$$\boxed{\lim_{n\to\infty} \frac{f(n)}{g(n)}}$$ $0 \to g(n)$ faster growth
$\infty \to f(n)$ faster growth

Bern Safa
1901042681

1-) a) $\lim_{n\to\infty}\left(\frac{n^2+7n}{n^3+7}\right) \overset{\text{L'hospital (derivation)}}{=} \lim_{n\to\infty}\left(\frac{n^2}{n^3+\cancel{7}}\right) = \lim_{n\to\infty}\left(\frac{1}{n}\right) = 0 \Big\}\ f(n)=O(g(n))$

b) $\lim_{n\to\infty}\left(\frac{12n+\log_2 n^2}{n^2+6n}\right) \overset{''}{=} \lim_{n\to\infty}\left(\frac{12+\cancel{2\cdot\frac{1}{n\ln 2}}^{0}}{\underbrace{2n+6}_{\to\infty}}\right) = 0 \Big\}\ f(n)=O(g(n))$

c) $\lim_{n\to\infty}\left(\frac{n\cdot\log_2 3n}{n+\log_2(5n^3)}\right) \overset{''}{=} \lim\left(\frac{\overbrace{\log_2 3n}^{\infty}+\left(\cancel{n}\cdot\frac{1}{\cancel{n}\ln 2}\right)}{1+\underbrace{\cancel{\frac{3}{n\ln 2}}}_{0}}\right) = \infty \Big\}\ f(n)=\Omega(g(n))$

d) $\lim_{n\to\infty}\left(\frac{n^n+5n}{3\cdot 2^n}\right) \overset{''}{=} \lim_{n\to\infty}\left(\frac{n\cdot n^{n-1}+5}{3\cdot n\cdot 2^{n-1}}\right) = \lim_{n\to\infty}\left(\frac{\overbrace{n\cdot\cancel{(n-1)}\cdot n^{n-2}}^{\infty}}{3\cdot\cancel{(n-1)}\cdot 2^{n-2}}\right) = \infty$

$f(n)=\Omega(g(n))$

e) $\lim_{n\to\infty}\left(\frac{3\sqrt{2n}}{\sqrt{5n}}\right) = \frac{3\sqrt{2}}{\sqrt{5}}\lim_{n\to\infty}\left(\frac{3\sqrt{n}}{\sqrt{n}}\right) = \lim_{n\to\infty}\left(\frac{1}{6\sqrt{n}}\right) = 0 \Big\}\ f(n)=O(g(n))$

2-) a) $O(n)$: n is length of names

b) $O(n^2)$: n is length of myArray, it becomes nested loop

c) It is infinite loop so we can't calculate time complexity.

d) If we assume that all the numbers are smaller than 4, then it will loop until it gives an boundary error. In that case, complexity will be $O(n)$ in the worst-case

3-) First one is going to be "1+1+1+... +n". So it's complexity is O(n).

Second one is also going to O(n) because myArray.length will be n.

So, both of them has same complexity which is O(n).

But, second one is more advantageous than first one because of readabilty, writabilty, maintabilty and flexibilty.

For readability and writabilty, while second one has only 2 line for looping, first one has n line for same job. So, it can be even 1000 or 3000 line while second one is still 2 line. So for loop is more advantageous in readabilty and writabilty.

For maintabilty and flexibilty, in for loop, we can change the number of print with just changing "i < myArray.length" part. But for the first one we have to add or delete every line until reach the wanted value. So, for loop is also more advantageous in maintabilty and flexibilty.

4-) NO, we can't solve it in constant time.

Let's spilit the problem into 2 parts: finding the number, cannot finding the number

1-) Cannot finding: If the specific number is not inside the array, the complexity would be O(n) in any case. Because it will go through every elements in array.

2-) Finding: If we find the specific number in any place which we don't know, the big-O complexity still would be O(n). Let's say we found the number on 3rd index, so we go through in array 3 times. Or, we found it on 1st index and the complexity would be O(1). But, because of we don't know the number place also the array size, it's impossible to say exact number to find the number. Eventually, we would sum up all the probabilty for all indexes. So, the complexity would be O(n) again in worst-case.

As a result, we cannot find a specific number in an unknown array in constant time.

5-) In my algorithm there will be 2 loop seperately. In first loop, it is going to find smallest and the largest value. Let's call them minA and maxA. In the second loop, we are going to make thing for array B. Let's call them minB and maxB.

Then we will multiply all of them with each other, so there will be 4 values. The reason why I didn't calculate just only minimum values is, there will be negative values. So their product will be positive and maybe the product is not going to smallest because of this.

At the end we will compare these 4 values with if-else blocks and find the smallest product.

So, the first for loops complexity is O(n). And second ones is O(m). Other if-else lines just single lines so they won't effect the Big-O notation. When we sum, the complexity will be O(n+m). We can't omit one of them because we don't know the size of arrays. Here it will be "linear time algorithm" with O(n+m).

Pseudo-code:

```
minA = A[0] //a0
maxA = A[0] //a0
minB = B[0] //b0
maxB = B[0] //b0

for i=1 to (n-1)            } O(n)
    if A[i] < minA
        minA = A[i]
    if A[i] > maxA
        maxA = A[i]

for j=1 to (m-1)            } O(m)
    if B[j] < minB
        minB = B[j]
    if B[j] > maxB
        maxB = B[j]

val1 = minA . minB
val2 = minA . maxB
val3 = maxA . minB
val4 = maxA . maxB

min = val1

if val2 < min
    min = val2

if val3 < min
    min = val3

if val4 < min
    min = val4

print(min)
```

Contimue of 5th questien

Just in case, there is an another solution for this problem which is using nested loop and multiply inside of the nested loop. But nested loops complexity would be $O(n \times m)$. So it wouldn't be linear time algorithm.