Data Structures and Algorithms - Homework #2

Name/Surname: Emre Kibar   Student Number: 210104004093

Question 1

ⓐ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n} = \lim\limits_{n\to\infty} \frac{n^4-6n^3+9n^2}{5n^3+n} = \lim\limits_{n\to\infty} \frac{n^3(n-6+\frac{9}{n})}{n^3(5+\frac{1}{n^2})}$

$\Rightarrow \lim\limits_{n\to\infty} \frac{n-6+\frac{9}{n}}{5+\frac{1}{n^2}} = \frac{\infty-6+\frac{9}{\infty}}{5+\frac{1}{\infty}} = \frac{\infty}{5} = \infty \Rightarrow \boxed{f(n) \in \Omega(g(n))}$

ⓑ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{n^3}{\log_2 n^4} = \lim\limits_{n\to\infty} \frac{n^3}{4.\log_2 n} = \frac{1}{4} \lim\limits_{n\to\infty} \frac{n^3}{\log_2 n} = \frac{\infty}{\infty}$

$\xrightarrow[\text{l'hospital rule}]{\text{Using}} \frac{1}{4} \lim\limits_{n\to\infty} \frac{\frac{d}{dn}(n^3)}{\frac{d}{dn}(\log_2 n)} = \frac{1}{4} \lim\limits_{n\to\infty} \frac{3n^2}{\frac{1}{\ln(2).n}} = \frac{1}{4} \lim\limits_{n\to\infty} 3n^3 \ln(2)$

$= \frac{3\ln(2)}{4} \lim\limits_{n\to\infty} n^3 = \infty \Rightarrow \boxed{f(n) \in \Omega(g(n))}$

ⓒ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{5n.\log_2(4n)}{n.\log_2(5^n)} = \lim\limits_{n\to\infty} \frac{5n\log_2(4n)}{n^2\log_2(5)} = \frac{5}{\log_2(5)} \lim\limits_{n\to\infty} \frac{\log_2(4n)}{n} = \frac{\infty}{\infty}$

$\xrightarrow[\text{l'hospital rule}]{\text{Using}} \frac{5}{\log_2(5)} \lim\limits_{n\to\infty} \frac{\frac{d}{dn}(\log_2(4n))}{\frac{d}{dn}(n)} = \frac{5}{\log_2(5)} \lim\limits_{n\to\infty} \frac{\frac{1}{\ln(2)n}}{1}$

$= \frac{5}{\log_2(5)} \lim\limits_{n\to\infty} \frac{1}{\ln(2)n} = 0 \Rightarrow \boxed{f(n) \in O(g(n))}$

ⓓ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{n^n}{10^n} = \lim\limits_{n\to\infty} \left(\frac{n}{10}\right)^n$

$\left.\begin{array}{l} \Rightarrow \lim\limits_{n\to\infty} \frac{n}{10} = \frac{1}{10} \lim\limits_{n\to\infty} n = \infty \\ \lim\limits_{n\to\infty} n = \infty \end{array}\right\} \Rightarrow \lim\limits_{n\to\infty} \left(\frac{n}{10}\right)^n = \infty^\infty = \infty$

$\Rightarrow \boxed{f(n) \in \Omega(g(n))}$

ⓔ $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \lim\limits_{n\to\infty} \frac{8n\sqrt[5]{2n}}{n\sqrt[3]{n}} = \lim\limits_{n\to\infty} \frac{8(2n)^{\frac{1}{5}}}{n^{1/3}} = 8 \lim\limits_{n\to\infty} \frac{2^{1/5} n^{1/5}}{n^{1/5}(n^{2/15})}$

$\Rightarrow \sqrt[5]{2}.8 \lim\limits_{n\to\infty} \frac{1}{\sqrt[15]{n^2}} = 0 \Rightarrow \boxed{f(n) \in O(g(n))}$

CamScanner ile tarandı

Question 2

ⓐ
```
static void methodA(String str_array[]){
    for(int=0; i<str_array.length; i++)
        str_array[i]="";
}
```

→ Declaration of method is constant complexity, $O(1)$.

→ This loop iterates over each element and it will run "n" iterations. So, complexity is $O(n)$.

→ Assigning an element is constant complexity, $O(1)$ but inside the loop it is implemented "n" times, so complexity is $O(n)$.

⇒ Time complexity of this method is $O(1)+O(n)=O(n+1)$ or by simplifying it is $\boxed{O(n)}$

ⓑ
```
static void methodB(str_array[]){
    for(int i=0; i<str_array.length; i++)
        methodA(str_array);
    for(int j=0; j<str_array.length; j++)
        System.out.println(str_array[j]);
}
```

→ Declaration of method is constant complexity, $O(1)$.

→ This loop iterates over each element and it will run "n" iteration. So, complexity is $O(n)$.

→ The complexity of methodA is $O(n)$ and it will be called n times inside the loop. So, complexity is $O(n^2)$.

→ This loop iterates over each element and it will run "n" iteration. So, complexity is $O(n)$.

Printing an element has constant complexity, $O(1)$ but within the loop, it will be implemented n times. So, complexity is $O(n)$.

⇒ Time complexity of this method is $O(1)+O(n^2)+O(n)=O(n^2+n+1)$. Then, we can simplify it as $\boxed{O(n^2)}$ which is fastest growing part.

ⓒ
```
static void methodC(String str_array[]){
    for(int i=0; i<str_array.length; i++)
        for(int j=0; j<str_array.length; j++)
            methodB(str_array);
}
```

→ Declaration of method is constant complexity, $O(1)$.

→ This loop iterates over each element and it will run "n" iterations. So, complexity is $O(n)$.

→ This loop iterates over each element and it will run "n" iterations. So, complexity is $O(n)$ but this loop is a nested loop and it will be called n times. So, total complexity is $O(n.n)=O(n^2)$

The complexity of methodC is $O(n^2)$ and within the nested loop, it will be called n^2 times. Then, the complexity is $O(n^2.n^2)=O(n^4)$

⇒ Time complexity of this method is $O(1)+O(n^4)=O(n^4+1)$. By simplifying, the complexity of methodC is $\boxed{O(n^4)}$.

CamScanner ile tarandı

(d) static void methodD (String str_array[]){ → Declaration of method is constant complexity, O(1).

```
for(int i=0; i<str_array.length; i++){
    System.out.println(str_array[i]);
    str_array[i--] = "";
  }
}
```

→ This loop iterates over each element and it will run "n" iterations. So, complexity is O(n).

→ Printing an element is constant complexity, O(1).

This line cause an infinite loop because as a loop condition "i" is incremented each iteration but in this line, "i" is decremented. So, this loop will not reach ending condition and it has no meaningful time complexity.

⇒ Time complexity of this method cannot be calculated due to infinite loop problem.

(e) static void methodE(String str_array[]){ → Declaration of method is constant complexity, O(1).

```
for(int=0; i<str_array.length; i++)
    if(str_array[i] == "")
      break;
}
```

→ This loop iterates over each element in the worst-case which means there isn't any empty string within the "str_array". So, it iterates "n" times and the complexity is O(n).

Checking for empty string and "break" has constant complexity, O(1) but in worst-case scenario, there wouldn't be an empty string. So, loop continue the iteration and each iteration check the if condition. So, the worst-case complexity is O(n).

⇒ Time complexity of this method in worst case is $\boxed{O(n)}$.

CamScanner ile tarandı

Question 3

ⓐ
```
METHOD FindMaxDifference(A)
    IF A.length <2
        RETURN
    ENDIF
    RETURN A[A.length-1] - A[0]
ENDMETHOD
```

- Method declaration has constant complexity, $O(1)$.
- Checking the length of "A" for validity using if statement has constant complexity, $O(1)$.
- Returning the difference and subtracting two integers has both constant complexity, $O(2.1) = O(1)$.

$\Rightarrow$ The time complexity of this method is $O(1)$ as the dominant factor in the worst-case scenario.

ⓑ
```
METHOD FindMaxDifference(A)
    IF A.length <2
        RETURN
    ENDIF
    maxInt = A[0]
    minInt = A[0]
    FOR i from 0 to A.length-1
        IF A[i] > maxInt
            maxInt = A[i]
        ELSE IF A[i] < minInt
            minInt = A[i]
        ENDIF
    ENDFOR
    RETURN maxInt - minInt
ENDMETHOD
```

CamScanner ile tarandı

- Declararation of the method has constant complexity, O(1).
- Checking the length of "A" for validity using if statement has constant complexity, O(1).
- Initializing "maxInt" and "minInt" as "A[0]" is constant complexity, O(1).
- The for loop that is used for finding the greatest and the smallest integer iterates over every element within the "A" and it has linear complexity, O(n).
- The if statements which controls the "A[i]" for "maxInt" and "minInt" have constant complexity, O(1) but within the loop, the conditions checked "n" times. So, the total complexity is linear, O(n).
- Returning the difference and subracting "maxInt" and "minInt" has both constant complexity, O(1).

⇒ The time complexity of this method is O(n) as the dominant factor in the worst case scenario.

CamScanner ile tarandı