Question 1 For each of the function pairs below, show whether $f(n) = O(g(n))$, or $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$ by using the limit approach.

a) $f(n) = n^2 + 7n$ and $g(n) = n^3 + 7$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \frac{n^2+7n}{n^3+7} = \frac{1/n + 7/n^2}{1 + 7/n^3} = \frac{0+0}{1+0} = \frac{0}{1} = 0$$

↓ dividing both numerator and denominator by n^3

n approaches infinity

$f(n) = O(g(n))$

$g(n)$ grows faster than $f(n)$ asymptotically

---

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

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \frac{12n + \log_2 n^2}{n^2+6n} = \frac{12/n + \frac{\log_2 n^2}{n^2}}{1 + 6/n} = \frac{\frac{\log_2 n^2}{n^2}}{1} = \frac{0}{1} = 0$$

↓ dividing both numerator and denominator by n^2

$f(n) = O(g(n))$

$g(n)$ grows faster than $f(n)$ asymptotically

---

c) $f(n) = n \log_2 3n$ and $g(n) = n + \log_2 (8 \cdot n^3)$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \frac{n \log_2 3n}{n + \log_2 (8 \cdot n^3)} = \frac{n \log_2 3 + n \log_2 n}{n + \log_2 8 + \log_2 n^3} = \frac{\log_2 3 + \log_2 n}{1 + \frac{\log_2 8}{n} + \frac{\log_2 n^3}{n}} = \frac{\log_2 3 + \log_2 n}{1} = \infty$$

↓ dividing both numerator and denominator by n

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

$f(n)$ grows faster than $g(n)$ asymptotically

---

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

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \frac{n^n + 5n}{3 \cdot 2^n} = \frac{1}{3} \lim_{N\to\infty} \frac{n^n + 5n}{2^n} = \frac{n^n}{2^n} + \frac{5n}{2^n} = \frac{1}{3}(\infty + 0) = \infty$$

$\frac{n^n}{2^n} \to \infty$, $\frac{5n}{2^n} \to 0$

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

$f(n)$ grows faster than $g(n)$ asymptotically

---

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

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \frac{\sqrt[3]{2n}}{\sqrt{3n}} = \frac{\sqrt[3]{2}}{\sqrt{3}\sqrt[6]{n}} = \frac{\sqrt[3]{2}}{\sqrt{3}} \lim_{N\to\infty} \left(\frac{1}{\sqrt[6]{n}}\right) = \frac{\sqrt[3]{2}}{\sqrt{3}} \cdot \frac{1}{\infty} = 0$$

↓ dividing both numerator and denominator by $\sqrt[3]{n}$

$\lim \to \infty$

$f(n) = O(g(n))$

$g(n)$ grows faster than $f(n)$ asymptotically

Question 2 Analyze the worst-case time complexity of the following methods.

PS: For each method, if there is an array, assume its length as n where $n \in \mathbb{Z}^+$

a)
```
static void methodA(String names[]) {
    for (int i = 0; i < names.length; i++)
        System.out.println(names[i]);
}
```
$\theta(1)$ inside, $\theta(n)$ overall

$T(n) = \theta(1) * \theta(n) = \theta(n)$

the worst-case time is $O(n)$

---

b)
```
static void methodB() {
    String[] myArray = new String[] {"CSE222", "CSE505", "HW2"};
    for (int i = 0; i < myArray.length; i++)
        methodA(myArray);
}
```
$\theta(n)$ inside, $\theta(n)$ overall

$T(n) = \theta(n) * \theta(n) = \theta(n^2)$

the worst-case time is $O(n^2)$

in specific case, length of the myArray is 3 so methodA called 3 times. Each time processing 3 elements - so n is $3*3 = 9$

the worst time taken will be approximately $9^2 = 81$ units of time

c)
```
static void methodC (int numbers[]) {
    int i = 0;
    while (i < numbers.length)
        System.out.println(numbers[i]);
}
```

This is infinite loop. So we cannot calculate any worst-case time complexity

d)
```
static void methodD (int numbers[]) {
    int i = 0;
    while (numbers[i] < 4)
        System.out.println(numbers[i++]);
}
```

In the worst-case if all elements in the array are less then 4, the while loop iterate n times until it reaches the end of the array.

$T(n) = \Theta(n)$ $O(n)$ ← the worst-case time is

Question 3 | What is the difference between the time complexities of the following methods? Which one is more advantageous?

The first method "withoutLoop" has time complexity $T(n) = \underbrace{\Theta(1) + \Theta(1) + \Theta(1) \dots \dots \Theta(1)}_{n \text{ times}}$

$T(n) = \Theta(n)$

The second method "withLoop" also has the complexity $T(n) = \Theta(n)$

If the array has a small size and the number of iterations is fixed, the first method might be faster since it avoids the overhead of the loop. However, if the array is large or the number of iterations is variable the second method would be more efficient. Additionally the second method more flexible.

Question 4 | Consider an array of n integers ($n \in \mathbb{Z}^+$) You don't have any informations at whether the array is sorted or not and you are supposed to check if the array contains a specific integer. Considering all possible inputs, can you solve the problem in constant time? If so, write down the pseudo-code of the algorithm and analyze its time complexity. If not, explain why.

In unsorted array, its not possible to find a specific integer in constant time. Because it is necessary to execute every element of the array. In the best case desired integer located in front of the array, in the worst-case end of the array. Or its not. The best possible time complexity for this problem is $O(n)$

Question 5 | Consider two integer arrays A and B as follows:

$A = [a_0, a_1, \dots, a_{n-1}]$

$B = [b_0, b_1, \dots, b_{m-1}]$

where $n, m \in \mathbb{Z}^+$. Design a linear time algorithm to find the minimum value of $a_i \cdot b_j$ where $0 \le i < n$ and $0 \le j < m$. Explain your algorithm (along with the pseudo code) and analyze its worst-case time complexity.

```
function findMin(A, B) {
    n = length(A)        → n is length of A
    m = length(B)        → m is length of B
    min_val = A[0] * B[0]   ← first min_val is first element of A and B and its multiplicand
    for i = 0 to n-1:       → outer loop iterates A array
        for j = 0 to m-1:   ← inner loop iterates B array
            temp_val = A[i] * B[j]
            if temp_val < min_val
                min_val = temp_val
    return min_val
}
```

$\Theta(1)$ } $\Theta(m)$ } $\Theta(n) = O(n * m)$ ← the worst-case time is

depending on the if condition, the minimum value is renewed for each new value.

If the desired min value is end of the value of A and B arrays, the worst-case complexity will be as above.