Assignment 11, CS5008 Fall2023

Problem 1 - Growth of Functions

Organize the following functions into six columns. Items i n the same column should have the same asymptotic growth rates (they are big-O and big-Θ of each other). If a column i s to the left of another column, all 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

Constant	Linear	Logarithmic	Quadratic	Exponential	Factorial
100, 10000	3n, 100n	$n\log_3 n$, $n\log_2 n$	n^2 , $5n^2 + 3$	2^n	n!

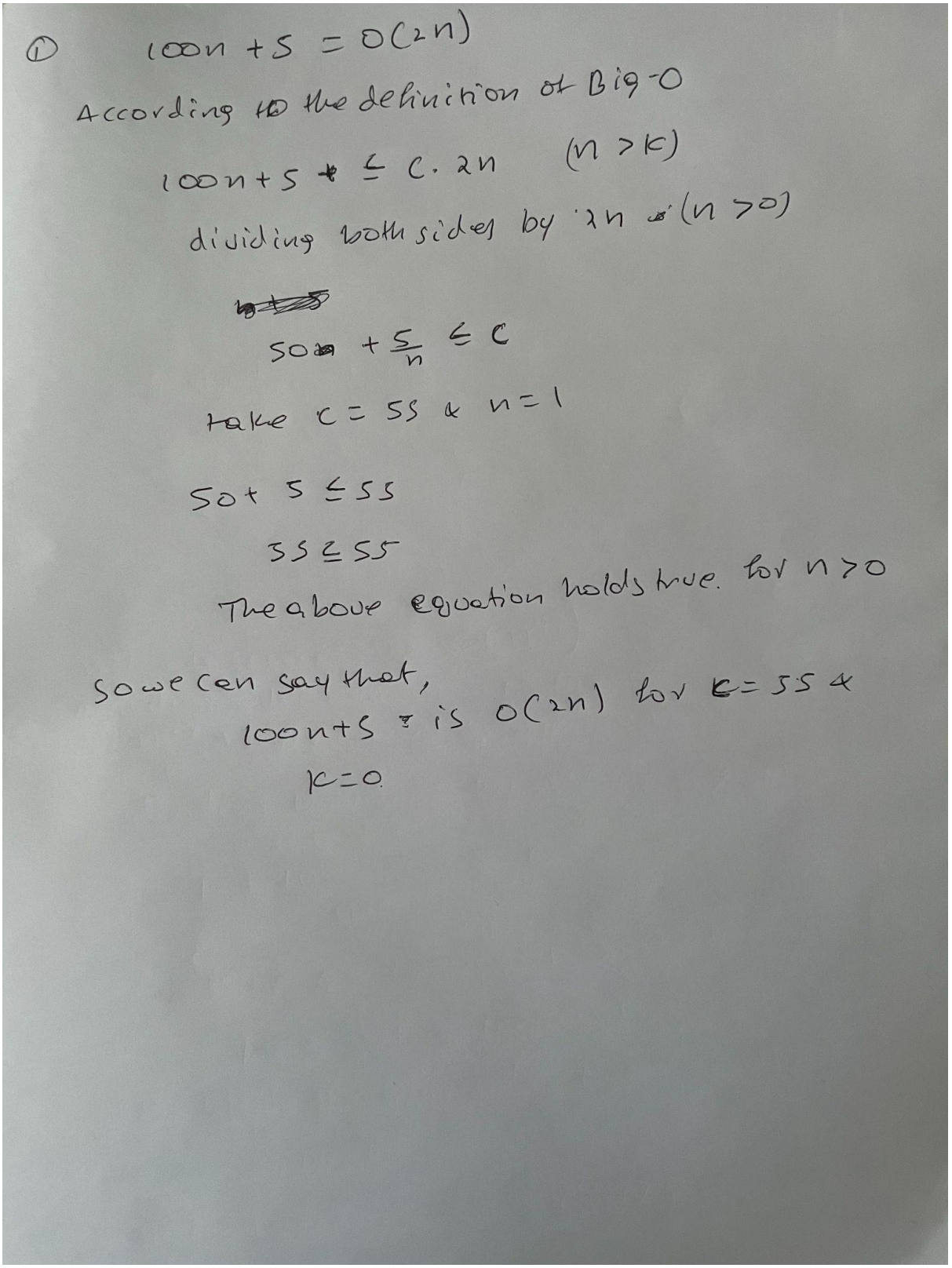
Problem 2 - Function Growth Language

Match the following English explanations to the best corresponding Big-O function by drawing a line from the left to the right.

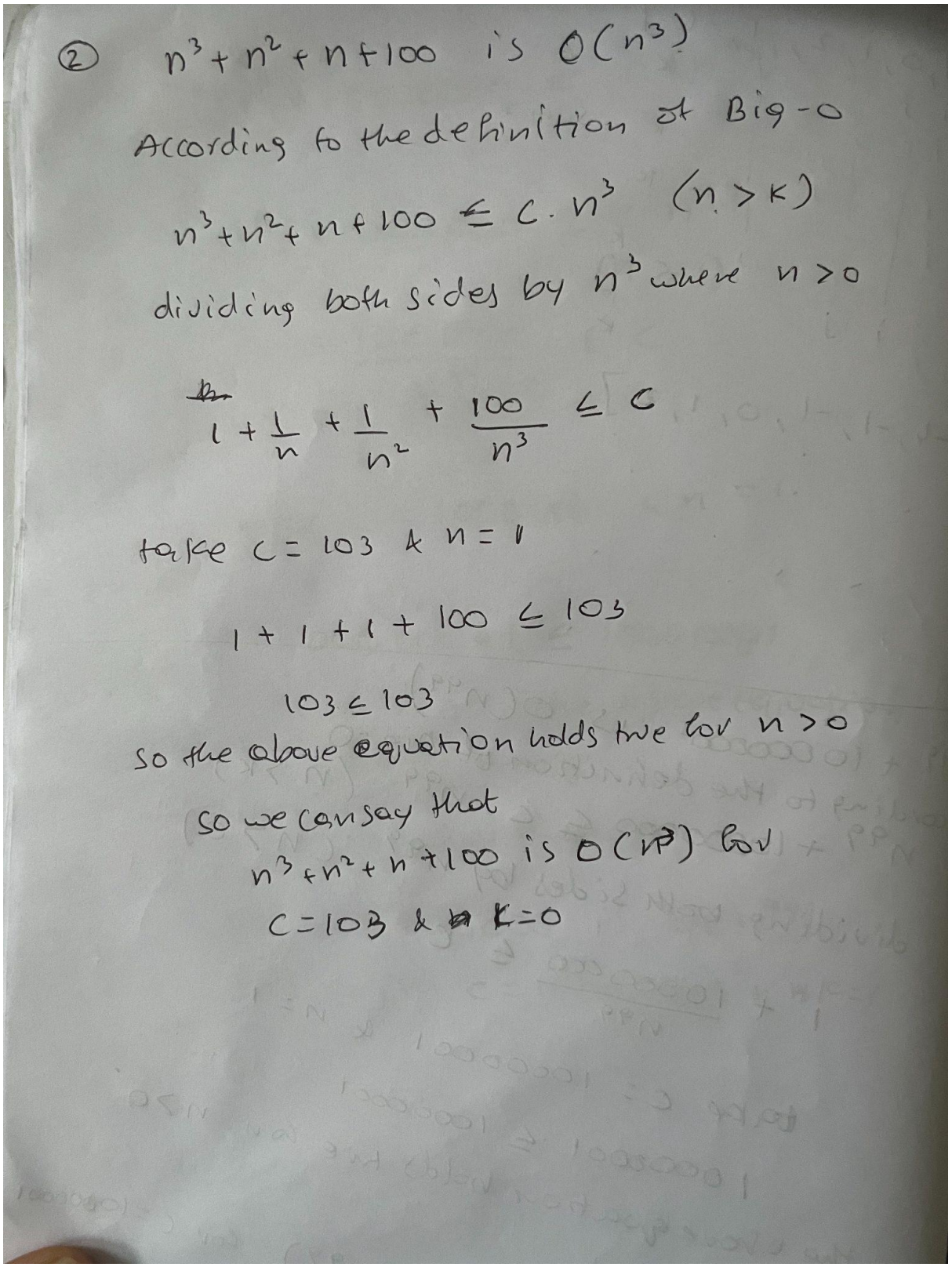
1. Constant time -> O(1)
2. Logarithmic time -> $O(\log_2 n)$
3. Linear time -> O(n)
4. Quadratic time -> $O(n^2)$
5. Cubic time -> $O(n^3)$
6. Exponential time -> $O(2^n)$
7. Factorial time -> O(n!)

Problem 3 - Big-O

1. Using the definition of big-O, show 100n + 5 = O(2n).



2. Using the definition of big-O, show n3 + n2 + n + 100 = O(n3).



② $n^3 + n^2 + n + 100$ is $O(n^3)$

According to the definition of Big-O

$n^3 + n^2 + n + 100 \leq c \cdot n^3 \quad (n > k)$

dividing both sides by n^3 where $n > 0$

$$1 + \frac{1}{n} + \frac{1}{n^2} + \frac{100}{n^3} \leq c$$

take $c = 103$ & $n = 1$

$1 + 1 + 1 + 100 \leq 103$

$103 \leq 103$

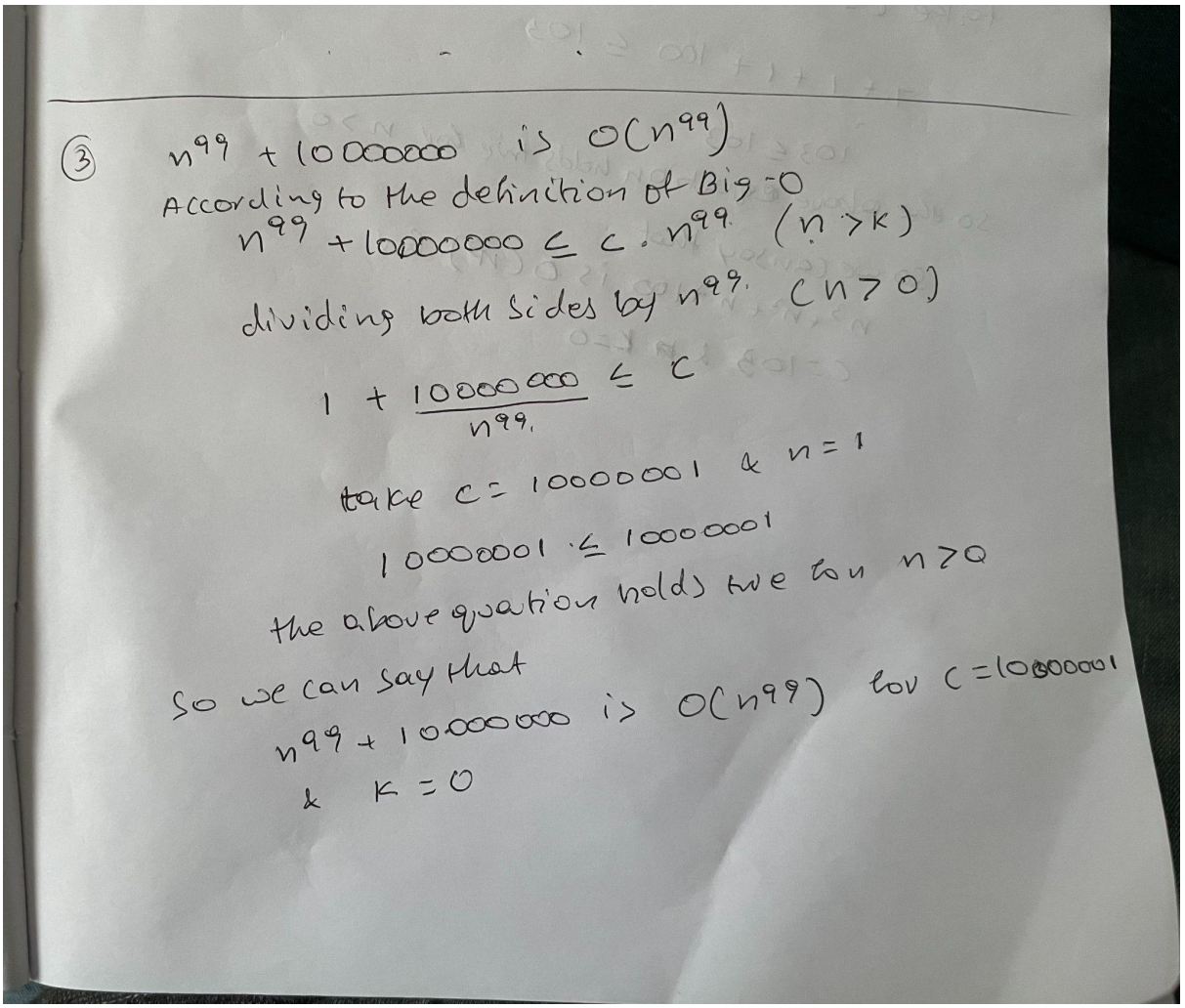
so the above equation holds true for $n > 0$

so we can say that

$n^3 + n^2 + n + 100$ is $O(n^3)$ for

$c = 103$ & $k = 0$

3. Using the definition of big-O, show n99 + 10000000 = O(n99)).



Problem 4 - Searching

We will consider the problem of search in ordered and unordered arrays.

1. We are given an algorithm called search which 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 take in the worse possible case to search for a given element in the unordered array?
-> In the worst-case scenario, the element you are searching for may be in the last position of the array. For an unordered array of length 2048, the worst-case scenario would be that we have to check all 2048 elements to find the element. Therefore, in the worst-case scenario, it would take 2048 steps to search for a given element in the unordered array.

2. Describe a fasterSearch algorithm to search for an element in an ordered array. In your explanation, include the time complexity using Big-O notation and draw or otherwise explain clearly why this algorithm is able to run faster.

-> A faster search algorithm for an ordered array is the binary search algorithm. Binary search takes advantage of the fact that the array is ordered, allowing it to efficiently narrow down the search space by repeatedly dividing it in half.

Lets compare this with the linear search algorithm, take the following array:

nums = [12,23,45,67,88] and our target is 67.

Using linear search it would take us 4 comparisons to find the target i.e

Is 12 = 67 ? no check next

Is 23 = 67 ? no check next

Is 45 = 67 ? no check next

Is 67 = 67 ? yes we found the element.

So in case of linear search the time complexity would be O(n).

But in case of binary search, we know that the array is sorted so the first element is the smallest and last is the largest. Using this knowledge, we take the middle element and see if the middle element is equal to the target. If it's not the target we take the middle element and check if the target is less than or greater than the target. If the target is less than the middle element, we discard the elements greater than the middle element which would be the right half of the array. Similarly if the target is more than the middle element, we discard the elements less than the middle element which would be the left half of the array. After Discarding one of the halves we repeat the same process till the element is found. Lets try this on the array that we used before.

nums = [12,23,45,67,88] and our target is 67.

Mid = 45

Is mid = target? No

Is mid > target ? no so discard the left half

Mid = 67

Is mid = target? Yes we found the element.

So it takes us 2 steps to find the target

The space complexity will be $O(\log_2 n)$

3. How many steps does your fasterSearch algorithm (from the previous part) take to find an element in an ordered array of length 256 in the worse-case? Show the math to support your claim

->In the worst-case scenario, the element you are searching for is either not present in the array or is at the last position.

Binary search divides the array into half in its each step:

Step 1: 256/2

Step 2: 128/2

We continue this till only one element remains, we can express this as:

-> $256 / 2^x = 1$

-> $256 = 2^x$

-> taking log on both sides

-> $\log_2 256 = \log_2 2^x$

-> $x * (1) = \log_2 256$

-> $x = \log_2 2^8$

-> $x = 8$

So, in the worst-case scenario, binary search would take 8 steps to find an element in an ordered array of length 256.