CSE222 - HW2

Elifnur Kahraler
180104261 7

1.)

a) $f(n) = n^2 + 7n$, $g(n) = n^3 + 7$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^2+7n}{n^3+7} \Rightarrow \left(\text{ambiguity } \frac{\infty}{\infty}\right) \Rightarrow \text{L'Hospital}$$

$$= \frac{(n^2+7n)'}{(n^3+7)'} = \frac{2n+7}{3n^2} \Rightarrow \text{L'Hospital} = \frac{2}{6n}$$

$\Rightarrow$ The limit is 0, so $f(n) = O(g(n))$. Therefore, we can say that $f(n)$ grows slower than $g(n)$, or equivalently, $g(n)$ grows faster than $f(n)$.

b) $f(n) = 12n + \log_2 n^2$, $g(n) = n^2 + 6n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{12n + \log_2 n^2}{n^2+6n} \Rightarrow \text{divide by } n^2 = \frac{\left(\frac{12}{n}\right) + \left(\frac{\log_2 n^2}{n^2}\right)}{1 + \left(\frac{6}{n}\right)}$$

$\Rightarrow \left(\frac{12}{n}\right)$ and $\left(\frac{6}{n}\right)$ both approach zero, and log approaches zero as well because the logarithmic function grows much slower than any power function. Therefore, the limit approaches zero, which means that $f(n) = O(g(n))$, $g(n)$ grows faster than $f(n)$.

c) $f(n) = n \cdot \log_2 (3n)$, $g(n) = n + \log_2 (8n^3)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n \cdot \log_2 (3n)}{n + \log_2 (8n^3)} \Rightarrow \text{divide by } n = \frac{\log_2 3n}{1 + \frac{\log_2 (8n^3)}{n}}$$

$\Rightarrow \log_2 (8n^3)/n$ approaches zero because the logarithmic function grows much slower than any power function. Therefore, the denominator approaches 1, and the numerator approaches infinity, but much slower than n. Therefore, the limit approaches 0, $f(n) = O(g(n))$. $g(n)$ grows faster than $f(n)$.

d) $f(n) = n^n + 5n$, $g(n) = 3 \cdot 2^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^n+6n}{3 \cdot 2^n} = \frac{n^n}{3 \cdot 2^n} + \frac{5n}{3 \cdot 2^n} = \frac{(n/2)^n}{3} + \frac{(5/2)^n}{3}$$

$\Rightarrow f(n)$ grows faster than $g(n)$, because of $\frac{(n/2)^n}{3}$ is infinity, $(n/2)^n$ grows faster than 3. $\frac{(5/2)^n}{3}$ is infinity, $(5/2)^n$ grows faster than 3. Therefore $f(n) = \Omega(g(n))$.

c) $f(n) = \sqrt[3]{2n}$, $g(n) = \sqrt{3n}$

$$\lim_{n\to\infty} \frac{\sqrt[3]{2n}}{\sqrt{3n}} = \frac{\sqrt[3]{2}\cdot\sqrt[3]{n}}{\sqrt{3}\cdot\sqrt{n}} = \frac{\sqrt[3]{2}}{\sqrt{3}\cdot\sqrt[6]{n}} \Rightarrow \frac{\sqrt[3]{n}}{\sqrt{n}} = \frac{n^{1/3}}{n^{1/2}} = n^{\left(\frac{1}{3}-\frac{1}{2}\right)} = n^{-1/6}$$

=) $f(n)$ grows slower than $g(n)$ or $f(n) = O(g(n))$.

2.)

a)
```
Static void methodA (String names[]){   // (names.lenght = n)
    for(int i=0; i<n ; i++){
       System.out.println(names[i]);
    }
}
```
=) Complexity of methodA is increases linearly. So, $O(n)$.

b)
```
Static void methodB(){
   String[] myArray = new String[] { "CSE222", "CSE505", "HW2"};
   for(int i=0; i<n; i++){
     methodA(myArray);
   }
}
```
=) Since methodA has a time complexity of $O(n)$, and the loop iterates over myArray n times, the time complexity of the for loop is $O(n^2)$.

c)
```
Static void methodC(int numbers[]){
   int i=0;
   while (i< n){
     System.out.println(numbers[i]);
   }
}
```
=) It is infinite loop. $O(\infty)$. Because of i is not increment in while loop.

d)
```
Static void methodD (int numbers[]){
   int i=0;
   while(numbers[i] <4)
     System.out.println(numbers[i++]);
}
```
=) If numbers[0] greater than 4, $O(1)$ if all elements of numbers array are less than 4, $O(n)$.

=) best to worst

methodD , methodA , methodB , methodC

$O(1)$-$O(n)$ $O(n)$ $O(n^2)$ $O(\infty)$

3.) Both of them's complexity is O(n).
Compared to withoutLoop method, withloop method is more advantageous in cases where the number of iterations is dynamic or large, as it allows us to iterate over the array without writing out each statement explicitly. It also allows for more flexibility in terms of modifying the loop condition or adding additional operations within the loop body.

In general, using loop leads to more maintainable and readable code, and reduces the chances of errors or omissions.

4.) The best possible time complexity for this problem is O(n).
It is not possible to solve this problem in constant time since you have to look at every element in the worst case to determine if the specific integer is present in the array.

5.) a) Initialize the minimum value to positive infinity.
b) Initialize two variables i and j, to 0.
c) Traverse both arrays A and B simultaneously using i and j indices.
d) At each step, update the minimum value by taking the minimum of the current minimum value and the product of the current elements of arrays A and B.
e) If the product of the current elements of arrays A and B is less than 0, then increment either i or j, depending on which element is smaller in magnitude, since multiplying a negative number with a positive number would result in a negative number.
f) Repeat steps d-e until we have traversed the entire arrays A and B.

Code:

```
public static int findMin(int[] A, int[] B){
    int minval = Integer.MAX_VALUE;
    int i=0, j=0;
    while (i < A.lenght && j < B.length){
        int currentval = A[i] * B[j];
        if (currentval < minval){
            minval = currentval;
        }
        if(currentval <0){
          if ( Math.abs(A[i]) < Math.abs(B[j])){
              i++;
          } else{
              j++;
          }
        } else if (B[j] < A[i]){
            j++;
        } else{
            i++;
        }
    }
    return minval;
}
```

=> Time complexity is O(n+m). Linear Time.