Buket GENGER 210104006298 CSE 222 HW-2

Q1)

Limit approach:

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \rightarrow f(n) = O(g(n))$ Big O notation

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \rightarrow f(n) = \Omega(g(n))$ Big Omega notation

- $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \rightarrow f(n) = \Theta(g(n))$ Big Theta notation

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

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

$$\xrightarrow{\text{L'hospital}} \lim_{n\to\infty} \frac{4n^3-18n^2+18n}{15n^2+1} = \frac{\infty}{\infty}$$

$$\xrightarrow{\text{L'hospital}} \lim_{n\to\infty} \frac{12n^2-36n+18}{30n} = \frac{\infty}{\infty}$$

$$\xrightarrow{\text{L'hospital}} \lim_{n\to\infty} \frac{24n-36}{30} = \infty$$

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

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

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

$$\xrightarrow{\text{L'hospital}} \lim_{n\to\infty} \frac{3n^2}{\frac{4}{\ln(2).n}} = \lim_{n\to\infty} \frac{3.\ln(2).n^3}{4} = \infty$$

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

c) $f(n) = 5n.\log_2^{(4n)}$ and $g(n) = n.\log_2^{(5^n)}$

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

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

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

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

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

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

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{8n\sqrt[5]{2n}}{n\sqrt[3]{n}} = \lim_{n\to\infty} \frac{8.n.(2n)^{\frac{1}{5}}}{n.n^{\frac{1}{3}}} = \lim_{n\to\infty} \frac{8\sqrt[5]{2}\ 1}{\sqrt[15]{n^2}}$$

$$\lim_{n\to\infty} \frac{8.\sqrt[5]{2}}{\sqrt[15]{n^2}} = 0$$

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

Q2) Analyze the worst-case time complexity of the following methods. Assume that all of the arrays has length of n, where $n \in Z^+$

a)

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

`int i =0;` → takes constant time $O(1)$

`i< str_array.length;` → takes n time $O(n)$

`i++` → takes n time $O(n)$

`str_array [i] ="";` → takes n time $O(n)$

The methodA is in $O(3n+1) = O(n)$

b)

```
static void methodB (String 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];
}
```

`int i =0;` → takes constant time $O(1)$

`i< str_array.length;` → takes n time $O(n)$

`i++;` → takes n time $O(n)$

`methodA (str_array);` → as a function methodA is in $O(3n+1)$

methodA inside the for loop (n times)

so → $(3n+1) \cdot n = O(3n^2+n)$

(methodA, loop)

`int j =0;` take constant time $O(1)$

`j< str_array.length` → takes n time $O(n)$

`j++` → takes n time $O(n)$

`System.out.println (str_array[j];` → takes n time $O(n)$

$1+n+n+3n^2+n+1+n+n+n$

The method B is in $O(3n^2+6n+2) = O(n^2)$

c)

```
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) ;
}
```

int i=0; → take constant time O(1)

i < str_array.length → take n time O(n)

i++; → take n time O(n)

int J; → take 1.n time O(n) (this loop is inside other loop)

J < str_array.length; → take n.n = n^2 times $O(n^2)$

J++; → take n.n = n^2 times $O(n^2)$

method B (str_array); → $n^2 . (3n^2 + 6n + 2) = 3n^4 + 6n^3 + 2n^2$ times

↙ for 2 times loop ↓ method B $O(3n^4 + 6n^3 + 2n^2)$

The method C is in $O(3n^4 + 6n^3 + 4n^2 + 3n + 1) = O(n^4)$

d)

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

The code contains a for loop where the loop voloble i is incremented each iteration. However, inside the loop 'i' is also decremented, which causes the loop to run indefinitely. This is because the decrement operation 'i--', cancels out the increment operation 'i++', leading to i never reaching the termination condition of the loop, As a result the program gets stuck in an infinite loop.

Loop is infinite and will never terminate.

Therefore the worst-case time complexity of methodD is undefined.

e)

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

int i = 0; → take constant time O(1)

i< str_array.length; → take n times O(n)

i++; → take n time O(n)

if (str_array == "";) → take 1.n time O(n)

∝ In the worst_case, all elements of the string array are checked by the if statement and the for loop doesn't break. It means that none of the elements in the string array are empty.

Therefore the loop will iterate n times. Assume that the break statement is not executed.

The worst_case time complexity of methodE is O(n).

O(3n+1) = O(n).

Q3) Design an algorithm to find the maximum difference between two elements of a given array $A = [a_0, a_1, \ldots, a_{n-1}]$ where $a_i \in Z$ for $i \in N$ and $i < n$. Provide the pseudo-code of the algorithm along with an explanation and analyze it worst-case time complexity. Repeat this process for the following cases:

a) Assuming the array is sorted in ascending order,
b) Assuming the array is not sorted.

a)

```
function max_difference_sorted(arr):
    if length(arr) < 2 then
    return 0 // Array must have at least two elements
    end if
    return arr[length(arr) - 1] - arr[0]
end function
```

if (length(arr) < 2 → take constant time O(1)
return 0 → take constant time O(1)
return arr[length(arr) - 1] - arr[0] → take constant time O(1)

Code operates in constant time O(1). it perform a fixed number of operations regardless of array's size.

- Therefore the worst-case time complexity of code is O(1)

b)

```
function max_difference_unsorted(arr):
    if length(arr) < 2 then
        return 0
    end if

    min_element = arr[0] // Initialize with first element
    max_diff = min_int // Initialize with the min possible value

    for i from 1 to length(arr)-1:
        if arr[i] < min_element then
            min_element = arr[i]
        end if

        diff = arr[i] - min_element
        if diff > max_diff then
            max_diff = diff
        endif
    end for
    return max_diff
end function
```

O(n)

The complexity of the code until the for loop is O(1). This portion performs a constant number of operations and is indepented of the size of array.

if we look at the part of for loop, The worst case time complexity of the 'for loop part' is O(n). Because for loop iterates through the array length, which is n.

Therefore the time complexity is O(n)