ICS Answer Sheet #2

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

Problem 2.1: time complexity and landau sets

Given,

Two parts executed sequentially,

Size of input = n

Time complexity of the first part, $t_1(n) = 5n^2 + 16$

Time complexity of the second part, $t_2(n) = 6n^3 + n^2 + 18$

a. **To which big *O* sets do t_1 and t_2 belong?**

Method I

To know which big *O* sets these belong to we must look at two scenarios:

i. Which term grows the fastest in the expression and,
ii. remove its coefficient.

For t_1, that is $5n^2$ and by removing 5, we get, $O(n^2)$
For t_2, that is $6n^3$ and by removing 6, we get, $O(n^3)$

Method 2:

For t_1,

$Kn^2 \geq 5n^2 + 16$ ($\because$ after $n \geq n_0$)

Let K = 6,

$6n^2 \geq 5n^2 + 16$

$6n^2 – 5n^2 - 16 \geq 0$

$n^2 -16 \geq 0$

Let n = 1,

$-15 \geq 0$ (False)

Let n = 2,

$-12 \geq 0$ (False)

Let n = 3,

$-7 \geq 0$ (False)

Let n = 4,

$0 \geq 0$ (True)

Therefore, at $n \geq 4$ ($n \geq n_o$) for K = 6, the inequality becomes true and so the Big *O* notation will be $\boldsymbol{O(n^2)}$

For t_2,

$Kn^3 \geq 6n^3 + n^2 + 18$ ($\because$ after $n \geq n_0$)

Let K = 7,

$7n^3 \geq 6n^3 + n^2 + 18$

$7n^3 - 6n^3 - n^2 - 18 \geq 0$

$n^3 - n^2 - 18 \geq 0$

Let n = 1,

$-18 \geq 0$ (False)

Let n = 2,

$-14 \geq 0$ (False)

Let n = 3,

$0 \geq 0$ (True)

Therefore, at $n \geq 3$ ($n \geq n_o$) for K = 7, the inequality becomes true and so the Big *O* notation will be ***O*(n^3)**

b. To which big *O* set do the entire program belong?

Since, it is sequentially, $T(n) = t_1 + t_2$

$T(n) = 5n^2 + 16 + 6n^3 + n^2 + 18$

$T(n) = 6n^3 + 6n^2 + 34$

For T, using *Method 1* that is $6n^3$ and by removing 6, we get ***O*(n³)**

Using *Method 2,*

For T,

$Kn^3 \geq 6n^3 + 6n^2 + 34$ $(\because \text{after } n \geq n_0)$

Let K = 7,

$7n^3 \geq 6n^3 + 6n^2 + 34$

$7n^3 - 6n^3 - 6n^2 - 34 \geq 0$

$n^3 - 6n^2 - 34 \geq 0$

Let n = 1,

$-39 \geq 0$ (False)

Let n = 2,

$-50 \geq 0$ (False)

Let n = 3,

$-61 \geq 0$ (False)

Let n = 4,

$-66 \geq 0$ (False)

Let $n = 5$,

$-59 \geq 0$ (False)

Let $n = 6$,

$-34 \geq 0$ (False)

Let $n = 7$,

$15 \geq 0$ (True)

Therefore, at $n \geq 7$ ($n \geq n_o$) for $K = 7$, the inequality becomes true and so the Big *O* notation will be ***O*(n^3)**

c. **Prove that if $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$, then $(f_1 + f_2) \in O(\max\{g_1, g_2\})$.**

Here $g_1 \geq f_1$ and $g_2 \geq f_2$ *(Reflexive property of Asymptotic Notation)*

In other words,

Given is f_1 and f_2
Then $f_1(n) \leq O(f_1(n))$ and $f_2(n) \leq O(f_2(n))$

To prove this, I will use 1(b) as an example:

Let f_1 and f_2 here be t_1 and t_2 from the previous question respectively.

Let, $(f_1 = t_1) \leq g_1$
Let, $(f_2 = t_2) \leq g_2$

In 1(b) we added the two functions and we got a new function which we named T. With T we followed the same process as we did in 1(a) by finding the fastest growing term and removing its coefficient as well as *Method 2.*

The point to note is that the Big *O* for t_2 and T remained the same. The fastest growing term in them overpowers any other term and both happened to be n^3

The g_1 of the previous question is n^2 and the g_2 of the previous question is n^3. Since the g_2 term is the fastest in T, i.e. the max from the two functions, it stayed as the Big *O*.

$t_1 \in O(n^2)$ and $t_2 \in O(n^3)$

so, $(t_1 + t_2) \in O(\max\{n^2, n^3\}) = O(n^3)$.

$t_2 = \mathbf{6n^3} + n^2 + 18$ and $T = \mathbf{6n^3} + 6n^2 + 34 = \boldsymbol{O(n^3)}$

It's similar to how x^2 and nx^2 have the same Big *O* where $n \geq n_o$ and n_o is N^+ (*Method 2)*

In this question, g_1 and g_2 represents the Big *O* of their respective function f_1 and f_2. When we add these functions, similar to 1(b), we take the highest degree from g_1 and g_2. The result will be the highest or the max of g_1 and g_2.

Therefore, $(f_1 + f_2) \in O(\max\{g_1, g_2\})$.

Problem 2.2: proof by induction

Prove the following holds for n≥1

$$1^2 + 3^2 + 5^2 + \ldots (2n - 1)^2 = \sum_{k=1}^{n}(2k-1)^2 = \frac{2n(2n-1)(2n+1)}{6}$$

Let the proposition $P(n) = \sum_{k=1}^{n}(2k-1)^2 = \frac{2n(2n-1)(2n+1)}{6}$ be true.

Step 1: Prove for n = 1

$1^2 = \frac{2.1(2.1-1)(2.1+1)}{6}$

$1^2 = \frac{2(1)(3)}{6}$

$1 = 1$

Therefore, P(n) holds true for n = 1

Step 2: Assume n = k

$1^2 + 3^2 + 5^2 + \ldots (2k - 1)^2 = \frac{2k(2k-1)(2k+1)}{6}$ is true

Step 3: If n = k + 1, show P(k+1) holds true

$$1^2 + 3^2 + 5^2 + \ldots (2k - 1)^2 + (2k + 1)^2 = \frac{(2(k+1))(2(k+1)-1)(2(k+1)+1)}{6}$$

$$1^2 + 3^2 + 5^2 + \ldots (2k - 1)^2 + (2k + 1)^2 = \frac{(2k+2)(2k+1)(2k+3)}{6}$$

Taking LHS,

$1^2 + 3^2 + 5^2 + \ldots (2k - 1)^2 + (2k + 1)^2$

$\frac{2k(2k-1)(2k+1)}{6} + (2k+1)^2$ (∵ Using Step 2)

$\frac{2k(2k-1)(2k+1) + 6(2k+1)^2}{6}$

$\frac{(2k+1)\{2k(2k-1) + 6(2k+1)\}}{6}$

$\frac{(2k+1)(4k^2 + 10k + 6)}{6}$

$\frac{(2k+1)(2k+2)(2k+3)}{6}$

$\frac{(2k+2)(2k+1)(2k+3)}{6}$

= RHS

Step 4: Since the proposition also holds true for n = k + 1, the given equation holds true for all n≥1.

Problem 2.3: list comprehensions (haskell)

a. All positive factors of a number

To do this problem,

For Compiler:

```
main = print $ factors 210

factors f = [x | x <- [1..f], f `mod` x == 0]
```

While writing this code in a compiler, the 2nd line alone didn't work but, in the interpreter, it did. So, I had to add the main line for it to compile. To change the 210 to any other number, I just change it in the program and then recompile for it to show the factors for some other number.

For Interpreter:

```
>factors f = [x | x <- [1..f], f `mod` x == 0]
>factors 210
```

Adding the 2nd line alone in the interpreter works. In the next line we can write "factors 210" to get all the positive factors for 210 or factors *insert any N^+

b. All Pythagorean triads before 100 with no repetitions

To do this problem,

For Compiler:

```
main = print $ pyth 100

pyth :: Int -> [(Int, Int, Int)]

pyth x = [(a,b,c)|a<-[1..x], b<-[1..a], c<-[1..x], (a^2)+ (b^2) == (c^2)]
```

Again, writing the second and the third line alone didn't work and had to add the main line. The second line says that '*pyth'* is a function and its type signature is Integer. The third line basically calls the function and then finds all the Pythagorean triads before 100.

Repetitions are caused because for the correct value of 'c', one form has 'a' greater than 'b' and in the other form 'b' greater than 'a', In order to not have repetitions in 'a' and 'b', I limited 'b' to **1 to 'a'** so that 'b' always remains below a hence never has a chance to duplicate.

For Interpreter:

```
>pyth x = [(a,b,c)|a<-[1..x], b<-[1..a], c<-[1..x], (a^2)+ (b^2) == (c^2)]
>pyth 100
```

Adding the 3rd line alone in the interpreter works. In the next line we can write "pyth 100" to get all the non-repeating Pythagorean triads from 0 to 100.