CSE 222 Homework #2

Bedrhan Ömer Aksoy
200104004074

Q1)

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

limit approach:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{(n^2-3n)^2}{5n^3+n} = \frac{n^4-6n^3+6n^2}{5n^3+n} \begin{matrix} \times 1/n^3 \\ \times 1/n^3 \end{matrix}$$

$$= \frac{n-6+\cancel{\frac{6}{n}}}{5+\cancel{\frac{1}{n^2}}} = \frac{\infty-6+0}{5+0} = \infty$$

$\underline{f(n) = \Omega(g(n))}$

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

$$\lim_{n\to\infty} \frac{n^3}{\log_2 n^4} = \frac{n^3}{4\cdot\log_2 n} = \frac{\infty}{\infty} \Rightarrow \text{L'Hôpital rule}$$

$$= \left(\frac{n^3}{4\cdot\log_2 n}\right)' = \frac{3n^2}{4\cdot\frac{1}{n\cdot\ln 2}} = \frac{3\cdot n^3\cdot\ln 2}{4} \rightarrow \frac{3\cdot\infty\cdot\ln 2}{4} = \infty$$

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

limit approach:

$\underline{f(n) = \Omega(g(n))}$

c) $f(n) = 5n\cdot\log_2 4n$ and $g(n) = n\cdot\log_2(5^n)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{5\cancel{n}\cdot\log_2 4n}{\cancel{n}\cdot\log_2(5^n)} = \frac{5\cdot\log_2 4n}{n\cdot\log_2 5} = \frac{5}{n}\cdot\log_5 4n = \frac{5\log_5 4n}{n}$$

$$= \frac{\infty}{\infty} \Rightarrow \text{L'Hôpital Rule}$$

$$= \frac{5}{4n\cdot\ln 5} = \frac{5}{4\cdot\infty\cdot\ln 5} = 0$$

limit approach:

$\underline{f(n) = O(g(n))}$

Bedirhan Ömer Aksoy
20010400074

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

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

$$= n^n = n^{n \cdot \ln e} = e^{n \cdot \ln n} \quad , \quad 10^n = 10^{n \cdot \ln e} = e^{n \cdot \ln 10}$$

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

$$e^{\infty \cdot \ln(\frac{\infty}{10})} = \infty \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$ limit approach:

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

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{8\cancel{n} \cdot \sqrt[5]{2n}}{\cancel{n} \cdot \sqrt[3]{n}} = \frac{8(2n)^{\frac{1}{5}}}{\sqrt[3]{n}} = \frac{\infty}{\infty}$$ = L'Hôpital Rool

$$= \frac{\frac{8}{5} \cdot (2n)^{\frac{-4}{5}}}{\frac{1}{3} n^{\frac{-2}{3}}} = \frac{24}{5} \cdot \frac{(2n)^{\frac{-4}{5}}}{n^{\frac{-2}{3}}} = \frac{24}{5} \cdot \frac{2^{\frac{-4}{5}}}{n^{\frac{4}{5}} \cdot n^{\frac{-2}{3}}} = \frac{24}{5} \cdot \frac{2^{\frac{-4}{5}}}{\infty} = 0$$

limit approach: $f(n) = O(g(n))$

Bedirhan Omer Aksoy
20210404074

Q2)

a)

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

steps/exec	freq	total
2	$n+1$	$2n+3$
1	n	n
		$3n+3$

(+1 for int i=0)

$O(3n+3)$

worst-case time complexity $= O(n)$

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]);
}
```

steps/exec	freq	total
2	$n+1$	$2n+3$
$3n+3$	n	$3n^2+3n$
2	$n+1$	$2n+3$
1	n	n
		$3n^2+8n+6$

(+1 for int i=0)

(+1 for int j=0)

worst-case time complexity $= O(3n^2+8n+6) = O(n^2)$

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);
}
```

steps/exec	freq	total
2	$n+1$	$2n+3$
2	$n\cdot(n+1)$	$2n^2+3$
$3n^2+8n+6$	$n\cdot n$	$3n^4+8n^3+6n^2$
		$3n^4+8n^3+8n^2+2n+6$

(+1 for int i=0)

worst-case time complexity $= O(3n^4+8n^3+8n^2+2n+6) = O(n^4)$

(+1 for int i=…

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--] = "";
    }
}
```

steps/exec	freq	total
2	n+1	2n+3
1	n	n
2	n	2n
		5n+3

In the code part // str_array[i--] = "";
it causes an infinite loop. It conflicts with i++ in the loop iteration and i index stays same. It can't be expressed in terms of big O notation because the algorithm doesn't terminates.

(+1 for int i=0

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

steps/exec	freq	total
2	n+1	2n+3
1	n	n
1	0	0
		3n+3

freq = 0, because we aim to find worst-case scenario and if condition satisfies, loop will be exitted and its not the worst time complexity

worst-time complexity = $O(3n+3)$
$= O(n)$

Bedirhan Ömer Aksoy
20010400407[illegible]

20010400074
Bedirhan Ömer Aksoy

Q3)

a) Assuming the array is sorted in ascending order.

Algorithm:

- Initialize two variables max_diff and min_element
- Iterate the array A from left to right.
- Update min_element to the current element if its less than the current min_element.
- Update max_diff to the maximum of max_diff and the difference between the current element and min_element.
- Return max_diff.

Pseudo-code:

```
function max_difference (A):
    max_diff = A[1] - A[0]
    min_element = A[0]

    for i from 1 to length(A)-1:
        if A[i] - min_element > max_diff:
            max_diff = A[i] - min_element
        if A[i] < min_element:
            min_element = A[i]

    return max_diff
```

This algorithm iterates through the sorted array once and follows the minimum element found so far. It calculates the difference between the current element and min_element and updates max_diff if a larger difference is found. Since it only requires a single search through the array, the time complexity is linear O(n).

Bedirhan Ömer Aksoy
20010400074

b) Assuming the array is not sorted

Algorithm:

- Initialize two variables min_element and max_element.
- Iterate the array A from start to end.
- Update the min_element to the current element if its less than the current min_element.
- Update the max_element to the current element if its greater than the current max_element.
- Return the difference between max_element and min_element.

Pseudo-code:

```
function max_difference(A):
        min_element = A[0]
        max_element = A[0]
        for i from 1 to length(A) - 1:
            if A[i] < min_element:
                min_element = A[i]
            if A[i] > max_element:
                max_element = A[i]

        return (max_element - min_element)
```

This algorithm iterates through the array and finds the minimum and maximum elements. At the end, it returns the difference between max_element and min_element. Since, it only requires a single search through the array, the time complexity is linear, O(n).