# Week 03 - Lecture 1 Slides

# Dr. Yeganeh Bahoo

Created: 2023-09-24 Sun 21:05

## Table of Contents



## Lecture 1: Analysis of Algorithms (cont.)

Learning objectives: By the end of this lecture you should be able to

1. Use “Big-O” to describe program execution time

### $O(T(n))$ and $T(n)$: assignment statement

$$(\texttt{SETF}\ \ var\ \ form) \qquad \text{or} \qquad (\texttt{:=}\ \ var\ \ form)$$

Example:

```
(setf x (+ y b))          ; Or using RUTIL's := macro     (:= x (+ y b))
```

As long as the operations in *form* do not involve loops or recursive functions:

- Running time: $T(n) = c$, where $c$ is a constant, e.g., 1. Therefore, for our purposes $T(n) = 1$
- Order of magnitude: $O(1)$

## $O(T(n))$ and $T(n)$: LET and LET* form

$$(\texttt{LET}\ \ varInits\ \ form^*) \qquad \text{or} \qquad (\texttt{LET*}\ \ varInits\ \ form^*)$$

As long as the operations in $varInits$ do not involve loops or recursive functions: Example:

- we will assume the running time of each variable initialization is the running time of one assignment statement
- the running time of $varInits$, denoted $T(varInits)$ is the summation of the running times of each ot its variable initializations
- running time for the LET statement: $T(n) = T(varInits) + T(form^*)$

Example:

```
(let ((a 5) (b 6) (c 10) (d 0)  (e 40)
      (k 1) (x 1) (y 1) (v 1) (w 1) (z 1))
  (:= a (* (/ b c e) (+ d k x y v w z))))
```

- Running time: $T(n) = 11 + 1 = 12$
- Order of magnitude: $O(1)$

## $O(T(n))$ and $T(n)$: loops

$$(\texttt{DOTIMES}\ \ VarTestRes\ \ bodyForm^*) \quad \text{or} \quad (\texttt{DO}\ \ VarTestRes\ \ bodyForm^*)$$

As long as the variable initializations, variable update operations, test, and result form computation in $VarTestRes$ do not involve loops or recursive functions:

- We will ignore the computational time taken by DO and DOTIMES to carry out those operations.
    - We will only consider the running time of $bodyForm^*$

Example: Simple loop

```
(dotimes (i n)        ; Repeats n times.
  (:= x (+ i b))
  (:= y (* x 2)))
```

- Running time: $\sum_{i=0}^{n-1}(1+1) = 2n$
- Order of magnitude is therefore $O(n)$

## $O(T(n))$ and $T(n)$: loops (cont.)

```
(dotimes (i n)        ; Repeats n times.
  (:= x (+ i b))      ; takes constant time, say, 1 time unit
  (dotimes (j n)      ; Repeats n times
    (:= y (+ a b))))  ; takes constant time, say, 1 time unit
```

- Running time: $\sum_{i=0}^{n-1}(1 + \underbrace{\sum_{j=0}^{n-1} 1}_{inner\ loop}) = \sum_{i=0}^{n-1}(1+n) = n + n^2$
- Order of magnitude is therefore $O(n^2)$

## Obtaining $O(T(n))$ and $T(n)$: another example

```
(let ((a 5) (b 6) (c 10) (d 0)
      (k 1) (x 1) (y 1) (v 1) (w 1) (z 1)) ; each initialization takes constant time
  (dotimes (i n)
    (dotimes (j n)
      (:= z (* i i))  ; each assignment statement takes some constant time
      (:= y (/ j j))
      (:= z (+ i j))))
  (dotimes (k n)
```

```
    (:= w (+ (* a k) 45)) ; each assignment statement takes some constant time
    (:= v (* b b)))
  (:= d 33))
```

- Running time:

$$T(n) = 10 + (\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 3) + (\sum_{k=0}^{n-1} 2) + 1 = 3n^2 + 2n + 11$$

- Order of magnitute: $O(n^2)$

## Question

- Complete the blanks in the expressions below that provide the execution time $T(n)$ and Big-O performance (aka "order of magnitude/growth") for the following code fragment:

  ```
  (dotimes (i n) ;
    (dotimes (j n)
      (:= k (+ 2 2))))
  (dotimes (i n)
    (:= k (+ 2 2)))
  ```

  $T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \cdots + \sum_{i=0}^{n-1} \cdots = \cdots + n$

  $O(\cdots)$

### Solution

```
(dotimes (i n)
  (dotimes (j n)
    (:= k (+ 2 2))))
(dotimes (i n)
  (:= k (+ 2 2)))
```

$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1 + \sum_{i=0}^{n-1} 1 = n^2 + n$

$O(n^2)$

## Another example

Let's obtain $T(n)$ and Big-O for the code fragment below:

```
(:= k 0)
(do ((i 1 (* i 2)))   ; Note: increment of i is (setf i (* i 2))
    ((>= i n))
  (:= k (+ K (+ 2 2))))
```

So we rush to get $T(n)$: $T(n) = 1 + \sum_{i=1}^{n}$?????

- The sigma ($\Sigma$) notation assumes the incremet is by ones
- $i$ is not being incremented by ones!
- $i$ does not grow linearly, i.e., 1, 2, 3, 4, 5, .... It grows expotentially: 1, 2, 4, 8, 16, ...

To calculate $T(n)$ using a summation ($\Sigma$) we need to transform the increment of $i$ in increment by ones

## Example (Cont.)

- Notice that $i$ grows exponentially: 1, 2, 4, 8, ...

  ```
  (:= k 0)
  (do ((i 1 (* i 2)))   ; Note: increment of i is (setf i (* i 2))
      ((>= i n))
    (:= k (+ K (+ 2 2))))
  ```

- If we make $i = 2^p$, with $p$ starting from 0, incrementing $p$ by 1. When $i = n, p = \log n$ (log n on base 2), then the loop becomes:

  ```
  (:= k 0)
  (do ((p 0 (1+ p)))        ; ignore computation time
      ((>= p (log n 2)))    ; igonore computation time
    (:= k (+ k (+ 2 2)))))  ; 1 unit of time
  ```

Now we are ok. The increment of the variable $p$ that control the loop is by ones.

Therefore, $T(n)$ and Big-O for the revised code are

$$T(n) = 1 + \sum_{p=0}^{\log n - 1} 1 = 1 + \log n$$

$$O(\log n)$$

## Question

What is $T(n)$ and Big-O for the code fragment below?

```
(dotimes (i n)
  (do ((j 1 (* j 2))) ; ignore computation time
      ((>= j n))      ; ignore computation time
      (:= a (+ b c))) ; 1 unit of time
```

$$T(n) = \sum_{i=0}^{\cdots}(\sum_{p=0}^{\cdots} 1) = (\sum_{i=0}^{\cdots} 1 \ldots n) = 1 \cdots \log n$$

$$O(\cdots \log n)$$

### Solution

```
(dotimes (i n)
  (do ((j 1 (* j 2)))
      ((>= j n))
      (:= a (+ b c)))
```

$$T(n) = \sum_{i=0}^{n-1}(\sum_{p=0}^{\log n - 1} 1) = (\sum_{i=0}^{n-1} 1 \log n) = 1n \log n$$

$$O(n \log n)$$

## Homework

What is $T(n)$ and Big-O for function G?

```
(defun f (a b)
  (let ((acc 0))
```

```
    (dotimes (i a)
      (incf acc))))

(defun g(n)
  (dotimes (i n)
    (dotimes (j n (f i j))
      (setf x (+ i j)))))
```

**Solution**

$$T_g(n) = \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1}(1) + T_f(n))$$

$$T_f(n) = 1 + \sum_{i=0}^{a-1} 1 = 1 + a \quad \text{Substituting } T_f(n) \text{ in } T_g(n)$$

$$T_g(n) = \underbrace{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1}_{\sum_{i=0}^{n-1} n} + \underbrace{\sum_{i=0}^{n-1} 1}_{n} + \underbrace{\sum_{i=0}^{n-1} i}_{\frac{n(n-1)}{2}} \quad \text{Note: } a \text{ in function } f \text{ is } i \text{ in function } g$$

$$T_g(n) = \sum_{i=0}^{n-1} n + n + \frac{n(n-1)}{2} = n^2 + n + \frac{1}{2}(n^2 - n) \quad \therefore$$

$$\text{Running time of function G: } T_g(n) = n^2 + n + \frac{1}{2}(n^2 - n)$$

$$\text{Big-O: } O(n^2)$$