









**TAYLOR'S UNIVERSITY**

Wisdom • Integrity • Excellence

**Bachelor of Computer Science (Hons) Bachelor of Software Engineering (Hons)  
Bachelor of Information Technology (Hons)**

**Module Code :** ITS60504 (August 2022)

**Module Name :** Data Structures and Algorithms

Assignment No./Title	Assignment Task 2 Part 2 (30%)
Course Tutor/Lecturer	Mr. Sham Shul Shukri Mat
Submission Date	22 Nov 2022
<b>Student Name, ID and Signature</b>	
1. Lohan Ramachandran 0354536	
2. Wang Eetian Ethan 0354393	
3. Aaron Goh Zheng Xun 0354994	
4. Samin Yasar 0354539	
5. Muhammad Hanif Zulfikri 0352833	
6. Khor Jia Wen 0354908	

**Declaration** *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

*Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.*

<b>Marks/Grade:</b>	<b>Evaluated by:</b>
<b>Evaluator's Comments:</b>	

## Part 1 - Algorithm Analysis

### Segment A

$$3n + 3n^2 + 1 + 4 + 5 + 20n$$

$$\text{Dominant Term} = 3n^2$$

$$\text{Big O Notation} = O(n^2)$$

By simplifying the equation, we will end up with the quadratic equation  $3n^2 + 23n + 10$ . It has three main parts, which is quadratic function, linear function, and also a constant.  $3n^2$  is the dominant term because quadratic terms have the fastest growth rate compared to the other two terms. To get the Big O Notation, we removed the constant value to get the value which is  $O(n^2)$ .

### Segment B

$$8 + n + 2\log_3 n$$

$$\text{Dominant Term} = n$$

$$\text{Big O Notation} = O(n)$$

By simplifying the question, we will end up with the equation  $2\log_3 n + n + 8$ . From this equation, the linear function has the quickest growth rate compared to the other functions. This also means that  $n$  has the worst time complexity. Therefore,  $O(n)$  is the Big O Notation for the function.

### Segment C

$$\log_5 5n + \log_3 2n + \log_2 n$$

$$\text{Dominant term: } \log_2 n$$

$$\text{Big O: } O(\log_2 n)$$

Big O ranking from fastest to slowest:  $O(\log_5 n) > O(\log_3 n) > O(\log_2 n)$

While all of them are a logarithmic time complexity, the logarithmic base is different from each other. The higher the logarithmic base is the smaller the result is, therefore the Big O is  $O(\log_2 n)$  as it has a logarithmic base of 2.

#### Segment D

$$8 + n \log_2 8 + 3n$$

Dominant Term =  $6n$

Big O Notation =  $O(n)$

By simplifying the question, we will end up with the equation  $8 + 3n + 3n$  ( $\log_2 8 = 3$ ) and then  $8 + 6n$ .

Referring to the general ranking of Big O's,  $O(1)$  has the best time complexity and  $O(n!)$  has the worst. Hence  $6n$  is chosen as the dominant term and  $O(n)$  is the Big O Notation for the function.

#### Segment E

$$3n + n \log_3 n + \log_2 n + 20$$

Dominant term:  $n \log_3 n$

Big O:  $O(n \log_3 n)$

Big O ranking from fastest to slowest:  $O(1) > O(\log_2 n) > O(n \log_3 n) > O(n)$

$O(n \log_3 n)$  is the big O due to the fact that it is a multiplication of linear and logarithmic time complexity. It is higher than  $O(n)$  as it is multiplied by  $\log_3 n$  who is also dependent on  $n$ .

### Segment F

$$n \log 2n + 7n^2 + n^4 + 2n^3$$

Dominant Term:  $n^4$

Big O Notation:  $O(n^4)$

By referring to the general ranking of the Big O's, the quartic function ( $n^4$ ) has the slowest time complexity compared to the other functions. Therefore,  $n^4$  is selected as the dominant term and  $O(n^4)$  will be the Big O Notation.

### Segment G:

Simplifying:

$$2n(n^4 + \log 2n + n) + n^2$$

$$2n^5 + 2n \log 2n + 2n^2 + n^2$$

$$= 2n^5 + 2n \log 2n + 3n^2$$

Dominant term:

$$2n^5$$

Big O complexity:

$$O(n^5)$$

Reasoning:

$2n^5$  was the dominant term because  $n$  is to the power of 5 which has the largest growth rate compared to  $2n \log 2n$  and  $3n^2$

Big O ranking:

$2n^5$  results in a Big O of  $O(n^5)$

$3n^2$  results in a Big O of  $O(n^2)$

$-2n \log_2 n$  results in a Big O of  $O(n \log n)$

### Segment H

$$\begin{aligned} & n(n^4 + n) + \log_2 n + 6n + 3^n \\ &= n^5 + n^2 + \log_2 n + 6n + 3^n \end{aligned}$$

Dominant term:  $3^n$

Big O:  $O(3^n)$

Big O ranking from fastest to slowest:  $O(\log_2 n) > O(n) > O(n^2) > O(n^5) > O(3^n)$

Referencing the general ranking of big O, It is noted that  $O(3^n)$  is the highest due to it being an exponential time complexity, behind  $O(n!)$ . Therefore,  $O(3^n)$  is the big O.

### Segment I

```
1 . void process(int n) {  
2 .     int tot = 0;  
3 .     for (int a = 1; a < n; a++) {  
4 .         n = n/5;  
5 .         tot = tot + n;  
6 .     }  
7 .     for (int b = 1; b < n; b++)  
8 .         tot = tot + n;  
9 .     for (int c = 1; c < 10; c++)  
10 .         tot = tot + n;  
11 . }
```

Full expression:

$$\log_5 n + n + 10$$

Big O:  $O(n)$

Although the first loop has  $n$  as the limit of the length and divided by 5 every loop, thus making a logarithmic time complexity of  $O(\log_5 n)$ , the loop after uses  $n$  as the limit of the length

without any changes, therefore making it a linear time complexity of  $O(n)$ . Because  $O(n)$  is more significant than  $O(\log_5 n)$ , therefore the big O is  $O(n)$ .

### Segment J

```
void process(int n) {  
    int tot = 0;  
    for (int c = 1; c < n; c++)          n +  
        tot = tot + n;  
    for (int b = 1; b < n; b++)          n *  
        for (int c = 1; c < n; c++)      n *  
            for (int d = 1; d < n; d++)  n  
                tot = tot + b + c + d;  
}
```

Full Expression

$$n + n^3 = O(n^3)$$

Big O Complexity

$$O(n) < O(n^3)$$

## Part 2 - Algorithm Development

### Segment A - a

```
void displayAa(int n){
    int tot = 0;
    int m = n;

    //irritates from 1 to n
    for(int a = 1; a <= m; a++){
        tot = tot + a;

        //irritates from 1 to  $\log_3 n$ 
        for(int b = 1; b <= n; b++){
            n = n/3;
            tot = tot + n;
        }
    }

    //irritates from 1 to n
    for(int c = 1; c <= m; c++){

        //irritates from 1 to n
        for(int d = 1; d <= m; d++){
            tot = tot + c + d;
        }
    }
}
```

## Segment B - a

```
void displayBa(int n){
    int tot = 0;

    //irritates from 1 to 2
    for(int a = 1; a <= 2; a++){
        tot = tot + a;

        //irritates from 1 to n
        for(int b = 1; b <= n; b++){
            tot = tot + a + b;
        }
    }

    //irritates from 1 to n
    for(int c = 1; c <= n; c++){

        //irritates from 1 to n
        for(int d = 1; d <= n; d++){

            //irritates from 1 to n
            for(int e = 1; e <= n; e++){

                //irritates from 1 to n
                for(int f = 1; f <= n; f++){
                    tot = tot + c + d + e + f;
                }
            }
        }
    }
}
```



```

//irritates from 1 to n
for(int g = 1; g <= n; g++){

    //irritates from 1 to n
    for(int h = 1; h <= n; h++){

        //irritates from 1 to n
        for(int i = 1; i <= n; i++){

            //irritates from 1 to n
            for(int j = 1; j <= n; j++){

                //irritates from 1 to n
                for(int k = 1; k <= n; k++){
                    tot = tot + g + h + i + j + k;
                }
            }
        }
    }
}

```

### Segment C - a

```
void displayCa(int n){
    int tot = 0;

    //irritates from 1 to n
    for(int a = 1; a <= n; a++){
        tot = tot + a;
    }

    //irritates from 1 to 3
    for(int b = 1; b <= 3; b++){
        tot = tot + b;
    }

    //irritates from 1 to 3
    for(int c = 1; c <= 3; c++){

        //irritates from 1 to n
        for(int d = 1; d <= n; d++){

            //irritates from 1 to n
            for(int e = 1; e <= n; e++){

                //irritates from 1 to n
                for(int f = 1; f <= n; f++){
                    tot = tot + c + d + e + f;
                }
            }
        }
    }
}
```

Segment D - a

```
void displayDa(int n){
    int tot = 0;
    int m = n;

    //irritates from 1 to  $\log_2 n$ 
    for(int a=1; a < n; a++){
        n = n/2;
        tot = tot + n;
    }

    //irritates from 1 to 2
    for(int b = 1; b <= 2; b++){

        //irritates from 1 to n
        for(int c = 1; c <= m; c++){
            tot = tot + c + d;
        }
    }
}
```