Discrete Structures

Fight for Survival

Dr. Mudassir Shabbir

August 27, 2026

Algorithm Analysis

Introduction to Algorithm Analysis

Algorithm Analysis:

- A step-by-step method for solving a problem.
- How do we know it's a "good" or an "efficient" algorithm?
- What does "good"' or "efficient" mean here?
- How should we analyze an algorithm?

Time Complexity	How much "time" does the algorithm take to return an output.
Space Complexity	How much "Space and Memory" resources are required to run an algorithm?
Performance	Does the algorithm return an "exact" solution or an "approximate" solution?

"Time and Space" complexity is most commonly known as "Computational Complexity".

Introduction to Algorithm Analysis

Algorithm Analysis:

- A step-by-step method for solving a problem.
- How do we know it's a "good" or an "efficient" algorithm?
- What does "good"' or "efficient" mean here?
- How should we analyze an algorithm?

Time Complexity	How much "time" does the algorithm take to return an output.
Space Complexity	How much "Space and Memory" resources are required to run an algorithm?
Performance	Does the algorithm return an "exact" solution or an "approximate" solution?

"Time and Space" complexity is most commonly known as "Computational Complexity".

Introduction to Algorithm Analysis

Algorithms may perform differently, that is more or less efficient in different situations

Time Complexity

- Best Case Scenario
- Worst Case Scenario
- Average Case Scenario

How can we formalize these ideas, and develop tools to quantify the above aspects for any algorithm?

Analyzing Algorithms: Time Complexity

We count the total number of "atomic operations" performed by the algorithm. Each atomic operation takes a unit time

The number of operations depends on the size of input, and so the time complexity is a function of the input size typically.

Analyzing Algorithms

Example

```
x = Fixed Number
For y=1 till n
   Compare x and A[y]
End

Total Operations: n
(Time Complexity)
```

Table: Algorithm to Search an element in a given list A.

Analyzing Algorithms

Example

For $x = 1$ till n
For $y = 1$ till n
  Compare x and y
End
End

Total Operations: n^2
(Time Complexity)

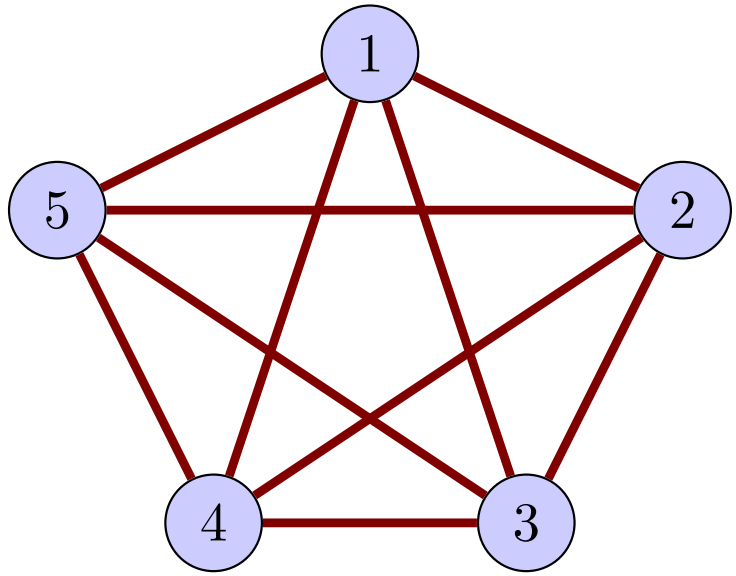
Table: Algorithm to Search duplicate elements.

Analyzing Algorithms

Problem: Given a set of n points, connect every pair of points by drawing a line between them.

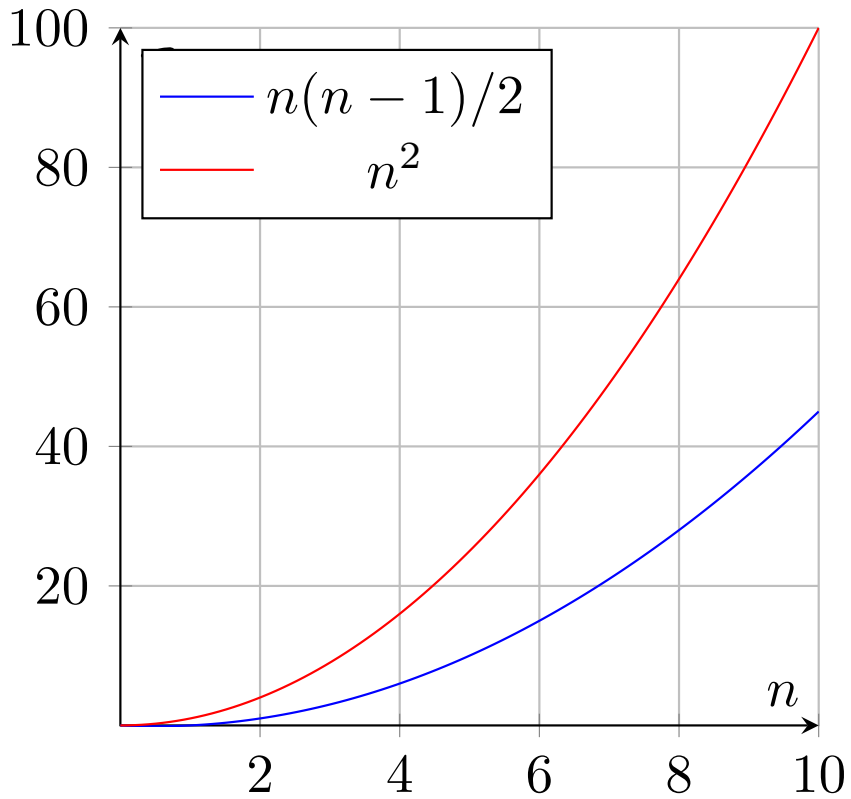
```
For i = 1 till n
   For j = i + 1 till n
    Draw a line between point i and j
   End
End
```

Total Operations: $n(n-1)/2$



Is it OK to say that the time complexity of the Algorithm is n^2 instead of $n(n-1)/2$?

Analyzing Algorithms



Time Complexity Analysis

No. of operations in the algorithm

"Exact" expression (in terms of input size n)	"Approximate" expression (in terms of input size n)
Could be hard to find	Simple but only give bounds

Time Complexity Analysis

Asymptotic Growth of the function f measures: how fast the output f(n) grows as the input n grows .

Our goal is to see if we can represent the limiting behavior of f(n) using some simpler functions that can give us a good idea of how fast the function grows with n.

We can go for the upper and lower bounds of f(n) under "certain conditions"

Announcements

- Two Makeup Classes on Monday May 20th at 2-5pm.

Extra Credit Problems

New Problem:

Find the number of quizzes with the same constraints as before but now each group contains three people.

The first solution to this problem will carry 30% credit to your final exam or is worth credit for two quizzes!

Extra Credit Problems

Consider a grid of $(x \times y \times z)$ points such that each point is colored either red, green, black, or blue. There always exists a 3D rectangle in the grid such that all eight of its vertices are the same colors. Find the minimum values of x, y, z.

Discrete Structures: Learning Objectives

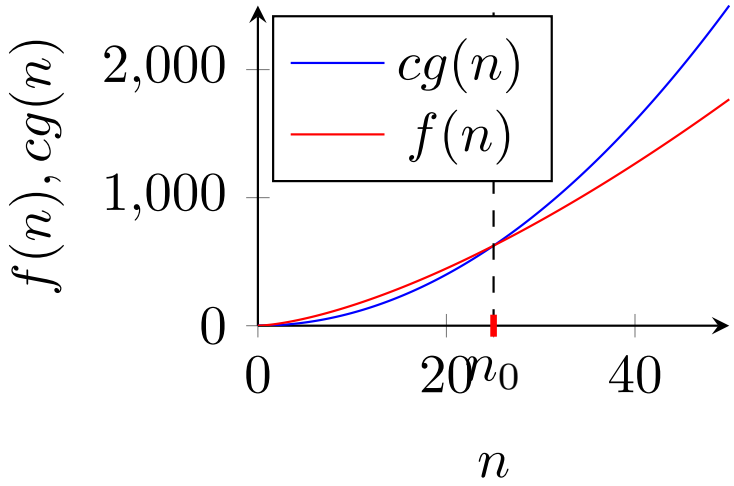
- ~~Understand and translate propositions with/without predicates and quantifiers~~
- ~~Be able to construct formal proofs using rules of Inference~~
- ~~Design and analyze relations on sets with/without reflexive, symmetric, and transitive properties~~
- ~~Distinguish and count types of different functions~~
- ~~Use Big Oh, Big Omega, Big Theta notations to analyze simple algorithms~~
- ~~Understand Recurrences~~
- ~~Construct Inductive Proofs~~
- Use rules of Counting
- Understand and solve problems on Graphs

Comparing Growth Rates: Big Oh

Big-oh is about finding the asymptotic upper bound of the function.

Formal definition of Big-Oh:
The notation $f = O(g)$ is read as "f is big-Oh of g".

$f(n) \leq cg(n)$ for all $n \geq n_o$



There is a constant c such that when f(n) and c g(n) are graphed, the graph of c g(n) will remain higher than f(n), as n gets large.

Big Oh: Example

$f(n) = 20 \log n + 10n$
$g(n) = n$,
Is $f(n) = O(g(n))$?

Big Oh: Example

$f(n) = 20 \log n + 10n$
$g(n) = n$,
Is $f(n) = O(g(n))$?

- Yes!
- Here $c = 15$, and $n_0 = 10$.
- Note that for $n \geq n_0$, $cg(n)$ is an upper bound of $f(n)$.

Comparing Growth Rates: Big Oh

Rules for Asymptotic Growth:

Let f, g, and h be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$:

1. If $f = O(h)$ and $g = O(h)$, then $f + g = O(h)$.
2. If $f = O(g)$ and c is a constant greater than 0, then $c.f = O(g)$.
3. If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.

Big Oh - Example

Comparing Growth Rates: Big Oh

Example:

Find the big Oh of an algorithm whose time complexity is the following:

$f(n) = 5n^3 + 16(n \log n) + 5.2^n$

$$5n^3 \text{ is } O(n^3) \text{ and } n^3 \text{ is } O(2^n)$$

$$16(n \log n) \text{ is } O(n \log n) \text{ and } n \log n \text{ is } O(2^n)$$

Hence,

$$f(n) = 5n^3 + 16(n \log n) + 5.2^n \text{ is } O(2^n)$$

Comparing Growth Rates: Big Oh

Summary:

We write: $f(n) = O(n)$
We say: f(n) is big O ("Oh") of n
Remember O(.) is solely an upper bound

Examples:

$200n = O(n)$
$n(n+1)/2 = O(n^2)$
$O(n \log n) + O(n^3) = O(n^3)$

Comparing Growth Rates: Big Ω

Big Omega Ω is about finding the asymptotic lower bound of the function.

Formal definition of Big Omega Ω:
The notation $f = \Omega(g)$ is read as "f is Omega of g".

$f(n) \geq cg(n)$ for all $n \geq n_0$, where c is positive constant.

Big Ω: Example

$f(n) = n^2 - 7n - 3$
$g(n) = n^2$

$$f(n) = \Omega(n^2)?$$

Big Ω: Example

$f(n) = n^2 - 7n - 3$
$g(n) = n^2$

$$f(n) = \Omega(n^2)?$$

- Here $c = 1/2$, and $n_0 = 20$.
- Note that for $n \geq n_0$, $cg(n)$ is an lower bound of $f(n)$.

Comparing Growth Rates: Big Ω

Rules for Asymptotic Growth:

Let f, g, and h be functions from $\mathbb{R}^+$ to $\mathbb{R}^+$:

- If $f = \Omega(h)$ AND $g = \Omega(h)$, then $f + g = \Omega(h)$
- If $f = \Omega(g)$ and c is a constant greater than 0, then $c.f = \Omega(g)$
- If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

Comparing Growth Rates: Big Ω

Summary:

We write: $f(n) = \Omega(g(n))$
We say: f(n) is big Ω of g(n)
Remember $\Omega(.)$ is simply a lower bound

Examples:
$200n = \Omega(n)$
$n^2 = \Omega(n)$
$n^2 = \Omega(n \log n)$

Comparing Growth Rates: Big Theta (Θ)

Big Theta (Θ):
The notation $f = \Theta(g)$ is read as "f is big theta of g".

Loosely speaking, $f = \Theta(g)$ means the algorithm is sandwiched between the same upper and lower bound. This gives us a precise measure of work.

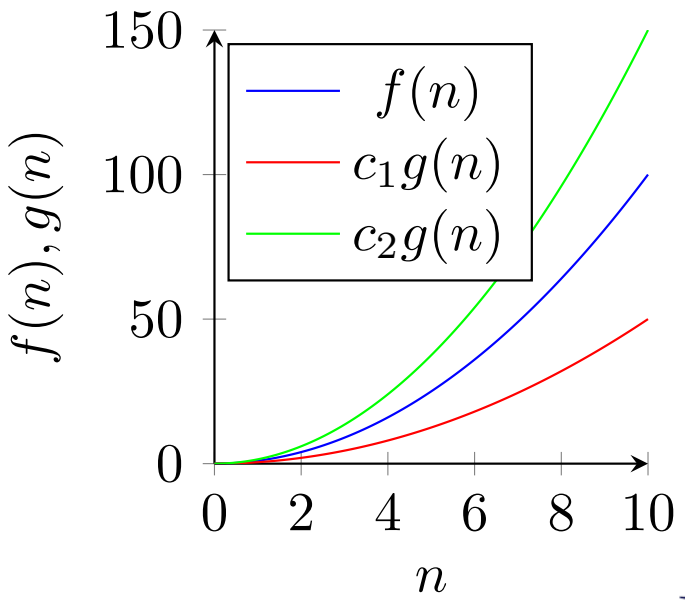
Comparing Growth Rates: Big Theta (Θ)

Let f and g be two functions from $\mathbb{Z}^+$ to $\mathbb{Z}^+$. Then $f = \Theta(g)$ if and only if

$$f = \Omega(g) \text{ and } f = O(g)$$

$f(n) = 0.5n^5 - 100n^3 + 3n - 1$
$g(n) = n^5$

$$f(n) = \Theta(n^5)$$



Comparing Growth Rates: Big Theta (Θ)

We write: $f(n) = \Theta(g(n))$

We say: $f(n)$ is "Big Θ of $g(n)$"

Translation: If two functions have a running time that differs by a constant, we say they have the same growth rate.

Comparing Growth Rates: Big Theta (Θ)

Example: Show $n(n+1)/2 = \Theta(n^2)$, $n \geq 0$

How about $c = 2$

$$n(n+1)/2 \leq c(n^2) \quad ???$$

$$n(n+1)/2 \leq 2(n^2)$$

$n(n+1)/2$ is $O(n^2)$

Comparing Growth Rates: Big Theta (Θ)

Example: Show $n(n+1)/2 = \Theta(n^2)$, $n \geq 0$

$$n(n+1)/2 \geq c(n^2) \quad \text{???}$$

How about $c = 1/2$

$$n(n+1)/2 \geq 1/2(n^2)$$

$n(n+1)/2$ is $\Omega(n^2)$

Comparing Growth Rates: Big Theta (Θ)

Example: Show $n(n+1)/2 = \Theta(n^2)$, $n \geq 0$

$$n(n+1)/2 \text{ is } O(n^2)$$

$$n(n+1)/2 \text{ is } \Omega(n^2)$$

Conclusion: $n(n+1)/2$ is $\Theta(n^2)$, which is a tight bound on the amount of work.

Comparing Growth Rates: Big Theta (Θ)

Example: Show $\log(n+1) = \Theta(\log n)$

Comparing Growth Rates: Big Theta (Θ)

Example: Show $\log(n+1) = \Theta(\log n)$

Approach: Again, we need to demonstrate two things to make our big theta case...

1. The function does **at most** $\log(n)$ work to say it is $O(\log n)$
2. The function does **at least** $\log(n)$ work to say it is $\Omega(\log n)$
3. Conclude the function does $\Theta(\log n)$ work, which is a tight bound.

Comparing Growth Rates: Big Theta (Θ)

Example: Show $\log(n+1) = \Theta(\log n)$

1. Show $\log(n+1)$ is $O(\log n)$

We know $n < n+1 < n^2$ for $n \geq 2$

So it follows that

$$\log(n) < \log(n+1) < \log(n^2) = 2\log(n) \text{ for } n \geq 2$$

Since $\log(n+1) < 2\log(n)$, we conclude that

$$\log(n+1) = O\log(n)$$

Comparing Growth Rates: Big Theta (Θ)

Example: Show $\log(n+1) = \Theta(\log n)$

2. Show $\log(n+1)$ is $\Omega(\log n)$

We observe $\log(n+1) > \log(n)$

So we can conclude it is $\Omega(\log n)$

3. Since $\log(n+1)$ is both $O(\log n)$ and $\Omega(\log n)$, we conclude that the given function

$$\log(n+1) = \Theta(\log n)$$

Comparing Growth Rates: Big Theta (Θ)

Order Of

If $f = \Theta(g)$, then f is said to be an order of g

Constant function

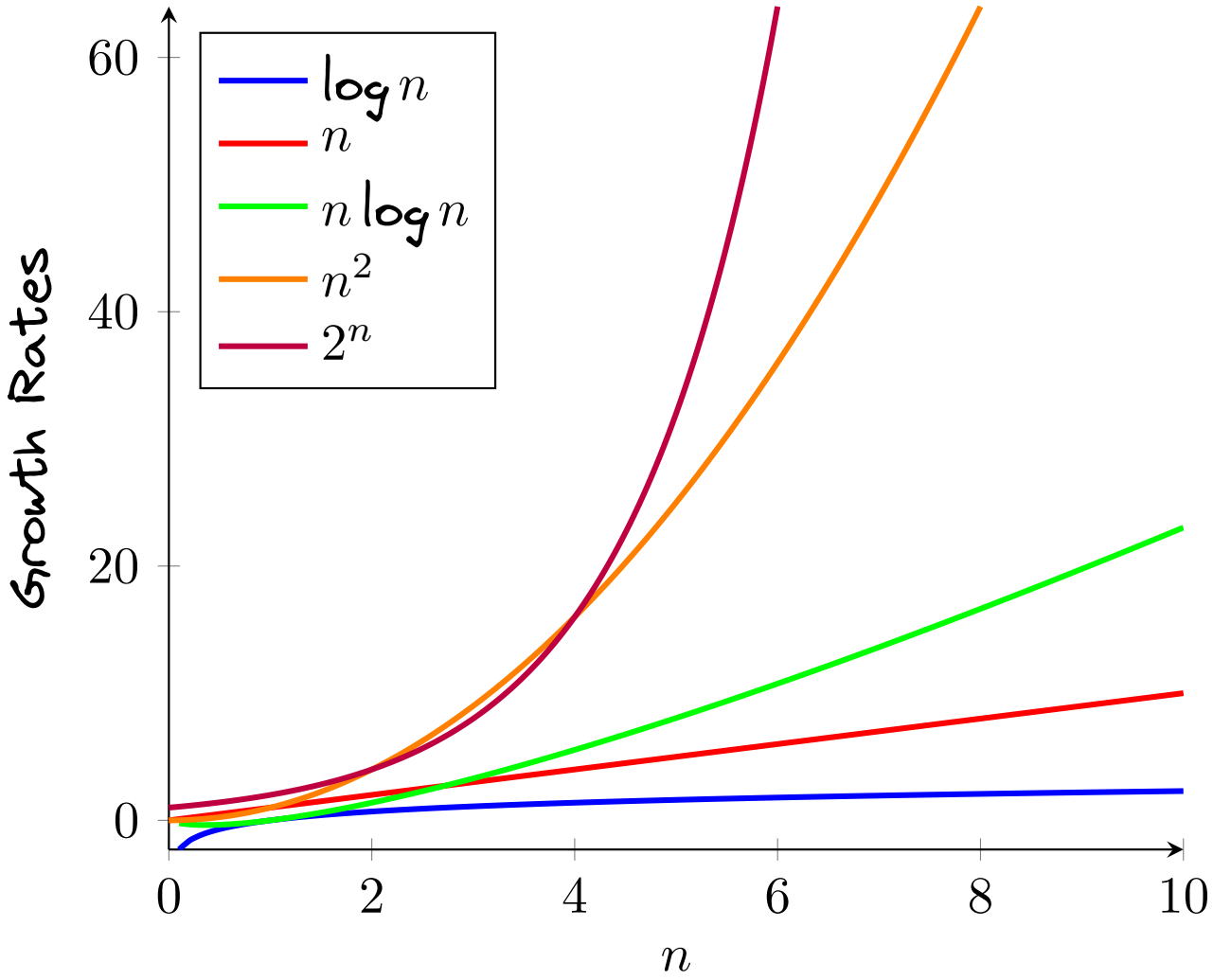
A function that does not depend on n at all is called a constant function.

Polynomial

A function f (n) is said to be polynomial if f (n) is $\Theta(n^k)$ for some constant $k \geq 0$

Name of Function	Function	Example Algorithm
Constant	$f(n) = c$	Accessing an array element
Logarithmic	$f(n) = \log n$	Binary search
Log-Logarithmic	$f(n) = \log \log n$	Smoothsort
Linear	$f(n) = n$	Simple search
Linearithmic	$f(n) = n \log n$	Efficient sorting algorithms
Quadratic	$f(n) = n^2$	Bubble sort
Cubic	$f(n) = n^3$	Matrix multiplication
Exponential	$f(n) = 2^n$	Algorithms with exhaustive search

Comparing Growth Rates



Comparing Growth Rates

Exercise (Round 1):

Indicate the "order of growth" relationships between the following expressions from lowest order to highest order. If two expressions have the same order, place them in a set together.

$$n!,\ n^2,\ \log(n),\ 3^n,\ n^3,\ n\log(n),\ \log(\log n),\ 2^n$$

Comparing Growth Rates

Exercise (Round 1):

Indicate the "order of growth" relationships between the following expressions from lowest order to highest order. If two expressions have the same order, place them in a set together.

$n!,\ n^2,\ \log(n),\ 3^n,\ n^3,\ n\log(n),\ \log(\log n),\ 2^n$

$\log(\log n),\ \log(n),\ n\log(n),\ n^2,\ n^3,\ 2^n,\ 3^n,\ n!$

Comparing Growth Rates

Exercise (Round 2):

Indicate the "order of growth" relationships between the following expressions from lowest order to highest order. If two expressions have the same order, place them in a set together

$$n^2,\ \log(n^2),\ (\log n)^2,\ 2n,\ 2^n,\ n^3,\ \log(n),\ n\log(n),\ \log(\log(n))$$

Comparing Growth Rates

Exercise (Round 2):

Indicate the "order of growth" relationships between the following expressions from lowest order to highest order. If two expressions have the same order, place them in a set together

$$n^2,\ \log(n^2),\ (\log n)^2,\ 2n,\ 2^n,\ n^3,\ \log(n),\ n\log(n),\ \log(\log(n))$$

$$\log(\log(n)),\ \{\log(n), \log(n^2)\},\ (\log n)^2,\ 2n,\ n\log(n),\ n^2,\ n^3,\ 2^n$$

Algorithm Analysis

Example: Algorithm to find the smallest in a sequence of numbers

Input: $a_1, a_2, a_3, \ldots, a_n$ Output: Smallest number in the sequence

$min := a_1$ — 1 Assignment Op

For $i = 2$ to n — loop iterated $n-1$ times

if $(a_i < min)$ — For loop tests i and increments i

$min := a_i$ — (2 Ops) 1 op for comparison + 1 op (in worst case) for assignment

End For

Return (min) — 1 op return

$f(n) =$ No. of Ops on a sequence of length n

$f(n) = (n-1)c + d = cn - c + d = \Theta(n)$

Algorithm Analysis

Example: Worst case Complexity- Find Maximum value of a function

Input: $a_1, a_2, a_3, \ldots, a_n$ Output: Largest M from the sequence

$max := M(a_i, a_j, a_k)$
For $i = 1$ to n
  For $j = 1$ to n
    For $k = 1$ to n
      $new := M(a_i, a_j, a_k)$
      if $(new > M(a_i, a_j, a_k))$
      $max := new$
    End For
  End For
End For

Return (max)

Algorithm Analysis

Example: Worst case Complexity- Find Maximum value of a function

Input: $a_1, a_2, a_3, \ldots, a_n$ Output: Largest M from the sequence

$max := M(a_i, a_j, a_k)$
For $i = 1$ to n
  For $j = 1$ to n
    For $k = 1$ to n
      $new := M(a_i, a_j, a_k)$
      if $(new > M(a_i, a_j, a_k))$
      $max := new$
    End For
  End For
End For

Return (max)

The worst-case time complexity of the algorithm is $O(n^3)$

Algorithm Analysis

Example: Worst case Complexity- Find Maximum value of a function

Input: $a_1, a_2, a_3, \ldots, a_n$ Output: Largest M from the sequence

$max := M(a_i, a_j, a_k)$
For $i = 1$ to n
  For $j = 1$ to n
    For $k = 1$ to n
      $new := M(a_i, a_j, a_k)$
      if $(new > M(a_i, a_j, a_k))$
      $max := new$
    End For
  End For
End For

Return (max)

The worst-case time complexity of the algorithm is $O(n^3)$

- O(1) to compute the value of M on any input
- At least one operation is performed in the innermost loop, so the time complexity of the algorithm is $\Omega(n^3)$
- The number of operations performed in the worst case is at most $cn^3 + 2$, which is $O(n^3)$.