161044085
Şehmus ACAR

1-)

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

$= \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n}$

→ To simplify this expression, I divide both the two functions by taking their limit

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

$= \lim_{n\to\infty} \frac{n-6+\frac{9}{n}}{5+\frac{1}{n^2}}$ → Here, as n goes to infinity, the terms $\frac{9}{n}$ and $\frac{1}{n^2}$ approach zero, so the limit becomes;

$= \lim_{n\to\infty} \frac{n-6}{5} = \infty$

Result = Since the limit is infinity, $f(n) = \Omega(g(n))$.

As n grows, the f(n) function grows faster than the g(n) function b-) $f(n) = n^3$ $\quad g(n) = \log_2(n^4)$

$= \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{\log_2(n^4)}$ (I use the equality $\log_2(n^4) = 4\log_2(n)$)

$= \lim_{n\to\infty} \frac{n^3}{4\log_2(n)}$ $\left(\text{If } n = 2^t\right) \to \left(t = \log_2(n)\right)$

$= \lim_{t\to\infty} \frac{(2^t)^3}{4t} = \lim_{t\to\infty} \frac{8^t}{4t}$

→ As t goes to infinity, the term $4t$ in the denominator grows, but the term 8^t in the numerator grows much faster.

So, the limit is infinity

$= \lim_{t\to\infty} \frac{8^t}{4t} = \infty$

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

- This show that as n grows, the f(n) function grows faster than the g(n) function c-) $f(n) = 5n \cdot \log_2(4n)$ $\qquad g(n) = n \cdot \log_2(5^n)$

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

I use the equality → $\left(\log_2(4n) = \log_2 4 + \log_2 n = 2 + \log_2 n \text{ and } \log_2(5^n) = n \cdot \log_2(5)\right)$

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

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

→ Here, as n goes infinity, the term $n \cdot \log_2(5)$ in the denominator grows faster than the term $2+\log_2(n)$ in the numerator.

Therefore, the limit approaches zero:

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

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

d-) $f(n) = n^n$, $g(n) = 10^n$

$= \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^n}{10^n}$

$= \lim_{n\to\infty} \left(\frac{n}{10}\right)^n$ → Here, as n goes infinity, the term $\left(\frac{n}{10}\right)^n$ goes to infinity

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

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

---

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

$= \lim_{n\to\infty} \frac{8 \cdot (2n)^{1/5}}{n^{\frac{1}{3}}}$

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

$= \lim_{n\to\infty} 8 \cdot 2^{\frac{1}{5}} \cdot n^{-\frac{2}{15}}$ → As n goes infinity, the term $n^{-\frac{2}{15}}$ approach zero

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

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

2-)

a-) For MethodA

- There's a single loop iterating 'n' times where 'n' is the length of 'str-array'

- Within each iteration, It performs a constant time operation

- So, the overall time complexity of methodA is '$O(n)$'

$O(1) \times O(n) = O(n)$

---

b-) MethodB calls methodA, which has $O(n)$ complexity 'n' times in a loop, giving $O(n) \times O(n)$ $= O(n^2)$ complexity for this part. It then prints elements in another loop, which is $O(1) \times O(n)$ $= O(n)$. Combined, the total complexity for methodB is dominated by the first part $\rightarrow O(n^2)$

---

c-) MethodC has two nested loops, each with 'n' iterations and call methodB, which has $O(n^2)$ complexity

- This result in $O(n) \times O(n) \times O(n^2) = O(n^4)$

d-) Method D has a loop that theoretically runs n times, but due to the 'i--' operation, it creates an infinity loop, which doesn't have a meaningful complexity notation. This is a logical error and makes the method unusable as is.

---

e-) method E iterates through the array, which in the worst case goes through 'n' elements once, so its $O(1) \times O(n) = O(n)$

3-)

a-) function maxDifferenceAscending(arr):

```
n=length (arr)
if n<2:
    return "Insufficient elements in the array"

max_diff = arr[1]- arr [0]
#Assume the difference between the first two elements as the maximum difference
for i from 2 to n-1:
    diff = arr [i]- arr [0]
    if diff >max_diff:
        max_diff =diff

return max_diff
```

This algorithm also involves a simple loop depending on the size of the array.

So time complexity O(n)

b-)

```
function maxDifferenceUnsorted(arr):
    n = length(arr)
    if n<2;
        return "There aren't enough stuff in the series"
    min_element = arr[0]
    max_diff = arr[1]- arr[0]
    for i from 1 to n-1:
        if arr[i]-min_element >max_diff
            max_diff =arr[i] - min_element
        if arr[i]<min_element
            min_element =arr[i]
    return max_diff
```

Time Complexity

This algorithm also involves a single loop depending on the array size

So time complexity $O(n)$