1) $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

approach $= \lim_{x \to \infty} \frac{f(x)}{g(x)}$ for all parts

Θ means $\lim_{n \to \infty}$

a) $\Theta \frac{n^2 + 7n}{n^3 + 1} = \Theta \frac{\cancel{n^2}(1 + \cancel{\frac{7}{n}}^{0})}{n^{\cancel{3}}(1 + \cancel{\frac{1}{n^3}}^{0})} = \Theta \frac{1}{n} = \boxed{0}$

when n goes to ∞ limit is 0 therefor

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

b) $\Theta \frac{12n + \log_2 n^2}{n^2 + 6n} = \Theta \frac{\cancel{n^2}\left(\cancel{\frac{12}{n}}^{0} + \frac{\log_2(n^2)}{n^2}\right)}{\cancel{n^2}(1 + \cancel{\frac{6}{n}}^{0})} = \Theta \frac{\log_2(n^2)}{n^2} = \Theta \frac{2\log_2(n)}{n^2} = 2\Theta \frac{\log_2 n}{n^2}$

→ l'hospital → $2\Theta \frac{(\log_2 n)'}{(n^2)'} = 2\Theta \frac{\frac{1}{n \ln(2)}}{2n} = \cancel{\frac{2}{2}} \ln(2)\, \Theta \frac{1}{n^2} = \boxed{0}$

when n goes to ∞ limit is 0 therefore

$f(n) = o(g(n))$

c) $\lim \frac{n \log_2(3n)}{n^2 \log_2(8n^3)} = \lim \frac{\cancel{n \log_2(3n)}}{\cancel{n \log_2(3n)}\left(\frac{1}{\log_2(3n)} + \frac{9}{n\log_2(3n)} + \frac{3\log_2(n)}{\log_2(\log_2(n))}\right)}$

$\frac{3\log_2(n)}{\log_2(n)\left(\frac{1}{\log_2(n)}+1\right)} \to 3$

$= \lim \frac{1}{3} = \boxed{\frac{1}{3}}$

When $n \to \infty$ limit is $\frac{1}{3}$ (positive constant)

therefore $\boxed{f(n) = \Theta(g(n))}$

d) $\lim \frac{n^n + 5n}{3 \cdot 2^n} = 3\lim\left(\frac{n^n}{2^n} + \frac{5n}{2^n}\right) = 3\lim\left(\frac{n^n}{2^n}\right) + 3\lim\left(\frac{5n}{2^n}\right)$

l'hospital

$3\lim \frac{5}{n 2^{n-1}} = 0$

$= 3\lim \frac{n^n}{2^n} + 0 = 3\lim\left(\frac{n}{2}\right)^n = \boxed{\infty}$

for a^n if $a>1$ limit is ∞, $\frac{n}{2} > 1$ ✓

When $n \to \infty$ limit is ∞ therefore

$\boxed{f(n) = \Omega(g(n))}$

e) $\lim \frac{\sqrt[3]{2n}}{\sqrt{3n}} = \lim \frac{(2n)^{\frac{1}{3}}}{(3n)^{\frac{1}{2}}} = \frac{2^{\frac{1}{3}} \cdot 3^{-\frac{1}{2}}}{n^{\frac{1}{2}} \cdot n^{-\frac{1}{3}}} = \frac{0{,}73}{n^{\frac{1}{6}}} = \boxed{0}$

When $n \to \infty$ limit is 0 therefor

$\boxed{f(n) = O(g(n))}$

P3. 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]);
  }
```

n // const C_2

$C_2 \cdot n$ $O(n)$

b)

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

$\rightarrow n$

// $O(n)$ $(C_2 \cdot n)$

$C_2 \cdot n \cdot n \rightarrow O(n^2)$

c)

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

infinite loop

$c_1 \cdot \infty$ ~~$O(\infty)$~~

no time comp.

d)

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

loop will execute n times until end throw exception if no 4

//const c_1

$c_1 \cdot n$ $O(n)$

in worst

3) (20 pts) What is the difference between the time complexities of the following methods?

3) Both of them ~~are~~ have $O(n)$ time complexity and seem's like same but actually the first one only incrementing (c_1) and printing (c_2) → $(c_1+c_2).n$ while the second one also compressing between lenght of array and control variable (c_3) → $(c_1+c_2+c_3).n$ looking at this the first one has better time efficiency but not sutiable for variable lenght arrays

4) No, because in the worst case we have to check all elements of array we cannot know the if specific element is wanted element or not in unsorted array and that makes it $O(n)$ time complexity

5) // This pseudocode written as [illegible]

```
findMinaxb(A,B):

    mina = maxa = A[0]
    minb = maxb = B[0]

    for x in A:        // find min and max
        if x < mina:   // values for A
            mina = x
        else if x > maxa:
            maxa = x

    for y in B:        // find min and max values
        if y < minb:   // for B
            minb = y
        else if y > maxb:
            maxb = y

    min = mina * minb
    if mina*maxb < min:
        min = mina* maxb
    if maxa * minb < min:
        min = maxa * minb
    if maxa* maxb < min:
        min = maxa* maxb

    return min;
```

→ 2

5) This algorithm makes :-

1. Find min and max value for array A

2. Find min and max value for array B

3. do cartesian product between min and max values and find min among them

2 We are doing this because arrays can contains negative values or all of values can be negative that's why we need to check all this production or negative - positive statement