GENERAL

Part 1

a) $f(n)=(n^2-3n)^2$ and $g(n)=5n^3+n$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\frac{(n^2-3n)^2}{5n^3+n}=\frac{\infty}{\infty}\xrightarrow{L'hospital}\frac{2(n^2-3n).(2n-3)}{15n^2+1}=\frac{4n^3-\dots n^2-}{15n^2+1}$$

$$\xrightarrow{L'hospital}\frac{12n^2-36n+18}{30n}\xrightarrow{L'hospital}\frac{24n-36}{30}=\infty$$

$$\frac{\infty}{\infty}$$

Therefore $f(n)\in\Omega(g(n))$

b) $f(n)=n^3$ and $g(n)=\log_2 n^4$

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

$$\xrightarrow{L'hospital}\frac{3n^2}{\frac{4n^3}{n^4\ln 2}}=\frac{3n^3\ln 2}{4}=\infty$$

Therefore $f(n)\in\Omega(g(n))$

c) $f(n)=5n.\log_2(4n)$

$g(n)=n.\log_2 5^n$

$$\lim_{n\to\infty}\frac{5n.\log_2 4n}{n.\log_2 5^n}\Rightarrow\frac{5n.\log_2 2^{4n}}{n.\log_2 5}=\frac{\infty}{\infty}$$

$$\xrightarrow{L'hospital}\frac{5.4}{\frac{4n.\ln 2}{\log_2 5}}\Rightarrow\frac{5}{n.\ln 5}\Rightarrow 0$$

Therefore $f(n)\in O(g(n))$

d) $f(n)=n^n$ and $g(n)=10^n$

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

Therefore $f(n)\in\Omega(g(n))$

e) $f(n)=8n.\sqrt[5]{2n}$

$g(n)=n.\sqrt[3]{n}$

$$\lim_{n\to\infty}\frac{8n.\sqrt[5]{2n}}{n.\sqrt[3]{n}}\Rightarrow\frac{\infty}{\infty}$$

$$\lim_{n\to\infty}\frac{8.(2n)^{1/5}}{(n)^{1/3}}\Rightarrow 8.2^{1/5}.n^{\frac{1}{5}-\frac{1}{3}}$$

$$\lim_{n\to\infty}8.2^{1/5}.n^{-2/15}\Rightarrow\frac{8.2^{1/5}}{n^{2/15}}=0$$

Therefore $f(n)\in O(g(n))$

Part2

GENERAL

a) "methodA" function has single loop that iterates each element of the array by n times. And each iteration is constant time operation. So, worst-case time complexity can be expressed as $O(n)$

$O(n \cdot 1) \Rightarrow O(n)$ (n: each iteration, 1: A)

b) First Loop ⇒ there is a loop iterates over each element of array. and calls "methodA" for each element. Now n times loop and n times inner loop = $O(n^2)$

Second Loop ⇒ Accessing and printing each element is $O(n)$ There is two parts but we're looking for the worst. So worst-case time comp. is $O(n^2)$

c) Outer loop iterates "n times". Inner loop iterates "n times" as well. Inner loop has "methodB". We have already find methodB's time comp.

So, the overall worst-case time comp. → $n \cdot n \cdot n^2 \Rightarrow O(n^4)$ (outer loop, inner loop, methodB)

d) There is a loop iterates "n times". Printing and accessing has constant time operations. But, since the index i inside the loop is manipulated, the loop is infinity aymore.

Worst-case time comp. cannot be determined in this case because the loop never terminates. It's error, throwing stackoverflow

cannot be determined

e) The loop checks each element, but the if condition only breaks the loop when it finds an empty string. But we're looking for the worst. This means the loop checks the entire array. So the worst case time comp. will be $O(n)$, n is size of the array.

Part 3

a) Algorith:

0. If the arry size is less than 2, no difference can be found
1. Calculate the difference between the last element and the first element of the arry. Because arry is ascending order
2. Return 'max-difference'

Pseudo-code:

```
func MaxDiffSorted (array A)
    if length(A) < 2
        return -1
    max-diff = A[length(A)-1] - A[0]
    return max-diff
```

Exp:

Since the array is sorted in ascending order, the last element - first element will give us the result.

The algorithm has O(1) in the worst-case.

b) Algorithm:

0. If the arry size is less than 2, no difference can be found.
1. Find the min element and its index in the arry.
2. Find the max element and " " " " " .
3. Calculate the difference between max element and min element.
4. Return result:

Pseudo Code

```
func MaxDiffUnsorted (array A)
    if length(A) < 2
        return -1
    min = A[0]
    max = A[0]
    for i from 1 to length(A)-1:
        if A[i] < min
            min = A[i]
        else if A[i] > max
            max = A[i]
    return max-min
```

Explanation:

- This algorithm iterates through the array and find both min and max elements.
- After finding, it calculates the difference between them.
- This algorithm operates in linear time comp. since it traverses the array only once.
- Worst-case time complexity:

It is O(n), n is size of the array. This is because it needs to traverse the entire array once to find both min, max element