Introduction

Data Structure

- A particular way of storing and organizing data in a computer so that it can be used efficiently by operations.

- They provide a means to manage large amounts of data efficiently, such as large databases.

- Data are simply values or set of values and Database is organized collection of data.

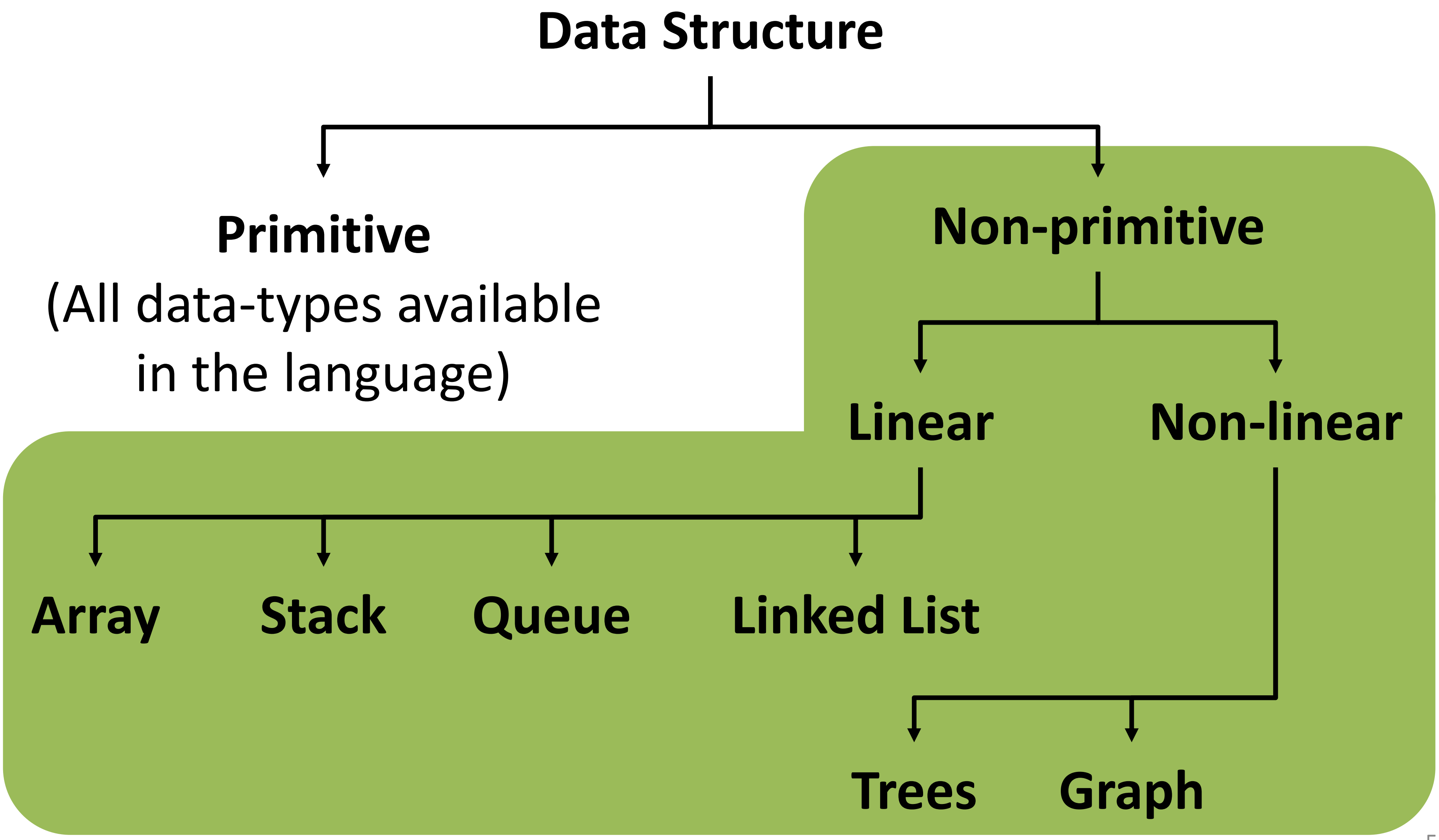
Abstract Data Type (ADT)

- Separate notions of **specification** and **implementation**.
- Realization of a data type as a software component, i.e. focus on **what** can be done with the data, not **how** it is done.
- The interface of the ADT is defined in terms of a type and a set of operations on that type. Each operation has a determined input as well as output.
- Implementation details are hidden from the user of the ADT and protected from outside access, a concept referred to as encapsulation.
- A **data structure** is the implementation for an ADT and are commonly deployed using classes in object-oriented languages.

Study of Data Structure Includes

- Logical description of data structure.
- Implementation of data structure.
- Quantitative analysis of data structure, this include amount of memory, processing time.

Classification of Data Structures



Selecting a Data Structure

- Select data structure first to solve a problem instead of writing complex program.
- Analyze the problem.
- Identify prominent operations and determine their constraints.
- Pick the data structure finally.
- Each data structure has its advantages as well as disadvantages.
  - It’s difficult to say that one data structure is better than another always.

Introduction to Algorithms

Algorithm

- A set of well defined instructions in sequence to solve a problem.
- Usually a high-level description of a procedure that manipulates well-defined input data to produce desired output data.
- Example: **Find the sum of two numbers**
  1. Take FIRST number as an input.
  2. Take SECOND number as an input.
  3. Add these two numbers.
  4. Output the result.

Contd…

Characteristics of a good algorithm

- **Definiteness:** Has clear and unambiguous steps.
- **Finiteness:** Should terminate.
- **Input/Output:** Has a defined set of inputs and outputs.
- **Effectiveness:** Should be effective and correct.

Contd…

Algorithm can be expressed as

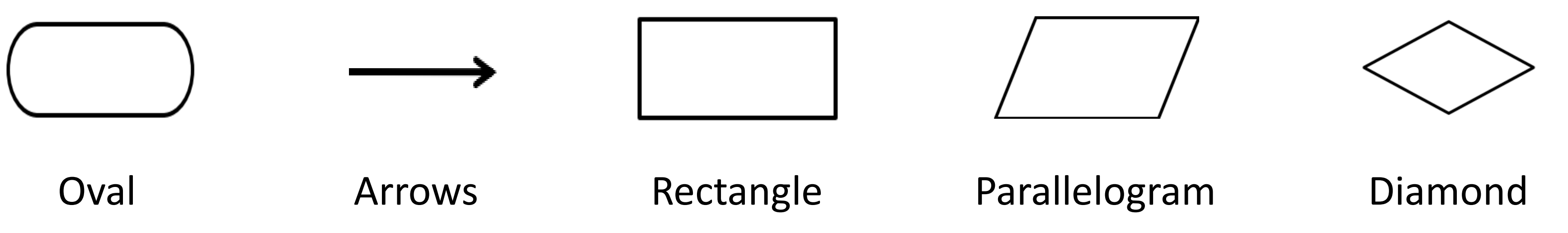
- Flowchart
- Pseudo-code

Flowchart

- Graphical/pictorial representation of an algorithm.
- Eases the task of writing high level programs with complex logic.

Flowchart Symbols

- Oval: Used at the Start and End of the flowchart.
- Arrows: Show the flow of control in the flowchart.
- Rectangle: Shows the processing step.
- Parallelogram: Represents the input taken from the user or to display the output to the user.
- Diamond: Represents the conditional flow of the steps.



Flowchart Constructs

- Sequence:

  Represents step-wise sequence of instructions to be followed for performing the desired task.

- Selection:

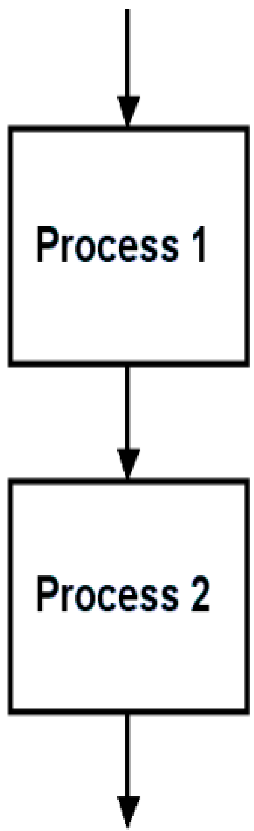
  Represents conditional flow of instructions. If true, one path is followed, otherwise the other.

- Iteration:

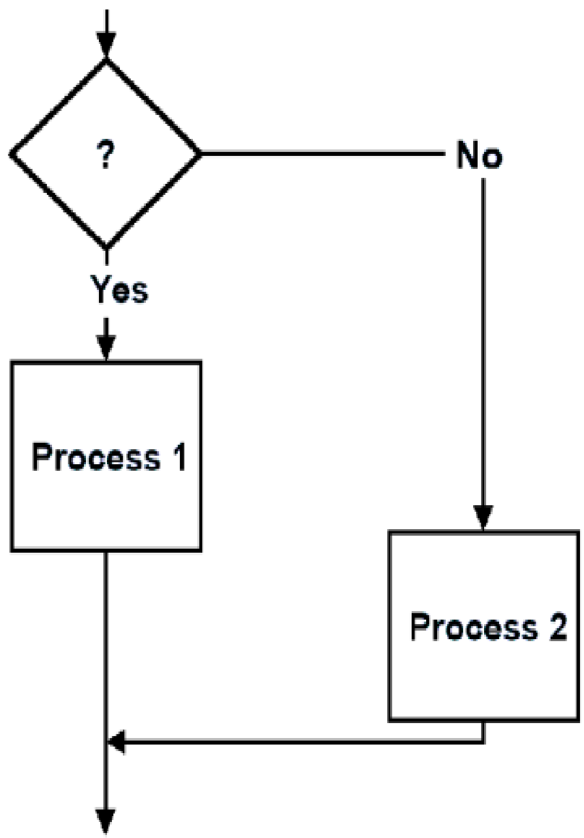
  Represents iterative flow of control. If true, same steps are repeated for some number of times, otherwise the loop terminates.

Contd…

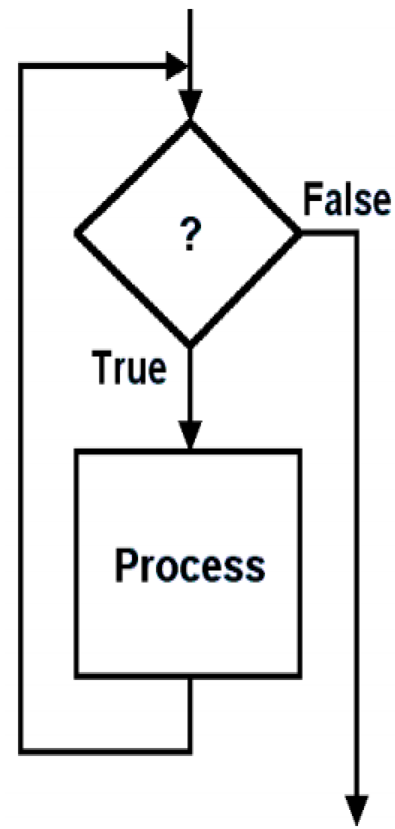
Sequence



Selection



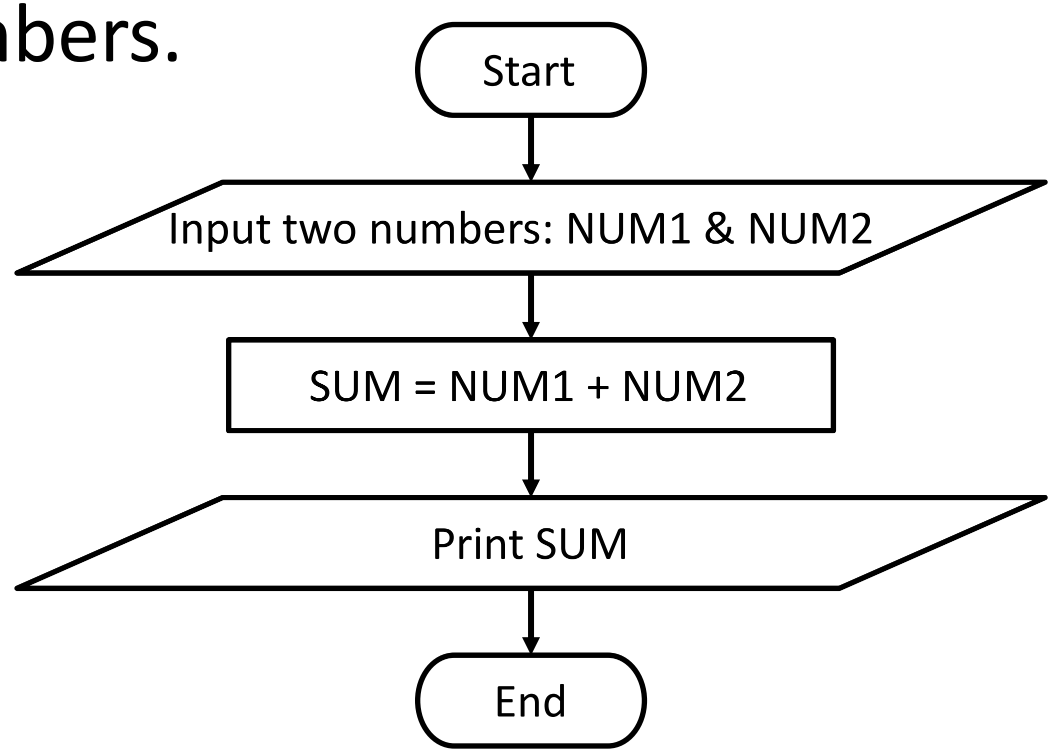
Iteration



Example

Calculate and print the SUM of two Numbers:

1. Take FIRST number as an input.
2. Take SECOND number as an input.
3. Add these two numbers.
4. Output the result.



Pseudocode

- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.
- Example:
  - Algorithm **sumOfTwoNumbers**(NUM1,NUM2)
  - Input: Two numbers **NUM1** and **NUM2**.
  - Output: The sum of two numbers.
    1. SUM = NUM1 + NUM2
    2. return SUM

Pseudocode Constructs

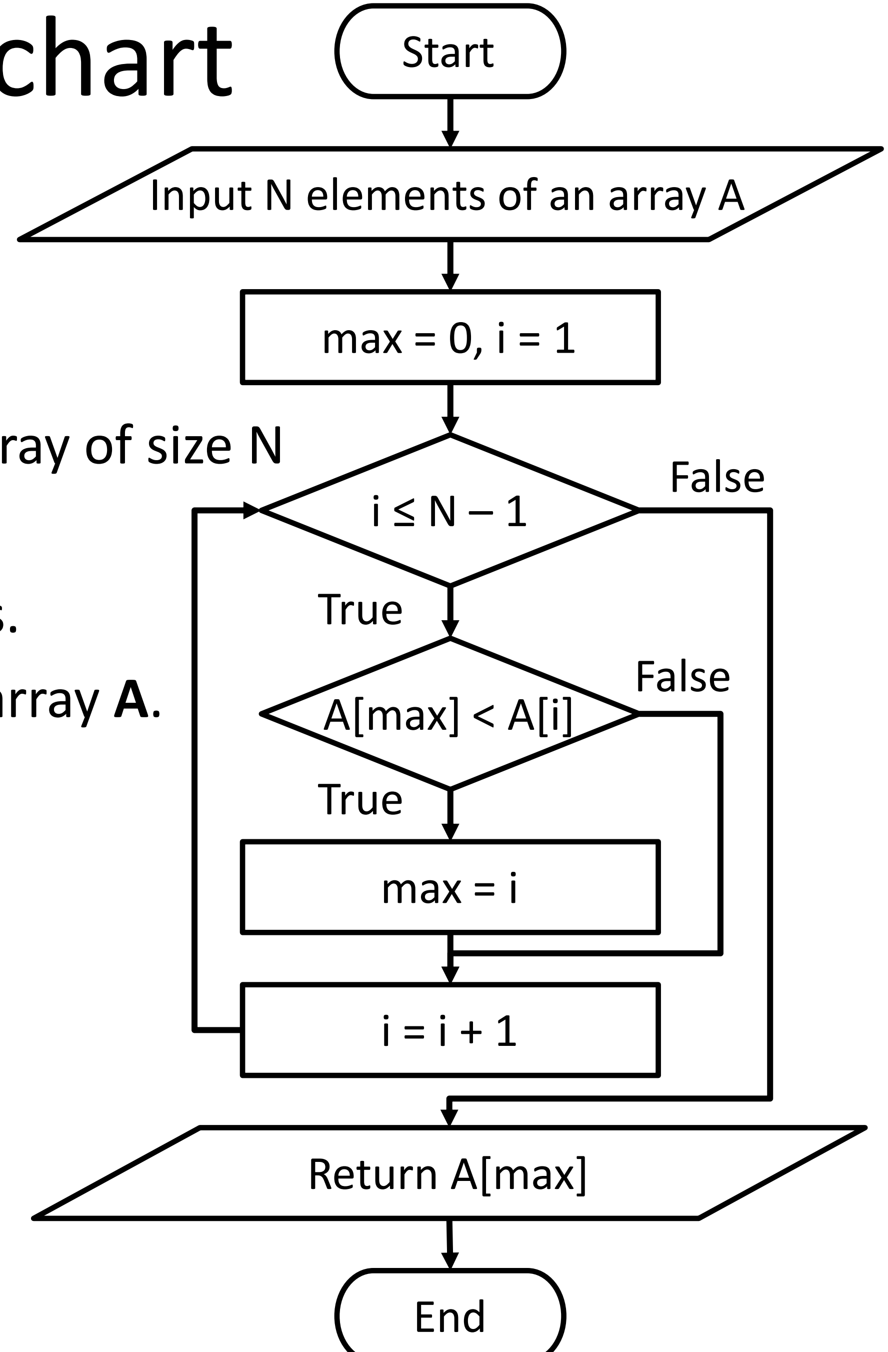
- It is more structured than usual prose but less formal than a programming language.
- One can use various programming constructs like:
  - **Decision structures**: **if** … **else** …
  - **Loops**: **for** … **while** … (**do** … **while** …)
  - **Array indexing**: **A**[**i**], **A**[**i**][**j**]
  - **Return a value**: **return** value
  - **Call another method** by writing its name and argument list.

Pseudocode Vs. Flowchart

Example: Find maximum element in an Array of size N

- Algorithm **arrayMaxElement**(A,N)
- Input: An array **A** containing **N** integers.
- Output: The maximum element in an array **A**.

```
max = 0
for i = 1 to N – 1 do
    if A[max] < A[i]
        max = i
return A[max]
```



Asymptotic Notations

Introduction

- How running time of an algorithm increases with the size of the input in the limit?
- Order of growth of the running time of an algorithm.
  - A simple characterization of algorithm's efficiency.
- Allows to compare the relative performance of alternative algorithms.
- Algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.

Example: A[i] = A[0] + A[1] + … + A[i]

Algorithm **arrayElementSum**(A,N)

Input: An array **A** containing **N** integers.

Output: An updated array **A** containing **N** integers.

1

```
for i = 1 to N – 1 do
    sum = 0
    for j = 0 to i do
        sum = sum + A[j]
    A[i] = sum
```

Option 2 is better

2

```
for i = 1 to N – 1 do
    A[i] = A[i] + A[i – 1]
```

Analysis of Algorithms

- Identify primitive operations, i.e., low level operations independent of programming language.
- Example:
  - Data movement operations (assignment).
  - Control statements (branch, method call, return).
  - Arithmetic and Logical operations.
- Primitive operations can easily be identified by inspecting the pseudo-code.

Contd…

	Cost	Frequency
for i = 1 to N – 1 **do**	c1	N
sum = 0	c2	N – 1
for j = 0 to i **do**	c3	$\sum_{i=1}^{N-1}(i+2)$
sum = sum + A[j]	c4	$\sum_{i=1}^{N-1}(i+1)$
A[i] = sum	c5	N – 1

$$c1N + c2(N-1) + c3\sum_{i=1}^{N-1}(i+2) + c4\sum_{i=1}^{N-1}(i+1) + c5(N-1)$$

$$c1N + c2N - c2 + c3\left(\frac{N(N-1)}{2}\right) + 2.c3.N - 2.c3 + c4\left(\frac{N(N-1)}{2}\right) + c4N - c4 + c5N - c5)$$

$$N^2\left(\frac{c3}{2} + \frac{c4}{2}\right) + N\left(c1 + c2 + \frac{3}{2}c3 + \frac{c4}{2} + c5\right) - (c2 + 2.c3 + c4 + c5)$$

Contd…

	Cost	Frequency
1. **for** i = 1 to N – 1 **do**	c1	N
2. A[i] = A[i] + A[i – 1]	c2	N – 1

$$c1N + c2(N-1)$$

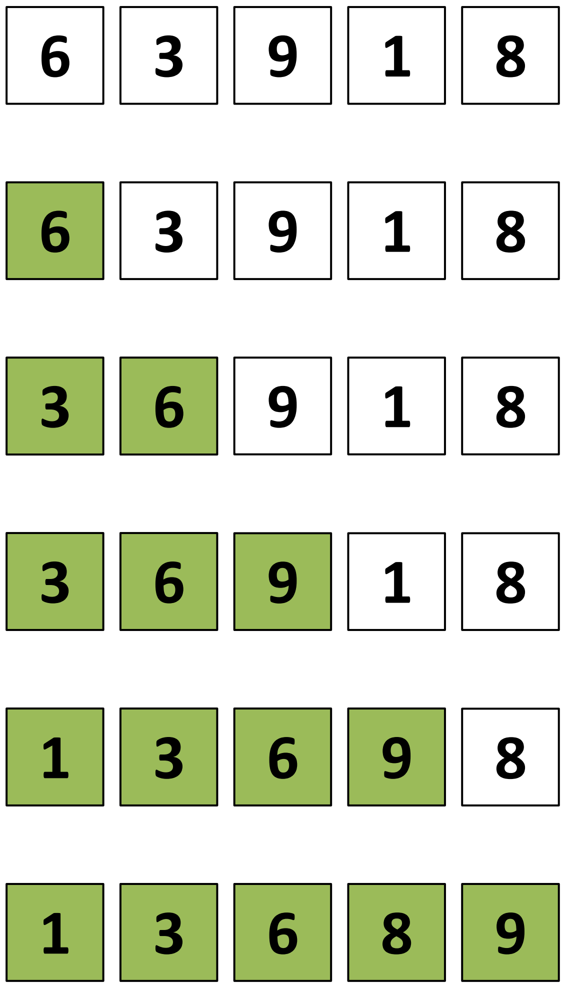
$$N(c1 + c2) - c2$$

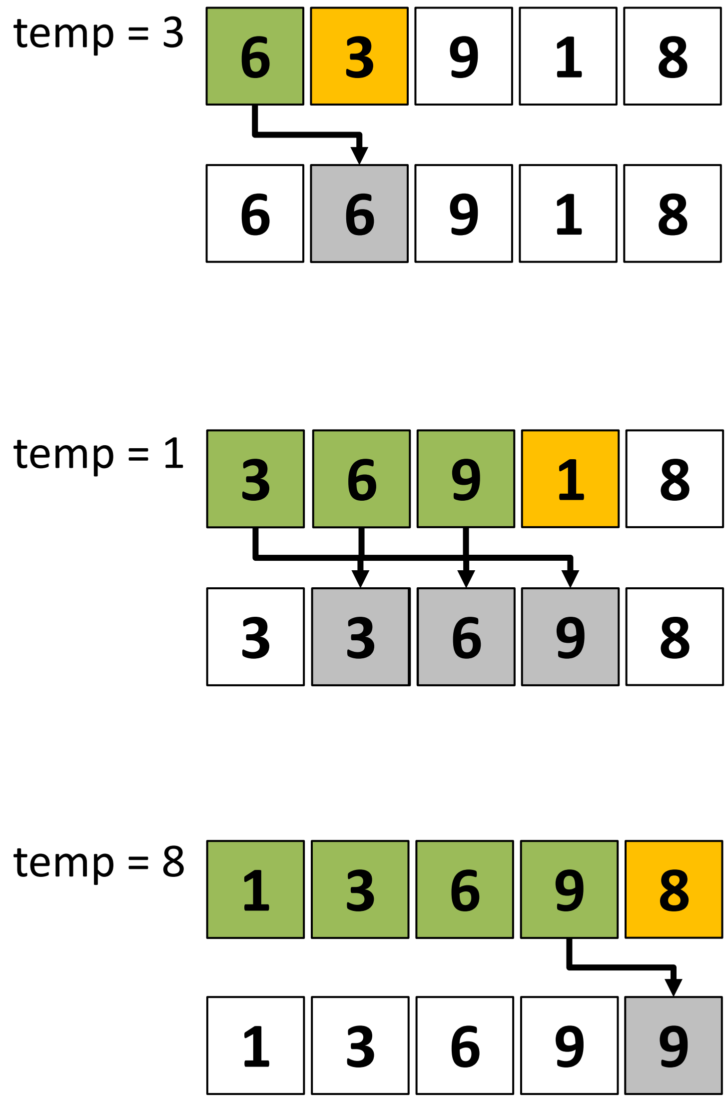
Asymptotic Notation

- Describes the running times of algorithms as a function of the size of its input.

- The running time of an algorithm can be
  - Worst-case running time
  - Average-case running time
  - Best-case running time

Insertion Sort





Example

Algorithm **insertionSort**(A, N)

Input: An array **A** containing **N** elements.

Output: The elements of **A** get sorted in increasing order.

for i = 1 to N – 1	c1	N
temp = A[i]	c2	N–1
j = i	c3	N–1
while j > 0 and a[j-1] > temp	c4	$\sum_{i=1}^{N-1} t_i$
a[j] = a[j-1]	c5	$\sum_{i=1}^{N-1} (t_i - 1)$
j = j – 1	c6	$\sum_{i=1}^{N-1} (t_i - 1)$
a[j] = temp	c7	N–1

Contd…

$$T(n) = c1N + c2(N-1) + c3(N-1) + c4\sum_{i=1}^{N-1} t_i$$
$$+ c5\sum_{i=1}^{N-1}(t_i - 1) + c6\sum_{i=1}^{N-1}(t_i - 1) + c7(N-1)$$

1. **for** i = 1 to N – 1
2. temp = A[i]
3. j = i
4. **while** j > 0 and a[j-1] > temp
5. a[j] = a[j-1]
6. j = j – 1
7. a[j] = temp

Best case:

$$T(n) = c1N + c2(N-1) + c3(N-1) + c4(N-1) + c7(N-1)$$

$$T(n) = (c1 + c2 + c3 + c4 + c7)N - (c2 + c3 + c4 + c7)$$

Worst case:

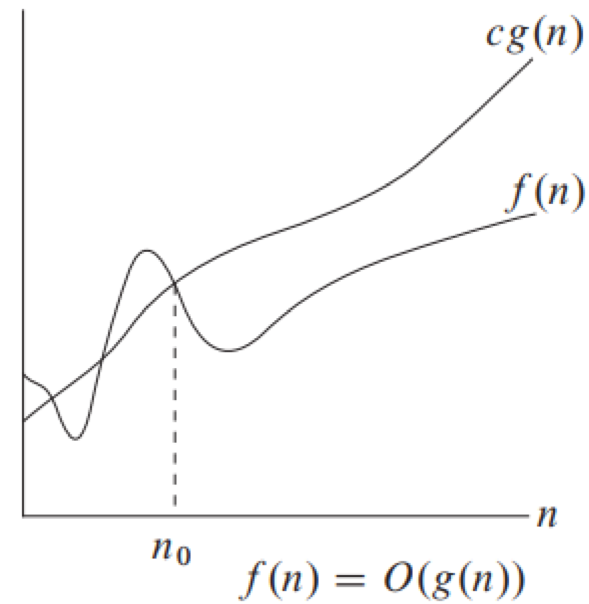
$$T(n) = c1N + c2(N-1) + c3(N-1) + c4\left(\frac{N(N+1)}{2} - 1\right) + c5\left(\frac{N(N-1)}{2}\right) + c6\left(\frac{N(N-1)}{2}\right)$$
$$+ c7(N-1)$$

$$T(n) = \left(\frac{c4}{2} + \frac{c5}{2} + \frac{c6}{2}\right)N^2 + \left(c1 + c2 + c3 + \frac{c4}{2} - \frac{c5}{2} - \frac{c6}{2} + c7\right)N - (c2 + c3 + c4 + c7)$$

Big-Oh Notation

- Gives only an asymptotic upper bound.

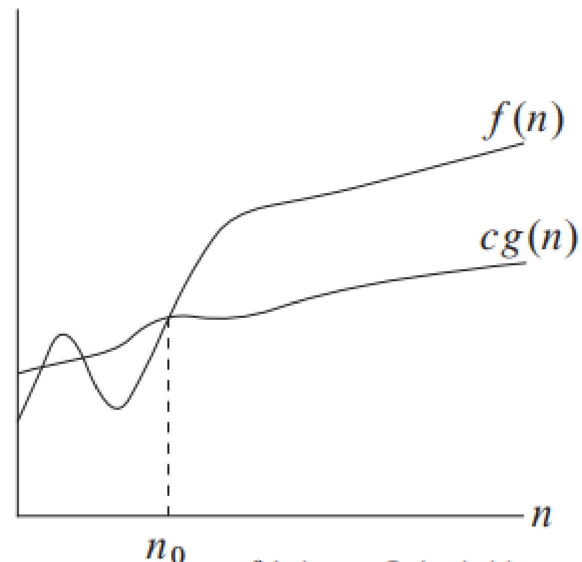
- For a given function g(n), O(g(n)) (pronounced “big-oh of g of n” or sometimes just “oh of g of n”) denotes the set of functions



$$O(g(n)) = \{ f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \} .$$

Big-Omega Notation

- Gives only an asymptotic lower bound.
- For a given function g(n), Ω(g(n)) (pronounced "big-omega of g of n" or sometimes just "omega of g of n") denotes the set of functions

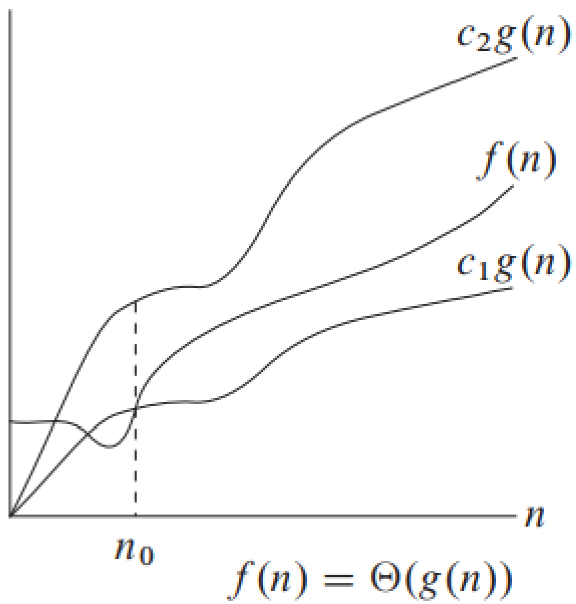


$f(n) = \Omega(g(n))$

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$

Theta Notation

- Gives an asymptotic tight bound.
- Function f(n) belongs to the set θ(g(n)) if there exist positive constants c_1 and c_2 such that it can be "sandwiched" between $c_1g(n)$ and $c_2g(n)$, for sufficiently large n.



- For a given function g(n), θ(g(n)) denotes the set of functions:

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \} .$$ [1]

Example: Big-Oh

Show that: $\mathbf{n^2/2 - 3n = O(n^2)}$

- Determine positive constants $\mathbf{c_1}$ and $\mathbf{n_0}$ such that

$$\mathbf{n^2/2 - 3n \le c_1 n^2} \text{ for all } \mathbf{n \ge n_0}$$

- Diving by $\mathbf{n^2}$

$$\mathbf{1/2 - 3/n \le c1}$$

- For: $\mathbf{n = 1}$, $\mathbf{1/2 - 3/1 \le c1}$ (Holds for $\mathbf{c1 \ge 1/2}$)

  $\mathbf{n = 2}$, $\mathbf{1/2 - 3/2 \le c1}$ (Holds and so on…)
- The inequality holds for any $\mathbf{n \ge 1}$ and $\mathbf{c_1 \ge 1/2}$.
- Thus by choosing the constant $\mathbf{c_1 = 1/2}$ and $\mathbf{n_0 = 1}$, one can verify that $\mathbf{n^2/2 - 3n = O(n^2)}$ holds.

Example: Big-Omega

Show that: $\mathbf{n^2/2 - 3n = \Omega(n^2)}$

- Determine positive constants $\mathbf{c_1}$ and $\mathbf{n_0}$ such that

$$c_1 n^2 \le n^2/2 - 3n \text{ for all } n \ge n_0$$

- Diving by $\mathbf{n^2}$

$$c_1 \le 1/2 - 3/n$$

- For: $\mathbf{n = 1}$, $\mathbf{c_1 \le 1/2 - 3/1}$ (Not Holds)
  $\mathbf{n = 2}$, $\mathbf{c_1 \le 1/2 - 3/2}$ (Not Holds)
  $\mathbf{n = 3}$, $\mathbf{c_1 \le 1/2 - 3/3}$ (Not Holds)
  $\mathbf{n = 4}$, $\mathbf{c_1 \le 1/2 - 3/4}$ (Not Holds)
  $\mathbf{n = 5}$, $\mathbf{c_1 \le 1/2 - 3/5}$ (Not Holds)
  $\mathbf{n = 6}$, $\mathbf{c_1 \le 1/2 - 3/6}$ (Not Holds and Equals ZERO)
  $\mathbf{n = 7}$, $\mathbf{c_1 \le 1/2 - 3/7}$ or $\mathbf{c_1 \le (7\text{-}6)/14}$ or $\mathbf{c_1 \le 1/14}$ (Holds for $\mathbf{c_1 \le 1/14}$)
- The inequality holds for any $\mathbf{n \ge 7}$ and $\mathbf{c_1 \le 1/14}$.
- Thus by choosing the constant $\mathbf{c_1 = 1/14}$ and $\mathbf{n_0 = 7}$, one can verify that $\mathbf{n^2/2 - 3n = \Omega(n^2)}$ holds.

Example: Theta

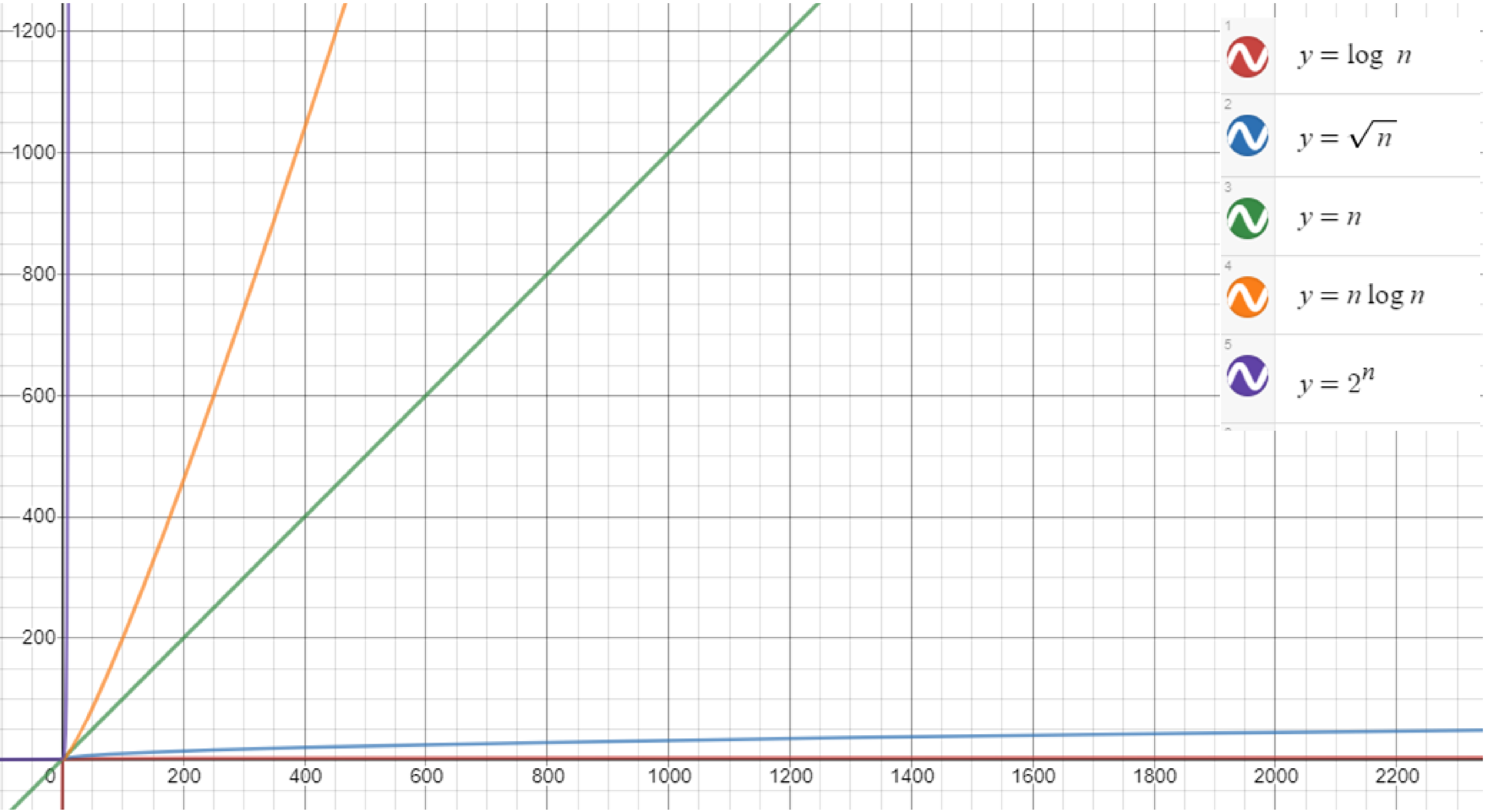
- Show that: $\mathbf{n^2/2 - 3n = \theta(n^2)}$
- Determine positive constants $\mathbf{c_1}$, $\mathbf{c_2}$, and $\mathbf{n_0}$ such that

$$c_1 n^2 \le n^2/2 - 3n \le c_2 n^2 \text{ for all } n \ge n_0$$

- Diving by $\mathbf{n^2}$

$$c_1 \le 1/2 - 3/n \le c_2$$

- Right Hand Side Inequality holds for any $\mathbf{n \ge 1}$ and $\mathbf{c_2 \ge 1/2}$.
- Left Hand Side Inequality holds for any $\mathbf{n \ge 7}$ and $\mathbf{c_1 \le 1/14}$.
- Thus by choosing the constants $\mathbf{c_1 = 1/14}$ and $\mathbf{c_2 = 1/2}$ and $\mathbf{n_0 = 7}$, one can verify that $\mathbf{n^2/2 - 3n = \theta(n^2)}$ holds.



1200
1000
800
600
400
200
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
$y = \log\ n$
$y = \sqrt{n}$
$y = n$
$y = n \log n$
$y = 2^n$

o-Notation

- o-notation denotes an upper bound that is not asymptotically tight.
- Formally o(g(n)) ("little-oh of g of n") is defined as the set

$$o(g(n)) = \{ f(n) : \text{ for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0 \} .$$

- For example, $2n = o(n^2)$, but $2n^2 \,!= o(n^2)$.
- Intuitively, in o-notation, the function f(n) becomes insignificant relative to g(n) as n approaches infinity; that is, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

ω-Notation

- ω-notation denotes a lower bound that is not asymptotically tight.
- One way to define it is as f(n) ∈ ω(g(n)) if and only if g(n) ∈ o(f(n)).
- Formally, ω(g(n)) ("little-omega of g of n") is defined as the set

$$\omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0 \text{, there exists a constant } n_0 > 0 \text{ such that } 0 \le c g(n) < f(n) \text{ for all } n \ge n_0 \} .$$

- For example, $n^2/2 = \omega(n)$, but $n^2/2 \ != \omega(n^2)$.
- The relation f(n) ∈ ω(g(n)) implies that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, if limit exists.