Name: Adrian deCola

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 x, there exists a y such that x plus y equals zero." If x and y are integers this is true. For any integer x, if we choose y to be $-x$, then $x+y=0$.

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

"There exists a y, for all x, such that x plus y equals x." If x and y are integers this is true. If y is zero then for all integers x, $x+y=x$.

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

"There exists an x, for all y, such that x plus y equals x." If x and y are integers, this is not true. Subtracting x from both sides we find $y=0$. However, this must be true for all integers y so this statement is false.

Name: _____ Homework 4 - Analysis

Problem 2: Growth of Functions

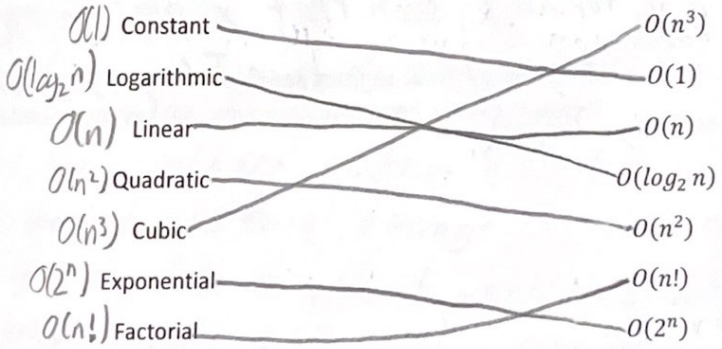
Organize the following functions into six (6) columns. Items in the same column should have the same asymptotic growth rates (they are big-Oh and big-θ of each other. If a column is to the left of another column, all of its growth rates should be slower than those of the column to its right.

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

100	$3n$	$n \log_3(n)$	n^2	2^n	$n!$
1000	$100n$	$n \log_2(n)$	$5n^2+3$		

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.



Name: _____ Homework 4 - Analysis

Problem 4: Big-Oh

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

$100n+5 \in O(2n)$ if there exists constants c and k such that $|100n+5| \leq c(2n)$ $\forall n \geq k$. This is true if $c=200$ and $k=1$. Substituting in these values we get: $|100n+5| \leq 400n \Rightarrow 300n \geq 5$. As this should be true for all $n \geq 1$, we find $300(n) \geq 300 \geq 5 \Rightarrow 300n \geq 5$. As this is true, $|100+5| \leq 200(2n)$ $\forall n \geq 1$. This shows $100n+5 \in O(2n)$.

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

$n^3+n^2+n+42 \in O(n^3)$ if there exists constants c and k such that $|n^3+n^2+n+42| \leq cn^3$ $\forall n \geq k$. This is true if $c=10$ and $k=10$. Substituting in these values we find $|n^3+n^2+n+42| \leq 10n^3 \Rightarrow 9n^3 \geq n^2+n+42 \Rightarrow 9n \geq 1+\frac{1}{n}+\frac{42}{n^2}$. As this must be true for all $n \geq 10$, we know $9n \geq 90$. Also, $\frac{1}{n} < 1$ and $\frac{42}{n^2} < 1$. Using this: $3 \geq 1+1+1 \geq 1+\frac{1}{n}+\frac{42}{n^2}$. Since $9n \geq 90 \geq 3 \geq 1+\frac{1}{n^2}+\frac{42}{n^2}$, we have shown $|n^3+n^2+n+42| \geq 10n^3$ $\forall n \geq 10$. This shows $n^3+n^2+n+42 \in O(n^3)$

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

$n^{42}+1000000 \in O(n^{42})$ if there exists constants c and k such that $|n^{42}+1000000| \leq cn^{42}$ $\forall n \geq k$. This is true for $c=10$ and $k=10$. Substituting these values in: $|n^{42}+1000000| \leq 10n^{42}$. As this must only be true for $n \geq 10$: $10^6 \leq 9n^{42}$. Also, since we must only show this for $n \geq 10$; $n^{42} \geq 10^{42}$. $9n^{42} \geq 9(10^{42}) \geq 10^{42} \geq 10^6$. Therefore $9n^{42} \geq 10^6$ and we have shown $|n^{42}+1000000| \leq 10n^{42}$ $\forall n \geq 10$. This shows $n^{42}+1000000 \in O(n^{42})$

Name: _____ 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 possible case, the element we are searching could be at the end of the array. In that case, it would take 2048 steps if we are not sure if the element is in the array.

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.

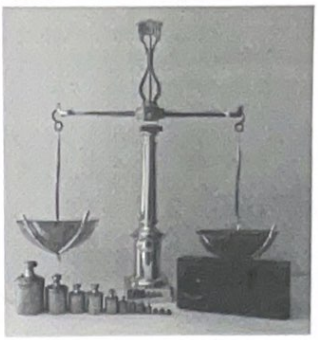
This algorithm could check at the middle of the array and if the element is too small or large we can search at the middle of the half of the list we are now certain it must be in until we find the element. This algorithm is able to run faster because each guess we eliminate half the array (± 1) instead of only one element. Since we eliminate about half the array each time, it takes at most $k+1$ guesses to find the element if the array length is 2^k because $2^k(\frac{1}{2})^{k+1} < 1$ meaning there is no more elements to guess and we have found the element. This means its time complexity is $O(\log_2 n)$.

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.

As $2,097,152 = 2^{21}$, we could eliminate half the list 21 times, and in the worst case, still be left with 1 element, meaning we need 22 steps to find an element in an ordered array of length 2,097,152 in the worst case.

Name: _____ 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.

To find the real coin we could, if there is an even number of coins, put half on each side. Whichever side is heavier must have the real coin so we can repeat on that pile. If the number of coins is odd we can leave one coin out and put half the coins on each plate. If the plates weigh the same, our real coin is the one we left out. If not, the heavier plate contains the coin and we can repeat this process on those coins until we find the real one.

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

In this scenario it would take 6 weightings to find the real coin, in the worst case. The first weighting we would eliminate 50 coins and 25 on the second. In the worst case, on the third weighting the real coin is on one of the plates and we eleminate 13 coins, leaving 12 left. Then on the fourth weighting we eliminate 6 coins, and 3 on the fifth weighting, leaving 3 coins. Then on the sixth weighting we know which coin is the real one.

Name: _____ 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?

In the worst case, the array is originally in descending order and each loop k we have to make k swaps: $1+2+\dots+n-1+n = \frac{n(n+1)}{2}$. $\frac{n(n+1)}{2} \in O(n^2)$. So in the worst case, insertion sort is $O(n^2)$.

In the best case, the array is already sorted and we just check each element with it's left neighbor. Therefore for each loop we do constant time work. We loop through the list, so, in the best case, insertion sort 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*)?

Yes, we could've gotten a better big-Oh complexity if we had additional storage. We could have used an algorithm like merge sort which recursively splits the array in half and calls merge sort on each of those two arrays and then merges them by continously adding the smaller of the two front elements into a new array, requiring space $O(n)$.