Q1)

a) $\lim_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n} = \frac{n^4-6n^3+9n^2}{5n^3+n} = \frac{\infty}{\infty}$

L'Hospital $\rightarrow \frac{4n^3-18n^2+18n}{15n^2+1} = \frac{\infty}{\infty}$

L'Hospital $\rightarrow \frac{12n^2-36n+18}{30n} \rightarrow$ We can focus on the highest power of n

$\lim_{n\to\infty} \frac{12n^2}{30n} = \lim_{n\to\infty} \frac{12n}{30} = \infty \Rightarrow f(n) = \Omega(g(n))$

---

b) $\lim_{n\to\infty} \frac{n^3}{\log_2 n^4}$ $\log_2 n^4 = 4\log_2 n \rightarrow \lim_{n\to\infty} \frac{n^3}{4\log_2 n}$

L'hospital $\rightarrow \lim_{n\to\infty} \frac{3n^2}{\frac{4}{\ln 2}\cdot\frac{1}{n}} \rightarrow \lim_{n\to\infty} \frac{3n^3}{\frac{4}{\ln 2}} = \frac{\infty}{\frac{4}{\ln 2}} = \infty \Rightarrow f(n) = \Omega(g(n))$

---

c) $\lim_{n\to\infty} \frac{5n\cdot\log_2(4n)}{n\cdot\log_2(5^n)}$ $\log_2 4n = 2+\log_2 n$, $\log_2(5^n) = n\cdot\log_2 5$

$\lim_{n\to\infty} \frac{5n(2+\log_2 n)}{n\cdot(n\cdot\log_2 5)} = \lim_{n\to\infty} \frac{10n+5n\cdot\log_2 n}{n^2\cdot\log_2 5}$

L'hospital $\rightarrow \frac{10+5\log_2 n}{2n\cdot\log_2 5}$

L'hospital $\rightarrow \lim_{n\to\infty} \frac{\frac{5}{n\ln 2}}{2\log_2 5} = \frac{0}{2\log_2 5} = 0 \Rightarrow f(n) = O(g(n))$

---

d) $\lim_{n\to\infty} \frac{n^n}{10^n} = \lim_{n\to\infty} \left(\frac{n}{10}\right)^n = \infty^\infty = \infty \Rightarrow f(n) = \Omega(g(n))$

---

e) $\lim_{n\to\infty} \frac{8n\sqrt[5]{2n}}{n\cdot\sqrt[3]{n}} = \lim_{n\to\infty} \frac{8\sqrt[5]{2n}}{\sqrt[3]{n}}$

L'hospital $\rightarrow \lim_{n\to\infty} \frac{\frac{8}{5}(2x)^{-4/5}}{\frac{1}{3}x^{-2/3}} = \frac{24}{5}\lim_{n\to\infty} \frac{1}{2^{4/5}\cdot n^{2/15}} = \frac{1}{\infty} = 0$

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

Q2)

a) static void methodA(String str_array[]){
for(int i=0; i<str_array.length; i++) →
str_array[i] = "0"; ⟶ takes constant time → O(1)

↳ The loop iterates n times, n is the length of 'str_array' so the time complexity of the loop is O(n)

↳ Each operation in the loop is constant time O(1) and loop iterates n time O(n) ⇒ time complexity = O(n) × O(1)
= O(n)

---

b) First loop:

↳ loop iterates n times (length of 'str_array')

↳ Each iteration, calls method A which has a time complexity of O(n)

↳ So, time complexity is O(n) × O(n) = O(n²)

Second Loop:

↳ Loop iterates n times

↳ printing each element is constant time O(1)

↳ So, time complexity is O(n) × O(1) = O(1)

First loop dominates the complexity due to its quadratic or dominant one, ⇒ O(n²) is the time complexity

---

c) Outer loop iterates n times

Inner loop also iterates n times for each iteration methodB called. We've already calculated time complexity of methodB is O(n²).

So, totally n² times methodB is calling.

⇓

$$O(n^2) \times O(n^2) = O(n^4)$$

---

d) Loop variable (i) decrementing after setting array (i--), and loop increments it (i++), So, i will not change and method will enter an infinite loop. Therefore, time-complexity cannot be calculated due to it is infinite behaviour.

e) On the worst case, there was no any empty stores in the array, or may be the last one (element) on the array.
In this case, loop iterates every element and the worst case time complexity is O(n) n is the length of the array.
And in every loop, there is O(1) operation (if an element is an empty str) so, the worst-case time complexity is $O(n) \times O(1) = O(n)$

Q3)

a) Sorted Version:

```
Difference Sorted Array (A):
    l = length(A)
    if (l < 2) then
        Return "Array should at least 2 elements"
    Return A[l-1] - A[0]   // since its sorted last - first
```

b) Not Sorted Version:

```
Difference Unsorted Array (A):
    l = length(A)
    if l < 2 then
        Return "Array should at least 2 elements"
    smallest = A[0]
    difference = A[1] - A[0]
    for i = 1 to l-1 do
        If A[i] - smallest > difference then
            difference = A[i] - smallest
        If A[i] < smallest then
            smallest = A[i]
    Return difference
```

Time complexities (WorstCase):

Sorted Version → Code performs constant operations regardless of array size, so, its O(1)

Not Sorted Version → Code scans (iterates) array once and does constant operations with each element. so its O(n) n = length of array.