Elifnur Kabalcı
180104017

CSE 222 - HW2

1)

a) $\lim_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n} = \frac{n^2(n-3)^2}{n(5n^2+1)} = \frac{n\cdot(n^2-6n+9)}{5n^2+1} = \frac{n^3-6n^2+9n}{5n^2+1} \approx \frac{n^3}{n^2} \Rightarrow n \to \infty$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \to f(n) = \Omega(g(n))$

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^3}{\log_2 n^4} = \frac{n^3}{4\cdot\log_2 n} \Rightarrow$ L'hospital $\Rightarrow \frac{3n^2}{4\cdot\frac{1}{n\cdot\ln 2}} = \frac{3}{4}\cdot n^2\cdot n\cdot\ln(2)$

$\Rightarrow \frac{3}{4}\cdot\infty^2\cdot\infty\cdot\ln 2 \Rightarrow \infty \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \to f(n) = \Omega(g(n))$

c) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{5n\cdot\log_2^{4n}}{n\cdot\log_2^{5n}} = \frac{5n\cdot\log_2^{4n}}{n^2\cdot\log_2^5} = \underbrace{\frac{5}{\log_2^5}}_{\text{constant}}\cdot\frac{\log_2^{4n}}{n} \Rightarrow \frac{\log_2^4+\log_2^n}{n} = \frac{2+\log_2^n}{n}$

$\Rightarrow$ L'hospital $\Rightarrow \frac{\frac{1}{n\cdot\ln 2}}{1} = \frac{1}{n\cdot\ln 2} \Rightarrow \underbrace{\frac{1}{\ln 2}}_{\text{constant}}\cdot\underbrace{\frac{1}{n}}_{} \Rightarrow \frac{1}{n} = \frac{1}{\infty} \Rightarrow 0$

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

d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^n}{10^n} = \left(\frac{n}{10}\right)^n \Rightarrow$ n goes to infinity so it grows faster than 10, so, $\frac{\infty}{\text{constant}} \to \infty$

$\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty, \to f(n) = \Omega(g(n))$

e) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{8n\cdot\sqrt[5]{2n}}{n\cdot\sqrt[3]{n}} = 8\cdot\frac{\sqrt[5]{2n}}{\sqrt[3]{n}} = 8\cdot\frac{(2n)^{1/5}}{n^{1/3}} = \underbrace{8\cdot 2^{1/5}}_{\text{constant}}\cdot\frac{n^{1/5}}{n^{1/3}} = n^{1/5-1/3} = n^{-2/15} = \frac{1}{\sqrt[15]{n^2}}$

$\Rightarrow \frac{1}{\infty} \Rightarrow 0 \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \to f(n) = O(g(n))$

2)

a) Static void methodA (String str_array[]){

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

`str_array[i] = " ";` → O(1)

n times O(1) => O(n)

str_array[0] = " " → O(1)
str_array[1] = " " → O(1)
⋮
str_array[n] = " " → O(1)

} n times

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]);
}
```

`methodA (str_array);` → O(n)

assign
str_array[0] = " "
⋮
str_array[n] = " "
} i=0

str_array[0] = " "
⋮
str_array[n] = " "
} i=n-1

} n times O(n) ⇓ O(n²)

`System.out.println (str_array[j]);` → O(1)

print
str_array[0]
⋮
str_array[n]
} n times O(1) ⇓ O(n)

O(n²) + O(n) => O(n²)

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)
}
```

`methodB (str_array)` → O(n²)

} n times O(n²) => O(n³)

} n times O(n³) ⇓ O(n⁴)

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--] = " ";
}
```

=> i → 0
str_array[0] → print
str_array[0] → assign
i → -1
for => i → 0

} infinite loop

=> i cannot increase. [i--] decrease, for increase => O(∞)

e) Static void methodE (String str_array[]){

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

=> If str_array's any node assigned to " ", it goes until the end
So, it search n times to array
→ O(n)

3)

a) Maximum Difference - Ascending Order

```
Algorithm max_difference (arr[])
    min <- arr[0]  ---------- O(1)
    max <- arr[arr.length - 1] -- O(1)
    return (max - min) ------ O(1)
```

// 1, 2, 3, 4, 5
↓ arr[0] ↓ arr[arr.length - 1]

=> Total Big-O is constant

=> O(1)

b) Maximum Difference - Not Sorted

```
Algorithm max_difference (arr[])
    min <- ∞ ----------> O(1)   // Assign max_value for initialy
    max <- (-∞) -------> O(1)   // Assign min_value for initialy
    for (i=0; i<arr.length; i++) ----> O(n)
        if (arr[i] < min) ---------> O(1)
            min <- arr[i] ---------> O(1)
        if (arr[i] > max) ---------> O(1)
            max <- arr[i] ---------> O(1)
    return (max - min) ------------> O(1)
```

// I write if - if instead of // if - else if. Because maybe list // have only one element, so // min and max will be the same.

// for area is n time O(1)
-> So it will linear big-O.
-> O(n)

for A and B:

- Before calling functions, we need to check array length. List is empty or not.

```
Algorithm is_empty (arr[])
    if ( arr.length <= 0 )
        return true ------> // list is empty
    else
        return false -----> // list is not empty
```

// This method has O(1). Big-O Notation is constant.