Emir İnce
20010004025

Q1)

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

$$\lim_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n} = \lim_{n\to\infty} \frac{n^4-6n^3+9n^2}{5n^3+n} = \lim_{n\to\infty} \frac{n^3-6n^2+9n}{5n^2+1}$$

L'Hospital $\Rightarrow \lim_{n\to\infty} \frac{3n^2-12n+9}{10n}$

L'Hospital $\Rightarrow \lim_{n\to\infty} \frac{6n-12}{10} = \infty$, So $f(n) = \Omega(g(n))$

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

$$\lim_{n\to\infty} \frac{n^3}{\log n^4} = \frac{1}{4} \cdot \lim_{n\to\infty} \frac{n^3}{\log_2 n}$$

L'Hospital $\Rightarrow \frac{1}{4} \lim_{n\to\infty} \frac{3n^2}{\frac{1}{n.\ln(2)}} = \frac{1}{4} \lim_{n\to\infty} 3n^3 \ln(2)$

$$\frac{1}{4} \cdot 3\ln(2) \cdot \lim_{n\to\infty} n^3 = \frac{1}{4} \cdot 3\ln(2) \cdot \infty^3 = \infty$$

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

c) $f(n) = 5n \cdot \log_2(4n)$ and $g(n) = n \cdot \log_2(5^n)$

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

$$= 5 \cdot \lim_{n\to\infty} \frac{\log_2(4n)}{n \cdot \log_2 5} = 5 \cdot \lim_{n\to\infty} \frac{\log_5(4n)}{n}$$

L'Hospital $\Rightarrow 5 \cdot \lim_{n\to\infty} \frac{\frac{1}{\ln(5) \cdot n}}{1} = 5 \cdot \lim_{n\to\infty} \frac{1}{\ln(5) \cdot n}$

$$5 \cdot \frac{1}{\ln(5)} \cdot \lim_{n\to\infty} \frac{1}{n} = 5 \cdot \frac{1}{\ln(5)} \cdot \frac{1}{\infty} = 0$$

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

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

$$\lim_{n \to \infty} \frac{n^n}{10^n} = \left(\frac{\infty}{10}\right)^{\infty} = \infty^{\infty} = \infty$$

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

e) $f(n) = 8n \cdot \sqrt[5]{2n}$ and $g(n) = n \cdot \sqrt[3]{n}$

$$\lim_{n \to \infty} \frac{8n \cdot \sqrt[5]{2n}}{n \cdot \sqrt[3]{n}} = \lim_{n \to \infty} \frac{8 \cdot 2^{1/5} \cdot n^{1/5}}{n^{1/3}}$$

$$= \lim_{n \to \infty} \frac{8 \cdot 2^{1/5}}{n^{2/15}} = 0$$

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

Q2) Analyze the worst case time complexity of the following methods?

a) In the case of methodA, the algorithm iterates through each element of the input array ('str_array') and performs a constant-time operation on each element. Because the length of an array is 'n', the time complexity of the algorith ise $O(n)$.

b) In step 1, calling 'methodA' for each element of $O(n^2)$, because each of the 'n' elements, 'methodA' iterates through the array again.

In step 2, iterating through the array to print each element makes it $O(n)$.

When considering both steps together, the overall time complexity of methodB is dominated by the $O(n^2)$ complexity from step1, so the answer is $O(n^2)$.

c) MethodC has nested loops, each looping through the entire array. Inside the inner loop, methodB, which is $O(n^2)$, is called. This results in $O(n^4)$ operations overall.

d) Due to the issue with decrementing the loop counter inside the loop, methodD is likely to encounter an infinite loop and not complete. So, it's time complexity is not calculable.

e) The time complexity of methodE is O(n), where 'n' is the length of array. This is because it needs to potentially check each element of the array before finding an empty string or reaching the end of the array in the worst case scenario.

Q3)

a) Assumming the Array is Sorted in Ascending order.

When the array is sorted, we can take advantage of the order to find the maximum difference in a single pass.

```
algorithm find_max_difference(array)
    n = len(array)
    if n < 2          // handle cases with less than 2 elements
        return 0
    return array[n-1] - array[0] // Difference between last and first
```

Explanation

1. We check if the array has less then two elements. If so, the maximum difference is zero.
2. In a sorted array, the largest element will be at the end (a[n-1]) and the smallest element at the beggining a[0].
3. Therefore, the maximum difference can be found by subs-tracting the first element from the last element.

Time Complexity

The time complexity of this algorith is O(1) because the running time of the algorithm is independent of the input size. The algorithm calculates the difference between the first and last elements of an array and completes the operation in constant time. Regardless of the input size, this operation takes the same amount of time, so the time complexity is constant.

b) Assumming the Array is not sorted

This approach uses nested loops to compare each element with every other element and keeps track of the maximum difference found so far.

```
algorithm Find_max_difference (array)
    if len(array) < 2
        return 0

    max_difference = abs(array[1] - array[0]
    min_element = array[0]
    for i from 1 to len(array) - 1
        if (abs(array[i] - min_element) > max_difference
            max_difference = abs(array[i] - min_element)
        if array[i] < min_element
            min_element = array[i]
    return max_difference
```

Explanation

1. First it checks if the length of array is less than 2. If so, it returns 0 because we need at least two elements to find diffence
2. Then, it initializes 'max_diff' as the absolute difference between second and the first element of array. It also initializes 'min_element' as the first element of the array.
3. It itaretes through the array. Updates 'max_difference' and 'min_element' as needed.
4. Finally, it returns 'max_difference', which represents the maximum difference between two elements in array

Time Complexity

The algorithm's time complexity is $O(n)$, where n is the size of the array. It iterates through the array once, performing simple operations like comparison and updates.