Algorithms: Efficiency, Analysis, and Order

By: Aminul Islam

Based on Chapter 1 of Foundations of Algorithms and Chapter 2 of CLRS

Objectives

Objectives

- Analyze techniques for solving problems
- Define an algorithm
- Define Every-case, Worst-case, Average-case, and Best-case complexity analysis of algorithms
- Define growth rate of an algorithm as a function of input size
- Classify functions based on the growth rate
- Define growth rates: Big O, Theta, and Omega

What is an Algorithm?

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book
- An algorithm is a step by step approach to solve a problem

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book

a well defined task

- An algorithm is a step by step approach to solve a problem

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book

question

a well defined task

- An algorithm is a step by step approach to solve a problem

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book
- An algorithm is a step by step approach to solve a problem

question

a well defined task

- Several solutions (algorithms) to solve a problem

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book
- question
- a well defined task
- An algorithm is a step by step approach to solve a problem
- Several solutions (algorithms) to solve a problem
  - We like the fastest one!

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book
- An algorithm is a step by step approach to solve a problem

question

a well defined task

- Several solutions (algorithms) to solve a problem
  - We like the fastest one!
  - We like the efficient one (minimum CPU and memory)

What is an Algorithm?

- Consider the question in Gauss's story or searching a telephone book

  question

  a well defined task

- An algorithm is a step by step approach to solve a problem
- Several solutions (algorithms) to solve a problem
  - We like the fastest one!
  - We like the efficient one (minimum CPU and memory)
  - How to analyze algorithms in terms of time and space?

Definition of "Problem"

Definition of "Problem"

- A problem is <u>a well defined task</u> or <u>a question</u> to which we seek an answer

Definition of “Problem”

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?

Definition of “Problem”

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?
  - The answer is a number
  - The problem has one parameter: n

Definition of "Problem"

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?
  - The answer is a number
  - The problem has one parameter: n
- Example (Problem): Finding if number X exists in a sorted list S with n elements

Definition of "Problem"

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?
  - The answer is a number
  - The problem has one parameter: n
- Example (Problem): Finding if number X exists in a sorted list S with n elements
  - The answer is either yes/no or 0/index
  - The problem has three parameters: S, X, n

Definition of "Problem"

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?
  - The answer is a number
  - The problem has one parameter: n
- Example (Problem): Finding if number X exists in a sorted list S with n elements
  - The answer is either yes/no or 0/index
  - The problem has three parameters: S, X, n
- Example (Problem): Sorting a list S with n elements in ascending order

Definition of "Problem"

- A problem is a well defined task or a question to which we seek an answer
- Example (Problem): What is the summation of 1 to n?
  - The answer is a number
  - The problem has one parameter: n
- Example (Problem): Finding if number X exists in a sorted list S with n elements
  - The answer is either yes/no or 0/index
  - The problem has three parameters: S, X, n
- Example (Problem): Sorting a list S with n elements in ascending order
  - The answer is a list with n elements sorted
  - The problem has two parameters: S and n

Problem Instance

Problem Instance

- A problem with particular parameters is called a "problem instance"

Problem Instance

- A problem with particular parameters is called a "problem instance"
- Example (Problem): What is the summation of 1 to n?

Problem Instance

- A problem with particular parameters is called a "problem instance"
- Example (Problem): What is the summation of 1 to n?
- Example (Problem Instance): What is the summation of 1 to $n = 100$?

Problem Instance

- A problem with particular parameters is called a "problem instance"
- Example (Problem): What is the summation of 1 to n?
- Example (Problem Instance): What is the summation of 1 to $n = 100$?
- Example (Problem): Finding if number X exists in a sorted list S with n elements

Problem Instance

- A problem with particular parameters is called a "problem instance"
- Example (Problem): What is the summation of 1 to n?
- Example (Problem Instance): What is the summation of 1 to $n = 100$?
- Example (Problem): Finding if number X exists in a sorted list S with n elements
- Example (Problem Instance): Finding if number $X = 12$ exists in a sorted list $S = [2, 3, 5, 7, 30, 42]$ with $n = 6$ elements

Problem Instance

- A problem with particular parameters is called a “problem instance”
- Example (Problem): What is the summation of 1 to n?
- Example (Problem Instance): What is the summation of 1 to $n = 100$?
- Example (Problem): Finding if number X exists in a sorted list S with n elements
- Example (Problem Instance): Finding if number $X = 12$ exists in a sorted list $S = [2, 3, 5, 7, 30, 42]$ with $n = 6$ elements
- Example (Problem): Sorting a list S with n elements in ascending order

Problem Instance

- A problem with particular parameters is called a "problem instance"
- Example (Problem): What is the summation of 1 to n?
- Example (Problem Instance): What is the summation of 1 to $n = 100$?
- Example (Problem): Finding if number X exists in a sorted list S with n elements
- Example (Problem Instance): Finding if number $X = 12$ exists in a sorted list $S = [2, 3, 5, 7, 30, 42]$ with $n = 6$ elements
- Example (Problem): Sorting a list S with n elements in ascending order
- Example (Problem Instance): Sorting a list $S = [9, 3, 6, 3, 7]$ with $n = 5$ elements in ascending order

Human-based Solutions vs Algorithms

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]
  - Human mind can solve some problem instances very efficiently

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]
  - Human mind can solve some problem instances very efficiently
- Sort S=[4, 6, 12, 7, 43, 3, 7, 97, 23, 56, 754, 43, 245, . . .]

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]
  - Human mind can solve some problem instances very efficiently
- Sort S=[4, 6, 12, 7, 43, 3, 7, 97, 23, 56, 754, 43, 245, ...]
  - Our mind solution cannot cover all problem instances!

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]
  - Human mind can solve some problem instances very efficiently
- Sort S=[4, 6, 12, 7, 43, 3, 7, 97, 23, 56, 754, 43, 245, . . .]
  - Our mind solution cannot cover all problem instances!
  - Our mind solution cannot scale!

Human-based Solutions vs Algorithms

- Sort S=[3, 4, 2, 5, 1]
  - Human mind can solve some problem instances very efficiently
- Sort S=[4, 6, 12, 7, 43, 3, 7, 97, 23, 56, 754, 43, 245, ...]
  - Our mind solution cannot cover all problem instances!
  - Our mind solution cannot scale!
- But, an algorithm for a problem must be able to solve all instances of a problem!

Presenting an Algorithm

Presenting an Algorithm

- We can use natural languages

Presenting an Algorithm

- We can use natural languages
  - It is not accurate and sometime vague!

Presenting an Algorithm

- We can use natural languages
  - It is not accurate and sometime vague!
- We generally use pseudo-code to present algorithms

Presenting an Algorithm

- We can use natural languages
  - It is not accurate and sometime vague!
- We generally use pseudo-code to present algorithms
  - Algorithms are language independent

Presenting an Algorithm

- We can use natural languages
  - It is not accurate and sometime vague!
- We generally use pseudo-code to present algorithms
  - Algorithms are language independent
  - We will use C/C++ pseudo code just to present algorithms

Analyze the Time Efficiency of an Algorithm

Analyze the Time Efficiency of an Algorithm

- It is also known as time complexity or complexity of an algorithm

Analyze the Time Efficiency of an Algorithm

- It is also known as time complexity or complexity of an algorithm
  - Memory efficiency is doable but less common

Analyze the Time Efficiency of an Algorithm

- It is also known as time complexity or complexity of an algorithm
  - Memory efficiency is doable but less common
- To know the time of an algorithm, we should count the number of instructions executed on the CPU

Analyze the Time Efficiency of an Algorithm

- It is also known as time complexity or complexity of an algorithm
  - Memory efficiency is doable but less common
- To know the time of an algorithm, we should count the number of instructions executed on the CPU
- Lets do an example ...

Sequential vs Binary Search

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

$\uparrow$ low $\qquad$ $\uparrow$ high

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$$\begin{array}{llllllll} & & & \downarrow & & & & \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ \uparrow & & & & & & & \uparrow \\ \text{low} & & & & & & & \text{high} \end{array}$$

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$$\begin{array}{cccccccc} & & & \downarrow & & & & \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ & & & & \uparrow & & & \uparrow \\ & & & & \text{low} & & & \text{high} \end{array}$$

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$$\begin{array}{cccccccc} & & & \downarrow & & \downarrow & & \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ & & & & \uparrow & & & \uparrow \\ & & & & \text{low} & & & \text{high} \end{array}$$

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

low high

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$$\begin{array}{cccccccc} & & & \downarrow & & \downarrow & \downarrow & \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ & & & & & & \uparrow & \uparrow \\ & & & & & & \text{low} & \text{high} \end{array}$$

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$\downarrow$ $\downarrow$ $\downarrow$

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

$\uparrow \uparrow$

high

low

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have comparisons

search X and $X \notin S$

$$\begin{array}{cccccccc} & & & \downarrow & & \downarrow & \downarrow & \downarrow \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ & & & & & & & \uparrow\uparrow \\ & & & & & & & \text{high} \\ & & & & & & & \text{low} \end{array}$$

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search comparisons
  - If $n = 8$ in binary search we have 4 comparisons

search X and $X \notin S$

S[1] S[2] S[3] S[4] S[5] S[6] S[7] S[8]

high

low

Sequential vs Binary Search

- Assume:
  - A list (array) S with size n
  - Number $X \notin S$ that we are searching for in S
- Using linear (sequential) search n comparisons
- Using binary search $\lg_2 n + 1$ comparisons
  - If $n = 8$ in binary search we have 4 comparisons

search X and $X \notin S$

$$\begin{array}{cccccccc} & & & \downarrow & & \downarrow & \downarrow & \downarrow \\ S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] & S[8] \\ & & & & & & & \uparrow\,\uparrow \\ & & & & & & & \text{high} \\ & & & & & & & \text{low} \end{array}$$

How Number of Comparisons Can be Important!

How Number of Comparisons Can be Important!

List size	**Comparisons in Sequential**	**Comparisons in Binary**
128	128	8
$2^{10} = 1024$	1024	11
$2^{20} = 1{,}048{,}576$	1,048,576	21
$2^{30} = 4{,}294{,}967{,}296$	4,294,967,296	31

How Number of Comparisons Can be Important!

List size	**Comparisons in Sequential**	**Comparisons in Binary**
128	128	8
$2^{10} = 1024$	1024	11
$2^{20} = 1{,}048{,}576$	1,048,576	21
$2^{30} = 4{,}294{,}967{,}296$	4,294,967,296	31

- The number of comparisons is a function of "list size"

Analyze the Time Efficiency of an Algorithm (2)

Analyze the Time Efficiency of an Algorithm (2)

- We do not need to count all instructions/commands to analyze the efficiency of an algorithm

Analyze the Time Efficiency of an Algorithm (2)

- We do not need to count all instructions/commands to analyze the efficiency of an algorithm
- We find and count the number of times basic operations is executed for a given input size

Input Size of an algorithm

Input Size of an algorithm

- Some examples:

Problem	Input Size

Input Size of an algorithm

- Some examples:

Problem	**Input Size**
Search X in a list with n elements	The number of elements n in the list

Input Size of an algorithm

- Some examples:

Problem	Input Size
Search X in a list with n elements	The number of elements n in the list
Sort a list with n numbers	The number of elements n in the list

Input Size of an algorithm

- Some examples:

Problem	Input Size
Search X in a list with n elements	The number of elements n in the list
Sort a list with n numbers	The number of elements n in the list
compute $n!$	n

Input Size of an algorithm

- Some examples:

Problem	**Input Size**
Search X in a list with n elements	The number of elements n in the list
Sort a list with n numbers	The number of elements n in the list
compute $n!$	n
Multiply two matrices	The dimensions of the matrices

Input Size of an algorithm

- Some examples:

Problem	**Input Size**
Search X in a list with n elements	The number of elements n in the list
Sort a list with n numbers	The number of elements n in the list
compute $n!$	n
Multiply two matrices	The dimensions of the matrices

- Some algorithms are not dependent on the size of the input. Example ?

Input Size of an algorithm

- Some examples:

Problem	**Input Size**
Search X in a list with n elements	The number of elements n in the list
Sort a list with n numbers	The number of elements n in the list
compute $n!$	n
Multiply two matrices	The dimensions of the matrices

- Some algorithms are not dependent on the size of the input. Example ?
- Algorithms that depend on the size of the input are the ones that might cause problems

Basic Operation of an algorithm

Basic Operation of an algorithm

- We pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done and call this instruction or group of instructions the Basic Operation of an algorithm

Basic Operation of an algorithm

- We pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done and call this instruction or group of instructions the Basic Operation of an algorithm
- Some examples:

Problem	**Basic Operation**

Basic Operation of an algorithm

- We pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done and call this instruction or group of instructions the Basic Operation of an algorithm
- Some examples:

Problem	**Basic Operation**
Search X in a list	Comparison of X with an element in the list

Basic Operation of an algorithm

- We pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done and call this instruction or group of instructions the Basic Operation of an algorithm
- Some examples:

Problem	**Basic Operation**
Search X in a list	Comparison of X with an element in the list
Sort a list of numbers	Comparison of two list elements and moving elements in the list

Basic Operation of an algorithm

- We pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done and call this instruction or group of instructions the Basic Operation of an algorithm
- Some examples:

Problem	**Basic Operation**
Search X in a list	Comparison of X with an element in the list
Sort a list of numbers	Comparison of two list elements and moving elements in the list
Multiplying two matrices with real numbers	Multiplication of two real numbers

Four Types of Time Complexity

Four Types of Time Complexity

- Every-Case Time Complexity, $T(n)$
- Worst-Case Time Complexity, $W(n)$
- Average-Case Time Complexity, $A(n)$
- Best-Case Time Complexity, $B(n)$

Every-Case Time Complexity, $T(n)$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) =$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)
  - $T(n) =$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)
  - $T(n) = (n-1) + (n-2) + \cdots + 1$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)
  - $T(n) = (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)
  - $T(n) = (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$
- What is the $T(n)$ for Gauss's algorithm to sum 1 to n?

Every-Case Time Complexity, $T(n)$

- $T(n)$ is the number of times a basic operation is executed for a given input size n
- Example 1: Summing elements of a list of size n
  - Basic operation: addition
  - Input size: n (list size)
  - $T(n) = n$
- Example 2: Exchange sort of a list of size n
  - Basic operation: comparison of $S[i]$ with $S[j]$
  - Input size: n (list size)
  - $T(n) = (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$
- What is the $T(n)$ for Gauss's algorithm to sum 1 to n?
- What is $T(n)$ for matrix (n by n) multiplication?