inequality becomes:

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

   We can now prove this inequality for $n \geq 1$:

   Base Case (n=1):

   $100 \cdot 1 + 5 = 105 \leq 100 \cdot 2 \cdot 1 = 200$ (True)

   Inductive Step: Assume the inequality holds for some $k \geq 1$, i.e.,

   $100k + 5 \leq 100 \cdot 2k$

   We need to show that it also holds for k+1:

   $100(k + 1) + 5 \leq 100 \cdot 2(k + 1)$

   Let's prove it: $100(k + 1) + 5 = 100k + 100 + 5$

   $=(100k + 5) + 100 \leq 100 \cdot 2k + 100$ (by the inductive assumption)

   $< 100 \cdot 2k + 100 \cdot 2 = 100 \cdot 2 (k + 1)$

   So, for all $n \geq 1$, the inequality $100n + 5 \leq 100 \cdot 2n$ holds, which means $100n + 5$ is indeed in $O(2n)$ with c=100 and n=1.

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

   $|n^3 + n^2 + n + 42| \leq c \cdot n^3$

   $|n^3 + n^2 + n + 42| = n^3 + n^2 + n + 42$ (Since $n^3 + n^2 + n + 42$ is always non-negative for positive n)

   $|n^3 + n^2 + n + 42| \leq c \cdot n^3$

   Let's choose c = 45 (greater than 42) and $n_0 = 1$
   For $n \geq 1$, the inequality becomes:
   $n^3 + n^2 + n + 42 \leq 45 \cdot n^3 + n^2 + n + 42 \leq 45 \cdot n^3$

Base Case (*n*=1):

$13 + 12 + 1 + 42 = 45 \leq 45 \cdot 1^3 = 45$ (True)

Inductive Step:

Assume the inequality holds for some $k \geq 1$, i.e.:

$K^3 + k^2 + k + 42 \leq 45 \cdot k^3$

We need to show that it also holds for $k+1$:

$(k+1)^3 + (k+1)^2 + (k+1) + 42 \leq 45 \cdot (k+1)^3$

Let's prove it: $(k+1)^3 + (k+1)^2 + (k+1) + 42$

$= k^3 + 3k^2 + 3k + 1 + k^2 + 2k + 1 + k + 1 + 42$

$= (k^3 + k^2 + k + 42) + 3k^2 + 3k + 45$ (by the inductive assumption)

$\leq 45 \cdot k^3 + 3k^2 + 3k + 45$

$< 45 \cdot k^3 + 45 \cdot k^3 = 90 \cdot k^3 = 45 \cdot (k+1)^3$

So, for all $n \geq 1$, the inequality $|n^3 + n^2 + n + 42| \leq c \cdot n^3$ holds, which means $n^3 + n^2 + n + 42$ is indeed in $O(n^3)$ with $c=45$ and $n_0 = 1$

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

$|n^{42} + 1{,}000{,}000| = n^{42} + 1{,}000{,}000$

Simplifying left hand-side

$|n^{42} + 1{,}000{,}000| \leq c \cdot n^{42}$

Now, let's choose $c=1{,}000{,}001$ (a positive constant greater than 1,000,000) and $n_0=1$.

For $n \geq 1$, the inequality becomes:

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

Base Case ($n=1$):

$1^{42} + 1{,}000{,}000$

$= 1 + 1{,}000{,}000$

$= 1{,}000{,}001 \leq 1{,}000{,}001 \cdot 1^{42} = 1{,}000{,}001$ (True)

Inductive Step:

Assume the inequality holds for some $k \geq 1$, i.e.,

$k^{42} + 1{,}000{,}000 \leq 1{,}000{,}001 \cdot k^{42}$.

We need to show that it also holds for $k+1$:

$(k+1)^{42} + 1{,}000{,}000 \leq 1{,}000{,}001 \cdot (k+1)^{42}$

Let's prove it:

$(k+1)^{42} + 1{,}000{,}000 = k^{42} + (\text{other terms}) + 1{,}000{,}000$

$=(k^{42} + 1{,}000{,}000) + (\text{other terms})$

$\leq 1{,}000{,}001 \cdot k^{42} + (\text{other terms})$ (by the inductive assumption)

$\leq 1{,}000{,}001 \cdot k^{42} + (\text{positive terms})$

$\leq 1{,}000{,}001 \cdot k^{42} + 1{,}000{,}001 \cdot k^{42} = 2 \cdot 1{,}000{,}001 \cdot k^{42} = 1{,}000{,}001 \cdot (2k^{42})$

$= 1{,}000{,}001 \cdot (k+1)^{42}$

So, for all $n \geq 1$, the inequality $n^{42} + 1{,}000{,}000 \leq 1{,}000{,}001 \cdot n^{42}$ holds, which means $n^{42} + 1{,}000{,}000$ is indeed in $O(n^{42})$ with $c=1{,}000{,}001$ and $n_0 = 1$.

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 scenario would be if the item is present at las index. Since this is an unordered array, and it takes one step per search to return the output; **it would take 2048 steps to search an item in worse possible scenario.**

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.

   A fasterSearch algorithm to search in sorted array could be as follows:

   1. Start with a sorted array.
   2. Compare the middle element of array with your target element. If target element is equal to middle element, then return true and end the search.
   3. Else If target element is greater than middle element then call the same search function on array to the right side of the middle element and repeat from step 2.
   4. Else If target element is less than middle element then call the same search function on array to the left side of the middle element and repeat from step 2.
   5. Repeat steps 2-4 until array of 1 element is left, if that one element is not the target element then return false.
   6. This algorithm would run with time complexity of O(logn).

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.

   It would take 21 steps for worst-case step of our algorithm fasterSearch.

   Since the Big-Oh complexity of algorithm is O(logn) we can calculate it as follows:

   Log(2,097,152) ≈ 21

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 equal groups of 50 each.
   2. Weigh the two groups against each other using the balancing scale:
      - If one side is heavier, it contains the real gold coin. In this case, you can proceed with the heavier group.
      - If the scale balances, it means the real gold coin is in the other group.
   3. Now, you have narrowed it down to 50 possibilities. Repeat the process:
      - Divide the group with the real coin (as determined in step 2) into two equal groups of 25 each.
      - Weigh these two groups using the balancing scale. The heavier group contains the real coin.
   4. Continue this process by repeatedly dividing the group containing the real gold coin in half and using the scale to identify the heavier side.
   5. Repeat this process until you are left with only one coin, which will be the real gold coin.

   The complexity of this algorithm is O(logn).

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

   It would take O(logn) time complexity where n = 100.

   Which indicates that log(100) = 6.64.

   It would take approximately 7 steps to find the required coin.

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 worst-case complexity of Insertion Sort is **$O(n^2)$.** This is when the array is sorted in reverse order. On each iteration it has to traverse the whole array again and sort it. Meanwhile, the best-case time complexity of insertion sort is **O(n).** This is when array is already sorted, but it still needs to iterate the outer loop on whole array to complete the algorithm.

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*)?

   Yes, we can get better complexity if we use additional storage. In current sort we are moving the elements in place which leads to worst case scenario like $O(n^2)$. We can use additional storage due to which our complexity can be improved. We can use binary search to find the exact position of the element to be inserted, and then insert the element in the additional storage array. This will improve the time complexity to O(nlogn).