CSE222 - HW2

Muhammed Ikbal
190104264
M. Ikbal

1- a) Simplify $f(n)$: $f(n) = (n^2-3n)^2 = n^4 - 6n^3 + 9n^2$

$$\lim_{n\to\infty} \frac{n^4-6n^3+9n^2}{5n^3+n} \xrightarrow{\text{divide } n^3} \lim_{n\to\infty} \frac{n-6+\frac{9}{n}}{5+\frac{1}{n^2}}$$

As n approaches infinity, $\frac{9}{n}$ and $\frac{1}{n^2}$ approach 0.

Thus, simplify: $\lim_{n\to\infty} \frac{n-6}{5}$ which is "∞" as an "n" grows.

since the $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ as n approaches infinity is infinity, $f(n)$ grows faster than $g(n)$, and therefore, $\boxed{f(n) = \Omega(g(n))}$.

b) $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{\log_2 n^4} = \lim_{n\to\infty} \frac{n^3}{4\log_2 n} \xrightarrow{\text{it's an indeterminate form } \frac{\infty}{\infty}} \xrightarrow{\text{apply L'Hopital Rule}} \lim_{n\to\infty} \frac{3n^2}{\frac{4}{n\cdot\ln(2)}} = \lim_{n\to\infty} \frac{3n^3\ln(2)}{4}$$

This simplifies to ∞ as n approaches infinity

Since the $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ as n approaches infinity is infinity, $f(n)$ grows faster than $g(n)$, and therefore, $\boxed{f(n) = \Omega(g(n))}$.

c) $$\lim_{n\to\infty} \frac{5n\cdot\log_2(4n)}{n\cdot\log_2(5^n)} = \lim_{n\to\infty} \frac{5n\log_2(4n)}{n^2\log_2(5)} \xrightarrow{\text{it's an indeterminate form } \frac{\infty}{\infty}} \xrightarrow{\text{apply L'Hopital Rule}} \lim_{n\to\infty} \frac{5(2+\log_2(n))}{n\log_2(5)} = \lim_{n\to\infty} \frac{10+5\log_2(n)}{n\log_2(5)} = 0$$

($\log_2 4$)

$$\lim_{n\to\infty} \frac{\frac{d}{dn}(10+5\log_2(n))}{\frac{d}{dn}(n\log_2 5)} = \lim_{n\to\infty} \frac{\frac{5}{n\cdot\ln(2)}}{\log_2(5)} = \lim_{n\to\infty} \frac{5}{n\cdot\ln(2)\cdot\log_2 5} = 0$$

The limit essentially approaches 0, that $f(n)$ grows at a slower rate compared to $g(n)$, therefore, $\boxed{f(n) = O(g(n))}$

d) $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^n}{10^n} = \lim_{n\to\infty} \left(\frac{n}{10}\right)^n$$ As "n" grows, $\frac{n}{10}$ becomes larger the expression grows exponentially.

Since the $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ as n approaches infinity is infinity, $f(n)$ grows faster than $g(n)$, and therefore, $\boxed{f(n) = \Omega(g(n))}$

e) $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{8n(2n)^{\frac{1}{5}}}{n\cdot(n)^{\frac{1}{3}}} = \lim_{n\to\infty} \frac{8\cdot(2)^{\frac{1}{5}}\cdot(n)^{\frac{1}{5}}}{(n)^{\frac{1}{3}}} = \lim_{n\to\infty} (2)^{\frac{16}{5}}\cdot(n)^{\frac{-2}{15}} = \lim_{n\to\infty} \frac{(2)^{\frac{16}{5}}}{(n)^{\frac{2}{15}}} = 0$$

As "n" approaches infinity, the term $\frac{(2)^{\frac{16}{5}}}{(n)^{\frac{2}{15}}}$ approaches 0 because the numerator remains constant while the denominator grows increasingly.

The limit essentially approaches 0, that $f(n)$ grows at a slower rate compared to $g(n)$, therefore $\boxed{f(n) = O(g(n))}$

2- a) "n" is length of the array. The loop runs "n" times. $O(n)$

The assignment operation is a constant time operation. $O(1)$, because it is regardless of the size of array (str_array[i] = "";). Method is in $O(n) \cdot O(1) = O(n)$

So, worst-case time complexity of "methodA" is $\boxed{O(n)}$.

b) "n" is length of the array and "methodA" complexity of $O(n)$. The first loop is called "methodA" for "n" times, the time complexity of this part is $O(n) \cdot O(n) = O(n^2)$

The second loop runs n times, print an elemet ($O(1)$). So, the time complexity of this part $O(n) \cdot O(1) = O(n)$

Thus, worst-case time complexity of "methodB" is $\boxed{O(n^2)}$.

c) "n" is length of the array and "methodB" complexity of $O(n^2)$. The outer loop runs "n" times. ($O(n)$). Inner loop runs "n" times. Thus, it is a nested loop $O(n) \cdot O(n) = O(n^2)$

For each iteration of the inner loop, "methodB" is called.

Thus, the worst-case time complexity of "methodC" is $O(n^2) \cdot O(n^2) = \boxed{O(n^4)}$

d) "n" is length of the array. The loop runs "n" times. ($O(n)$)

Print operation is $O(1)$. and assignment operation is $O(1)$.

But here, every "i++" increment in the loop balances the "i--" decrement in the assigned operation. In this case, the loop becomes infinite because the loop condition is not met.

e) "n" is length of the array. The loop runs "n" times. ($O(n)$)

If-statement that checks if the current element of the array is an empty string. ($O(1)$)

If the condition is true, the loop terminates, but for the worst-case time complexity, we must consider the case where the maximum number of iterations occur before the loop terminates prematurely.

Thus, the worst-case time complexity of "methodE" is $O(n) \cdot O(1) = \boxed{O(n)}$

3- a) In an increasing ordered array the maximum difference will always be between "a_0" and "a_{n-1}".

```
function MaxDiff (A)
    n = length (A)
    if n<2:
        return // Array must have at least two elements"
    max_diff = A[n-1] - A[0]
    return max_diff
```

The algorithm checks that the number of elements is not less than two and since it is sequential, it calculates the difference of first and last elements as the maximum difference

The worst-case time complexity is O(1) because the algorithm performs a fixed number of operations.

b) To find the maximum difference in an unsorted array, we can calculate by creating a starting difference and comparing this difference with the following elements.

```
function UnsortedMaxDiff (A)
    n = length (A)
    if n<2:
        return "Array must have at least two elements"
    minVal = A[0]
    max_diff = A[1] - A[0]
     for i from 1 to n-1:
        if A[i] - minVal > max_diff:
            max_diff = A[i] - minVal
        if A[i] < minVal:
            minVal = A[i]
    return max_diff
```

The algorith checks that the number of elements is not less than two. Initially, it assumes the difference of the the first and second elements as the maximum difference. As we progress through the array, the maximum difference is compared with the smallest element, updates are made in the if states. Conditions are checked until the end of the sequence and the maximum difference is calculated.

The worst-case time complexity is O(n) because the algorithm iterates through the array exactly once, performing a constant amount of work for each element.