GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

Homework 2 - Solutions

Q1)

a) I followed Limit Asymptotic Theorem in this question, that is:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = L \begin{cases} L = 0 \Rightarrow f(n) = O(g(n)) \\ L = \text{constant } (c) \Rightarrow f(n) = \Theta(g(n)) \\ L = \infty \Rightarrow f(n) = \Omega(g(n)) \end{cases}$$

So, the limit of $f(n)/g(n)$ must be evaluate.

$$\lim_{n\to\infty} \frac{n^2+7n}{n^3+7} = \frac{\frac{1}{n} + \frac{7}{n^2}}{1 + \frac{7}{n^3}} \overset{n\to\infty}{=} \frac{\frac{1}{\infty} + \frac{7}{\infty}}{1 + \frac{7}{\infty}} = \frac{0}{1} = 0$$

Since the limit is zero, we can conclude that $f(n) = O(g(n))$

b) $$\lim_{n\to\infty} \frac{12n + \log_2^{n^2}}{n^2 + 6n} = \frac{\lim_{n\to\infty} \frac{12}{n} + \frac{\log^{n^2}_2}{n^2}}{\lim_{n\to\infty} 1 + \frac{6}{n}} = \lim_{n\to\infty} \frac{12}{n} + \frac{\log_2^{n^2}}{n^2} = 0 + 0 = 0$$

($\lim_{n\to\infty} 1 + \frac{6}{n}$ equals 1)

$n \times n$ grows much faster than the $\log_2^{n^2}$ so this expression is also equals to 0.

$$\lim_{n\to\infty} \frac{12n + \log_2^{n^2}}{n^2 + 6n} = 0$$, we can say that $f(n) = O(g(n))$

c) $$\lim_{n\to\infty} \frac{n \cdot \log_2^{3n}}{n + \log_2^{8n^3}} = \lim_{n\to\infty} \frac{\log_2^{3n^2}}{n + \log_2^{8n^3}} = \frac{6n}{3n^2 \ln 2} = \frac{\lim_{n\to\infty} \frac{6}{(3n)\ln 2}}{\lim_{n\to\infty} 1 + \frac{24}{8n \ln 2}} = \frac{0}{1} = 0$$

$1 + \frac{24n^2}{8n^3 \ln 2}$

Since the limit is zero, we can conclude that $f(n) = O(g(n))$

d)

$$\lim_{n\to\infty} \frac{n^n + 5n}{3.2^n} = \underbrace{\lim_{n\to\infty} \frac{n^n}{3.2^n}}_{\text{This conclude as going to "}\infty\text{". Because } n^n \text{ grows much faster than } 2^n.} + \underbrace{\lim_{n\to\infty} \frac{5n}{3.2^n}}_{\text{This expression equals to "0". Because } 2^n \text{ grows much faster than } 5n.} = \infty + 0 = \infty$$

Since the limit is infinity, we can say that $f(n) = \Omega(g(n))$ //

e) $\lim_{n\to\infty} \frac{\sqrt[3]{2n}}{\sqrt{3n}}$ simplify $= \frac{\sqrt[3]{2}}{\sqrt{3}\sqrt[6]{n}}$

$$= \frac{\sqrt[3]{2}}{\sqrt{3}} \left(\lim_{n\to\infty} \frac{1}{\sqrt[6]{n}} \to 0\right) = \frac{\sqrt[3]{2}}{\sqrt{3}} \cdot \underbrace{\frac{1}{\infty}}_{0} = 0 \text{ //}$$

since the limit is zero, we can conclude that $f(n) = O(n)$ //

Q2)

a) Method A has a worst-case time complexity of $O(n)$, where n is the length of the array "names". This is due to the fact that the approach consists of a single for loop that runs n times and carries out a constant-time operation (a print statement) at each iteration.

b) Method B has a $O(n^2)$ time complexity, where n is the length of the array "myArray". This is because methodB calls methodA in its own for loop and it iterates over the entire array "names" which has a length of n. As a result complexity is equal to $n * n = n^2$.

c) methodC always stays in loop because i value is not increased. So this method doesn't have worst-case time complexity.

d) In the worst case, all of the "numbers" array's elements are less than 4 then that means loop will iterate number of elements in "numbers" array. Let's call it n. Then this methodD has a wast-case time complexity of O(n).

Q3)

withoutLoop has O(n) complexity because it prints the elements of the myArray n times, where n is the length of the myArray.

witLoop also has O(n) complexity because it consist of a single for loop that runs n times, where n is the length of the myArray.

Both method have the same time complexity but witLoop method is more advantageous because it is easier to read and maintainable rather than withoutLoop method.

Q4)

No, I think it is not possible to solve this problem in constant time. Because inspecting each array element takes O(n) times to check the entire array. Beside of this, we didn't even know array is sorted or not so we might have to search every element of the array.

Q5)

we can write the algorithm as in the below:

1. Define minValue and initialize to max Value that computer have.
2. Initialize i and j to 0
3. while i < n and j < m:
   a. compute $a_i * b_j$
   b. If computed value is less than minValue then update the minValue
   c. If $a_i < b_j$ increment i by one.
   d. Else increment j by one.
4. Return minValue.

Algorithm starts with the first element of the each array and keep compares the minimum value of the each product. In while loop we also compare the both elements value and we increase the value of smaller one because we might get smaller Product value in that array's values.

Here is the implementation in Java:

```
Public int minProduct (int[] A, int[] B, int n, int m) {
    int minValue = Integer.MAX_VALUE;
    int i=0, j=0;
    while (i<n && j<m) {
        int Prod = A[i] * B[j];
        if (prod < minValue) {
            minValue = Prod;
        }
    if (A[i] < B[j]) {
            i++;
    }
    else {
        j++;
    }
    }
  return minValue;
}
```

Question wants our to design to have linear time algorithm so this algorithm is what I design in the time complexity of O(n+m) because I only traverse both arrays only once.