Ahmet Yiğit
2021020204066

1)

a-) $f(n) = n^2 + 7n \qquad g(n) = n^3 + 7$

- Before we can comment on the solution, we need to take the limit to infinity.

$$\lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right) = \lim_{n \to \infty} \left(\frac{n^2+7n}{n^3+7} \right) \xrightarrow{\text{L'hospital}} \lim_{n \to \infty} \left(\frac{2n+7}{3n^2} \right) = \lim_{n \to \infty} \left(\frac{2}{6n} \right) = \lim_{n \to \infty} \left(\frac{1}{3n} \right) = 0$$

- The final solution approaches to "0".
- So the answer is $f(n) = O(g(n))$

b-) $f(n) = 12n + \log_2(n^2) \qquad g(n) = n^2 + 6n$

$$\lim_{n \to \infty} \left(\frac{12n + \log_2(n^2)}{n^2+6n} \right) \underset{\text{divide } n^2}{=} \lim_{n \to \infty} \left(\frac{\frac{12}{n} + \frac{\log_2(n^2)}{n^2}}{1 + \frac{6}{n}} \right) = \frac{\lim_{n \to \infty} \left(\overbrace{\frac{12}{n}}^{0} + \overbrace{\frac{\log_2(n^2)}{n^2}}^{0} \right)}{\lim_{n \to \infty} \left(\frac{1}{1} + \underbrace{\frac{6}{n}}_{0} \right)} = 0$$

- Answer is $f(n) = O(g(n))$

c-) $f(n) = n \log_2(3n) \qquad g(n) = n + \log_2(8n^3)$

$$\lim_{n \to \infty} \left(\frac{n \cdot \log_2(3n)}{n + \log_2(8n^3)} \right) \underset{\text{divide } n}{=} \lim_{n \to \infty} \left(\frac{\log_2(3n)}{1 + \frac{\log_2(8n^3)}{n}} \right) = \lim_{n \to \infty} \left(\frac{\log_2(3n)}{1+0} \right) = \infty$$

$$\lim_{n \to \infty} \frac{\log_2(8n^3)}{n} = \lim_{n \to \infty} \frac{\frac{24n^2}{8n^3 \ln 2}}{1} = \lim_{n \to \infty} \frac{3}{n \ln 2} = 0$$

- Solution is $f(n) = \Omega(g(n))$

d-) $f(n) = n^n + 5n \qquad g(n) = 3 \cdot 2^n$

$$\lim_{n \to \infty} \left(\frac{n^n + 5n}{3\, 2^n} \right) = \frac{1}{3} \left(\overbrace{\lim_{n \to \infty} \left(\frac{n^n}{2^n} \right)}^{\infty} + \overbrace{\lim_{n \to \infty} \left(\frac{5n}{2^n} \right)}^{0} \right) = \infty$$

- Solution approaches infinity so the answer is $f(n) = \Omega(g(n))$

e-) $f(n) = \sqrt[3]{2n}$ and $g(n) = \sqrt{3n}$

$$\lim_{n \to \infty} \left(\frac{\sqrt[3]{3} \quad n^{\frac{1}{3}}}{\sqrt{3} \quad n^{\frac{1}{2}}} \right) = \frac{1}{\sqrt[6]{3}} \lim_{n \to \infty} \left(\frac{1}{n^{1/6}} \right) = 0$$

- The solution approaches zero so the answer is $\Rightarrow f(n) = O(g(n))$

2) a) worst-case time complexity of the method is O(n).
• Because of code block iterations repeat it itself as long as input size.

b) MethodB uses methodA. MethodA worst-case time comp. is O(n) also methodB calls methodA functions n times, so O(n)*O(n) = O(n²)

c) It is impossible to calculate, because of loop never ends.

d) worst-case is all input is lower than h so the time comp is O(n)

3) There is no difference between two method in terms of time complexities.
• But withLoop function is more efficient because it has a simpler implementation and does the same task in fewer lines of code.
• Also another advantage is when we change the input size, we have to change to the code of withoutLoop function but in withLoop function can handle to print count.

4) No, there is no solution, to find exact number in a constant time. We have to search all array to it includes our target number or not, and it takes O(n) time.

pseudo-code:

```
function find_target(array, target):
    int idx:=0
    while idx < array.length Do
        if array[idx]== target:
            return true
        idx++;
    return false
```

b)

- My solution approach:

1-) Find min1 and max1 values in array A. $O(n)$
2-) Find max2 and min2 values in array B. $O(n)$
3-) Compare min1 * max2 and max1 * min2. $O(1)$
4-) return the smallest.

psuedo code:

```
function find-min-product(A, B, n, m)
        int idx := 0
        int min1 := infinity
        int max1 := -infinity
        int min2 := min1
        int max2 := max1;
        while idx < n Do
            if A[idx] > max1:
                max1 = A[idx]
            if A[idx] < min1:
                min1 = A[idx]
            idx ++;
        idx := 0
        while idx < m Do
            if B[idx] > max2:
                max2 = B[idx]
            if B[idx] < min2:
                min2 = B[idx]
            idx ++
        if min2 * max1 < min1 * max2 Do
            return min2 * max1
        return max2 * min1
```

Time Complexity = $O(n) + O(n) = 2 * (O(n)) = O(n)$

$\boxed{-10}$ -3 -1 0 1 3 $\boxed{7}$ min