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

   For all integer x, there exists an integer y such that x + y = 0.

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

   For all integer x, there exists a y such that x plus y equals x.

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

   For all integer y, there exists an x such that x + y = x.

Name: Po-Shen Lee 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.

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

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
10000, 100	3n, 100n	$n\ log_2\ n$, $n\ log_3\ n$	$5n^2 + 3$, n^2	2^n	$n!$

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.

Constant	$O(n^3)$
Logarithmic	$O(1)$
Linear	$O(n)$
Quadratic	$O(log_2\ n)$
Cubic	$O(n^2)$
Exponential	$O(n!)$
Factorial	$O(2^n)$

Answer:

Constant	$O(1)$
Logarithmic	$O(log_2\ n)$
Linear	$O(n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$
Factorial	$O(n!)$

Name: Po-Shen Lee Homework 4 - Analysis

Problem 4: Big-Oh

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

   By the Big-Oh definition,) f(n) = O(g(n)) means there exists some constant c such that for all n >= N f(n) ≤ c · g(n).
   Like: 100n + 5 ≤ C * 2n
   We can choose C = 50 and N = 1, then for all n ≥ 1,
   we have: 100n + 5 = 50 * 2n
   So, 100n + 5 ∈ O(2n).

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

   By the Big-Oh definition,) f(n) = O(g(n)) means there exists some constant c such that for all n >= N f(n) ≤ c · g(n).
   Like: n^3 + n^2 + n + 42 <= c * n^3.
   We can choose c = 2 and N = 1. then, for any n >= 1, n^3 + n^2 + n + 42 <= 2 * n^3.
   This satisfies the definition of big-Oh and therefore n^3 + n^2 + n + 42 ∈ O(n^3).

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

   By the Big-Oh definition,) f(n) = O(g(n)) means there exists some constant c such that for all n >= N f(n) ≤ c · g(n).
   In this case, n^42 + 1,000,000 is clearly less than or equal to c * n^42 for all n >= 1 and for c = 1,000,001.
   Therefore, n^42 + 1,000,000 is in O(n^42).

Name: Po-Shen Lee 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?

   In the worst-case scenario, it would take 2048 steps to search for a given element in an unordered array of length 2048 using this algorithm, since every element must be checked to determine if it is the desired one.

   Complexity = 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-Oh notation and draw or otherwise clearly explain why this algorithm is able to run faster.

   I will use the binary search while searching for the value in a sorted array.
   step1. Finding the middle element of the array, and comparing it to the target element.
   step2. If the middle element is greater than the target element, the algorithm continues its search in the left half of the array, otherwise, it continues its search in the right half of the array.
   Step3. This process continues until the target element is found or it becomes clear that the desired element is not in the array.

   The time complexity of the binary search is O(log n)
   It only needs 11 steps while searching a sorted array of length 2048.

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 number of steps taken by the binary search algorithm to find an element in an ordered array of length n in the worst case is given by the logarithmic time complexity, O(log n).

   log2(2097152) = 21

   This means that the binary search algorithm will take 21 steps in the worst case to find an element in an ordered array of length 2,097,152.

Name: Po-Shen Lee Homework 4 - 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.

   A solution to find the real coin using the binary search could be as follows:

   Step 1. Divide the 100 coins into two piles of 50 coins each.
   Step 2. Weigh one pile against the other.
   Step 3. If the two piles weigh the same, then the real coin is in the second pile. Repeat the process on this pile.
   Step 4. If one pile is heavier, then the real coin is in that pile. Repeat the process on this pile.

   This algorithm continues to cut the number of coins in half each time until we have only one coin left. This means that the time complexity of this algorithm is O(log n), where n is the number of coins.

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

   The number of weightings to find the real coin given the described algorithm is log base 2 of 100. Therefore, it must take at least 7 times to find the real coin given your algorithm.

Name: Po-Shen Lee 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?

   The worst-case big-Oh complexity of insertion sort is O(n^2). This is because in the worst-case scenario, each element of the array will need to be shifted to its final position, which would take n-1 operations. This would result in a total of (n-1) + (n-2) + ... + 1 operations, which is equivalent to n(n-1)/2, or O(n^2).

   The best-case big-Oh complexity of insertion sort is O(n). This is because in the best-case scenario, the array is already sorted and no operations are needed to rearrange the elements. In this case, the algorithm will simply iterate through the array and determine that the array is already sorted, resulting in a linear time complexity of 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*)?

   No, adding additional storage would not change the worst-case time complexity of the insertion sort, which is still O(n^2). Because in the worst case, each element needs to be compared to all elements in the sorted portion of the array and inserted in its proper place, leading to a total of n(n-1)/2 comparisons. The use of additional storage would not change the number of comparisons required.