Fundamentals of Computer and Programming

Lecture 2

Structured Programming and Algorithm Design

Instructor: Morteza Zakeri, Ph.D.

(zakeri@aut.ac.ir)

Modified Slides from Dr. Hossein Zeinali and Dr. Bahador Bakhshi

School of Computer Engineering,

Amirkabir University of Technology

Spring 2025

What We Will Learn

- Algorithms
- Structured program development
- Pseudo-code and Flowcharts
- Sample algorithms to practice problem-solving steps
- **Input** and **Output** analysis

Recap: Algorithms

- An algorithm is a procedure for solving a problem in terms of
  - the **actions** to execute, and
  - the **order** in which these actions should be executed.
- Specifying the order in which statements should execute in a computer program is called **program control**.

Structured Programming

- All programs could be written in terms of **three control structures:**
  - The **sequence** structure,
  - The **selection** structure, and
  - The **iteration** structure.
- The notion of **structured programming became** almost synonymous with "*goto* elimination."

Recap: Pseudocode

- **Pseudocode** is an informal artificial language similar to everyday English
- **Pseudocode** helps you develop algorithms before converting them to **structured C programs**.
- **Pseudocode** is convenient and user-friendly.
- **Pseudocode** is language-independent.
- Computers **do not execute** pseudocodes.

Recap: Flowcharts

- A **flowchart** is a **graphical representation** of an algorithm or a **portion of** an algorithm.
- You draw flowcharts using certain special-purpose symbols such as *rectangles*, *diamonds*, *rounded rectangles*, and *small circles*, connected by arrows called ***flowlines***.
- Flowcharts clearly show how control structures operate.
  - Pseudocode **is preferred** by most programmers.

Computing the average of three numbers

Algorithm: Average

1. print “Please enter three integers”
2. read x1, x2, x3
3. sum ← x1 + x2 + x3
4. average ← sum / 3
5. print “Average = ” average

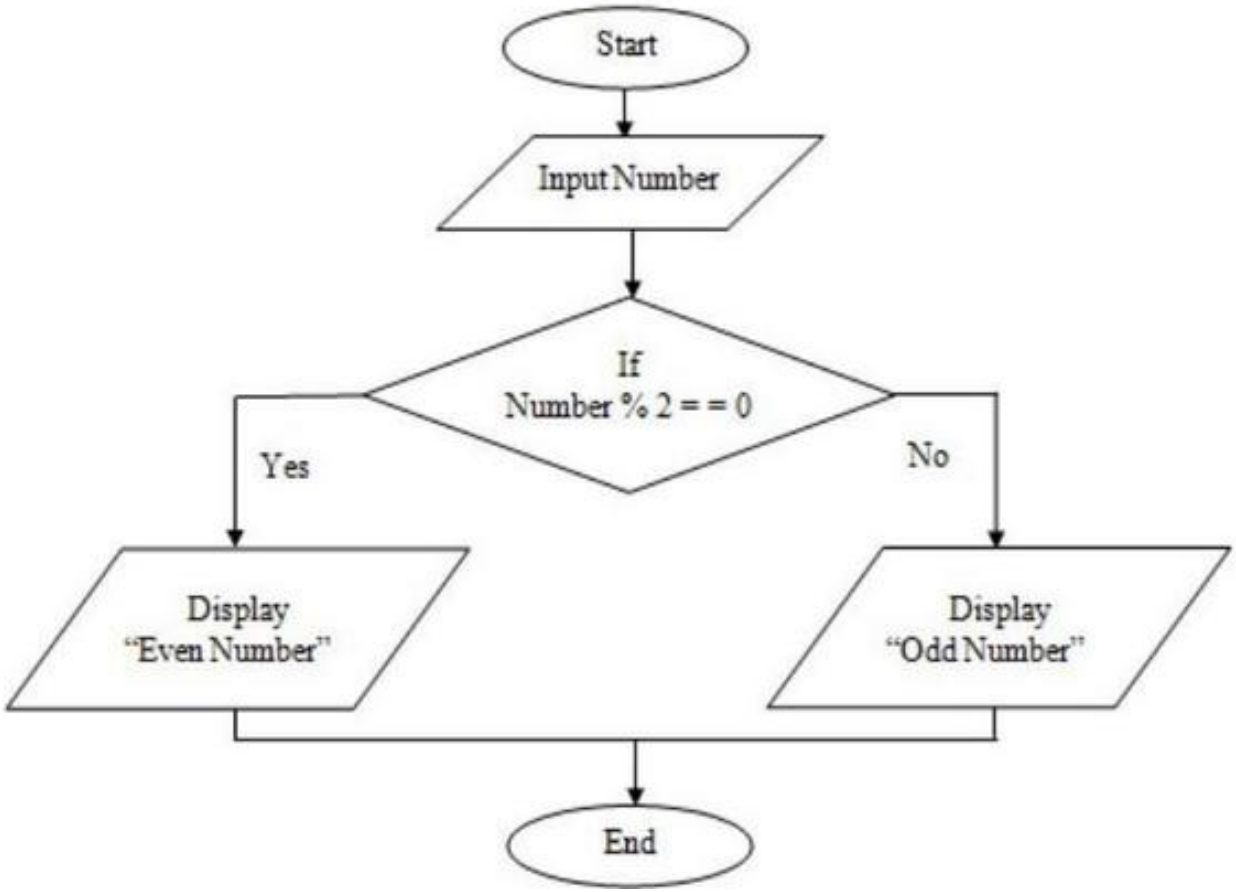
الگوریتم تشخیص زوج یا فرد بودن عدد (شبه کد)

Algorithm: Odd-Even-1

```
1. print "Please enter an integer"
2. read n
3. y ← n mod 2
4. if (y == 0)
        print "Number is even"
   else
        print "Number is odd"
```

الگوریتم تشخیص زوج یا فرد بودن عدد (روندنما)

- Flowchart



الگوریتم تشخیص زوج یا فرد بودن عدد

Algorithm: Odd-Even-2

1. print “Please enter an integer”
2. read n
3. if(n < 0)
   n ← -1 * n
4. while(n >= 2)
   n ← n – 2
5. if(n == 0)
   print “even”
   else
   print “odd”

Verify the Algorithm

الگوریتم تشخیص زوج یا فرد بودن عدد

Algorithm: *Odd-Even-3*

1. print “Please enter an integer”
2. read n
3. while (n >= 2) or (n <= -1)
   n ← n - sign(n) * 2
4. if (n == 1)
   print “odd”
   else
   print “even”

الگوریتمی که یک رشته عدد را که با ۰ تمام می‌شود را می‌گیرد و تعداد اعداد زوج و فرد را چاپ می‌کند

Algorithm: Count Odd-Even

```
odd_cnt ← 0
even_cnt ← 0
print "Please enter an integer"
read n
while (n != 0)
        y ← n mod 2
        if (y == 0)
                even_cnt ← even_cnt + 1
        else
                odd_cnt ← odd_cnt + 1
        print "Please enter an integer"
        read n

print "Odd = " odd_cnt "Even = " even_cnt
```

الگوریتمی که یک عدد صحیح مثبت را بگیرد و مجموع ارقام آن را چاپ کند

Algorithm: Digit-Sum

```
print "Please enter a positive integer"
read n
sum ← 0
m ← n
while (n != 0)
        y ← n mod 10
        sum ← sum + y
        n ← n - y
        n ← n / 10
print "sum of digits of" m " = " sum
```

Verify the Algorithm

الگوریتمی که یک عدد صحیح مثبت را بگیرد و آنرا در مبنای ۸ چاپ کند

Algorithm: Base-8

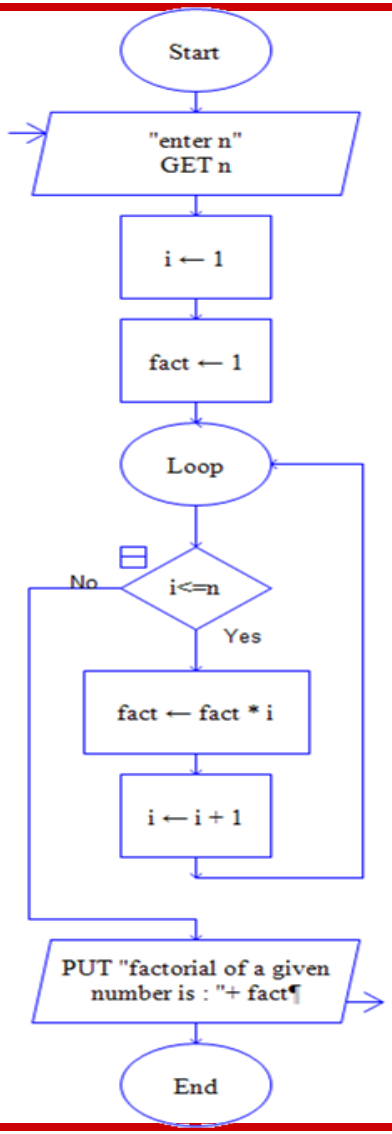
```
print "Please enter a positive integer"
read n
i ← 0
while (n != 0)
        x[i] ← n mod 8
        n ← floor (n / 8)
        i ← i + 1
i ← i - 1
while (i >= 0)
        print x[i]
        i ← i - 1
```

الگوریتمی که یک عدد صحیح مثبت را بگیرد و فاکتوریل آن را تولید کند

Algorithm: Factorial-1

```
print "Please enter a positive integer"
read n
i ← 1
result ← 1
while (i <= n)
        result ← i * result
        i ← i + 1

return  result
```



الگوریتمی که یک عدد صحیح مثبت را بگیرد و فاکتوریل آن را تولید کند

Algorithm: Factorial-2

```
print "Please enter a positive integer"
read n
result ← 1
while (n > 0)
        result ← result * n
        n ← n - 1

return  result
```

الگوریتمی که یک عدد صحیح مثبت را بگیرد و فاکتوریل آنرا تولید کند.

Algorithm: Factorial-Recursive (**n**)

```
if (n <= 1)
    return 1
else
    return n * Factorial-Recursive (n - 1)
```

الگوریتمی که یک رشته عدد را که محل عضو اول آن با **start** و محل عضو آخر آن با **end** مشخص شده است را به صورت صعودی مرتب کند.

Algorithm: sort (**x, start, end**)

```
while (start != end)
        j ← find index of minimum element from start to end
        swap x[j] and x[start]
        start ← start + 1
```

==========

Algorithm find_min(**x, start, end**)

```
y ← start
i ← start + 1
while (i <= end)
        if(x[i] < x[y])
                y ← i
        i ← i + 1
return y
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Verify the Algorithm

الگوریتمی که یک رشته عدد را که محل عضو اول آن با **start** و محل عضو آخر آن با **end** مشخص شده است را به صورت صعودی مرتب کند.

Algorithm swap(**x, j, i**)

temp ← x[j]
x[j] ← x[i]
x[i] ← temp

Algorithm swap2(**x, j, i**)

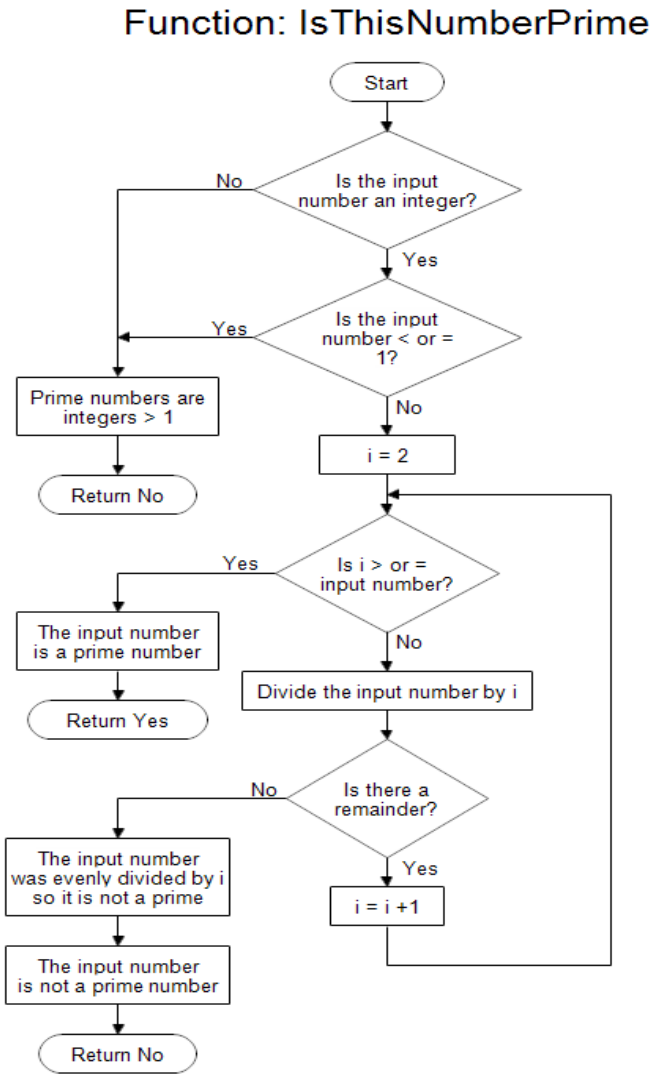
x[j] ← x[j] + x[i]
x[i] ← x[j] – x[i]
x[j] ← x[j] – x[i]

الگوریتمی که آرایه صعودی از اعداد صحیح را بگیرد و آن را به‌صورت درجا (بدون استفاده از آرایه دیگر) تبدیل به آرایه نزولی کند.

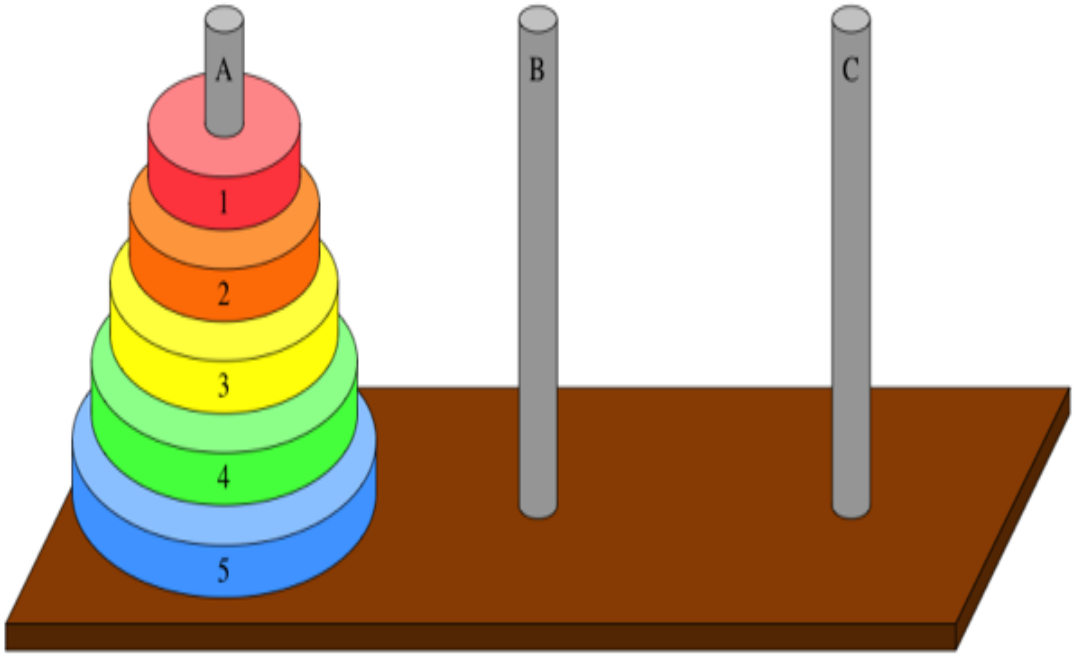
Algorithm reverse(**A, start, end**)

```
if (start >= end)
    return
else
    swap(A, start, end)
    reverse(A, start + 1, end - 1)
```

Flowchart to check whether a number is prime or not

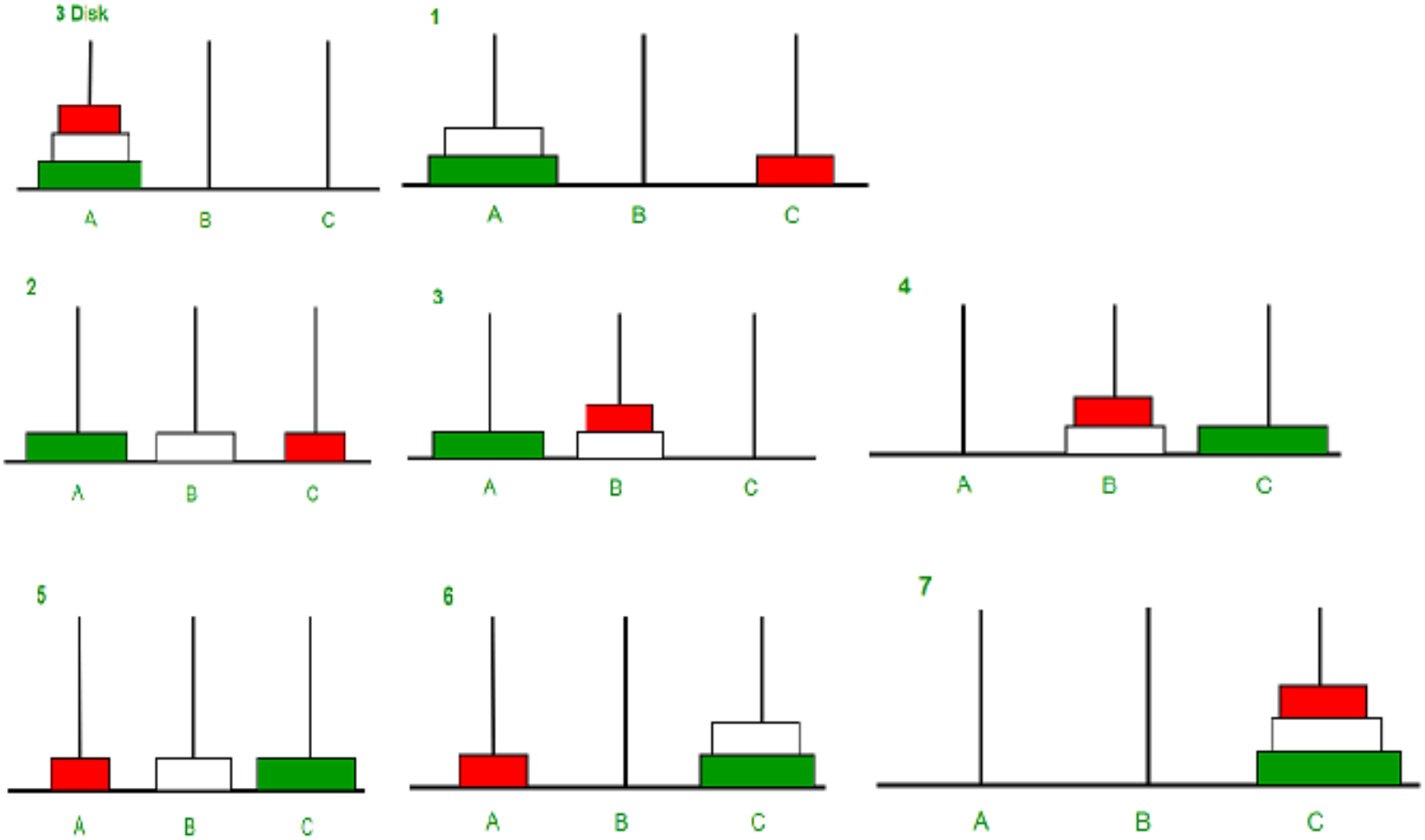


Tower of Hanoi



Tower of Hanoi

- Solution trace for n=3 disks:



Tower of Hanoi

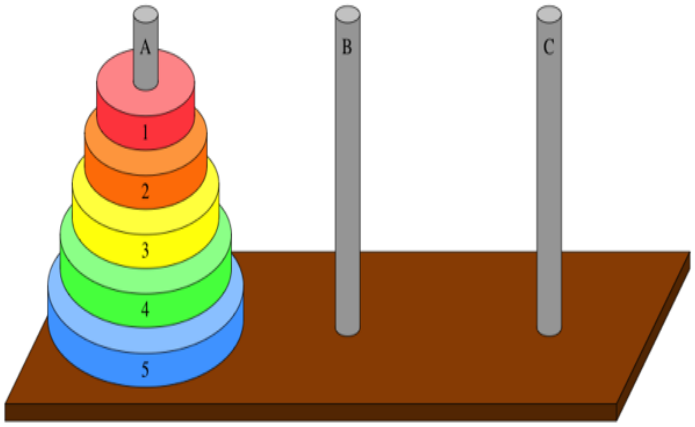
Algorithm Hanoi(n, source, target, auxiliary):

```
if (n <= 0)
    return

# Move n - 1 disks from source to auxiliary
Hanoi(n - 1, source, auxiliary, target)

# Move the nth disk from the source to the target
append the source last disk to the target

# Move the n - 1 disks that we left on the auxiliary onto the target
Hanoi(n - 1, auxiliary, target, source)
```



Tower of Hanoi

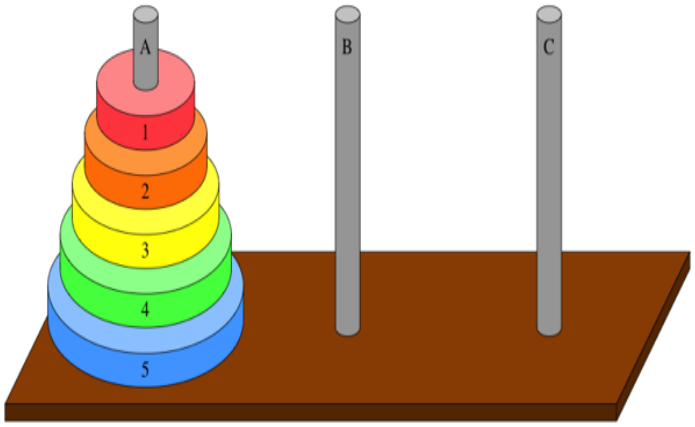
Algorithm Hanoi(n, source, target, auxiliary):

```
if (n <= 0)
    return

# Move n - 1 disks from source to auxiliary
Hanoi(n - 1, source, auxiliary, target)

# Move the nth disk from the source to the target
append the source last disk to the target

# Move the n - 1 disks that we left on the auxiliary onto the target
Hanoi(n - 1, auxiliary, target, source)
```



Example run:

A = [3, 2, 1]
B = []
C = []
Hanoi(3, A, C, B)

Tower of Hanoi

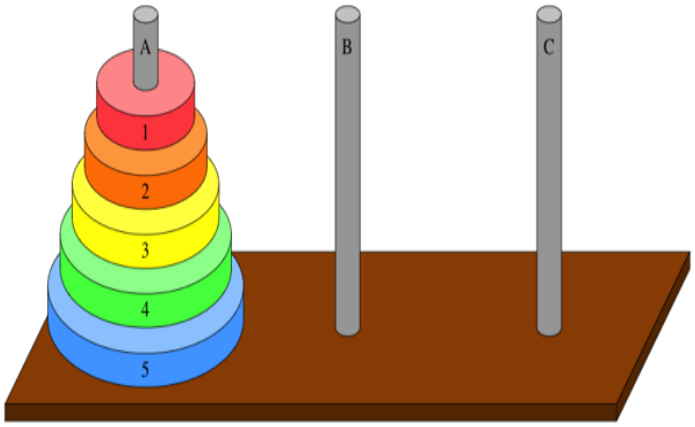
Algorithm Hanoi(n, source, target, auxiliary):

```
if (n <= 0)
    return

# Move n - 1 disks from source to auxiliary
Hanoi(n - 1, source, auxiliary, target)

# Move the nth disk from the source to the target
append the source last disk to the target

# Move the n - 1 disks that we left on the auxiliary onto the target
Hanoi(n - 1, auxiliary, target, source)
```

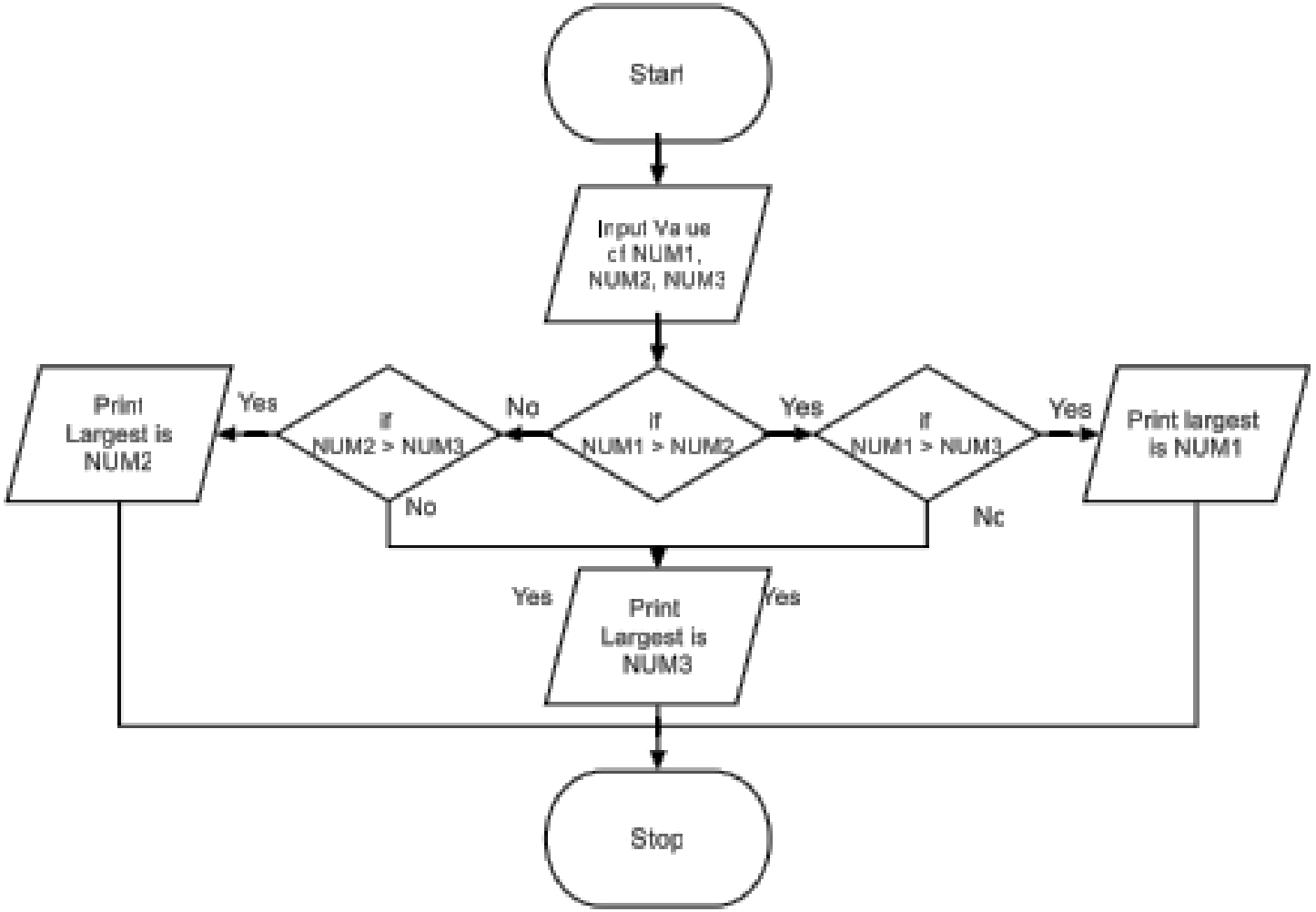


Example run:

A = [3, 2, 1]
B = []
C = []
Hanoi(3, A, C, B)

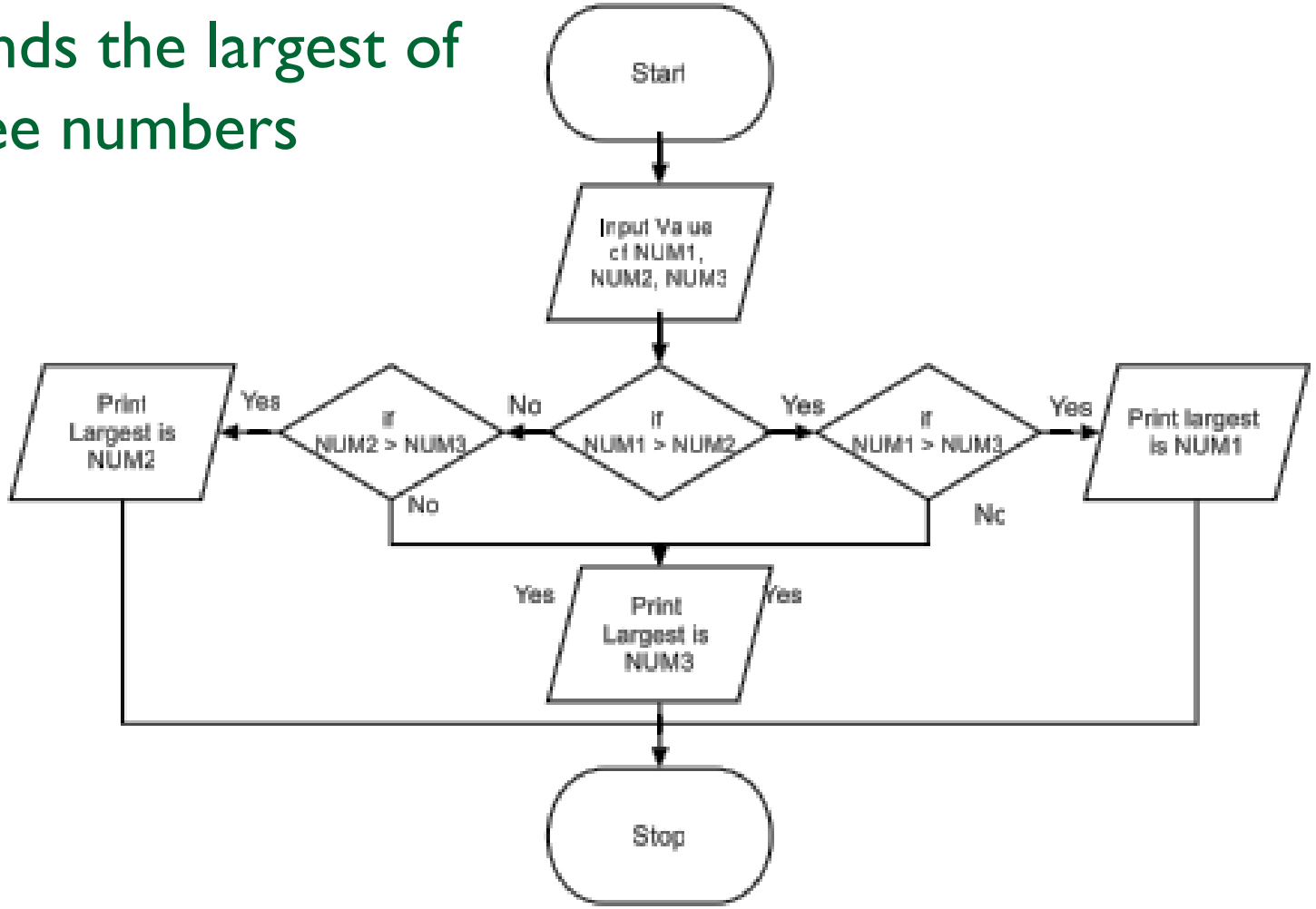
```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

What does the following flowchart compute?

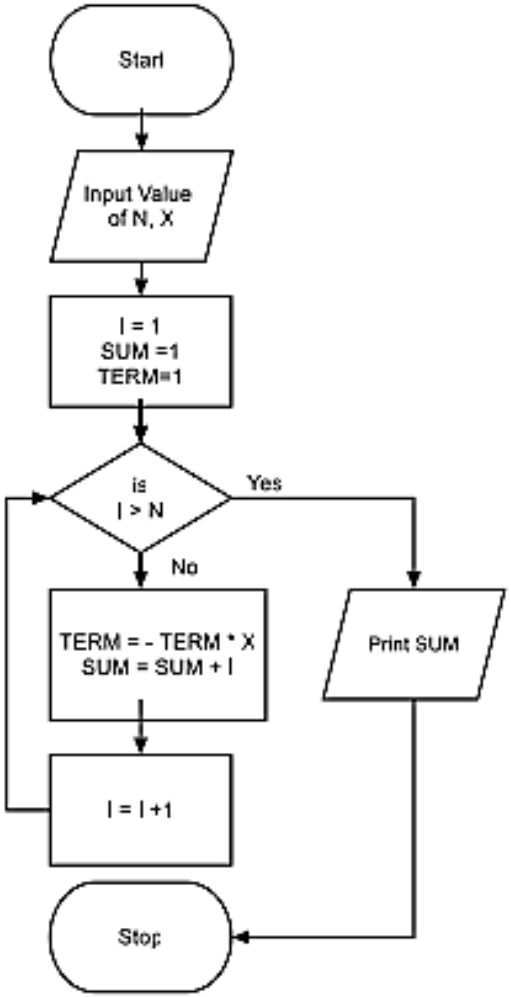


What does the following flowchart compute?

It finds the largest of three numbers



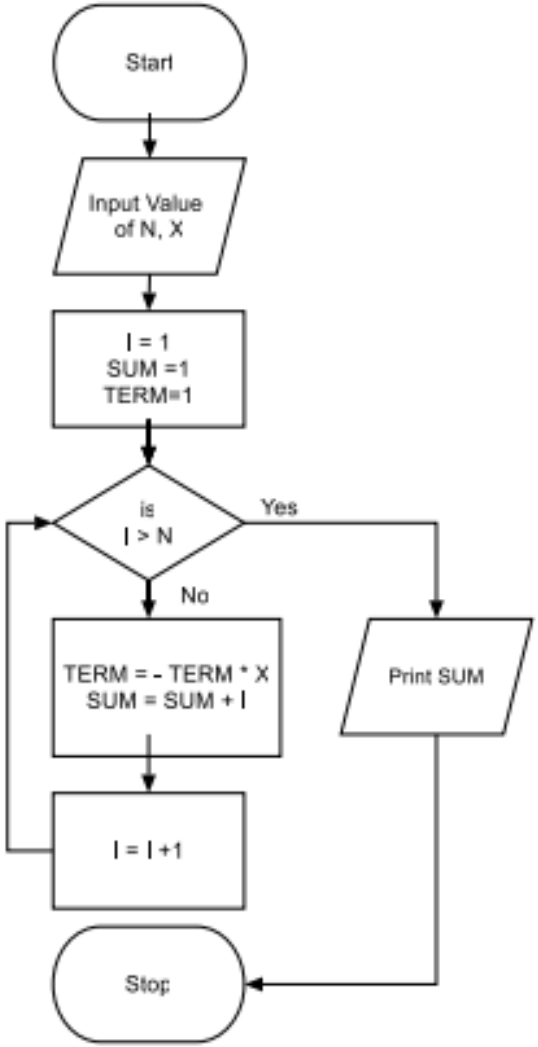
What does the following flowchart compute?



What does the following flowchart compute?

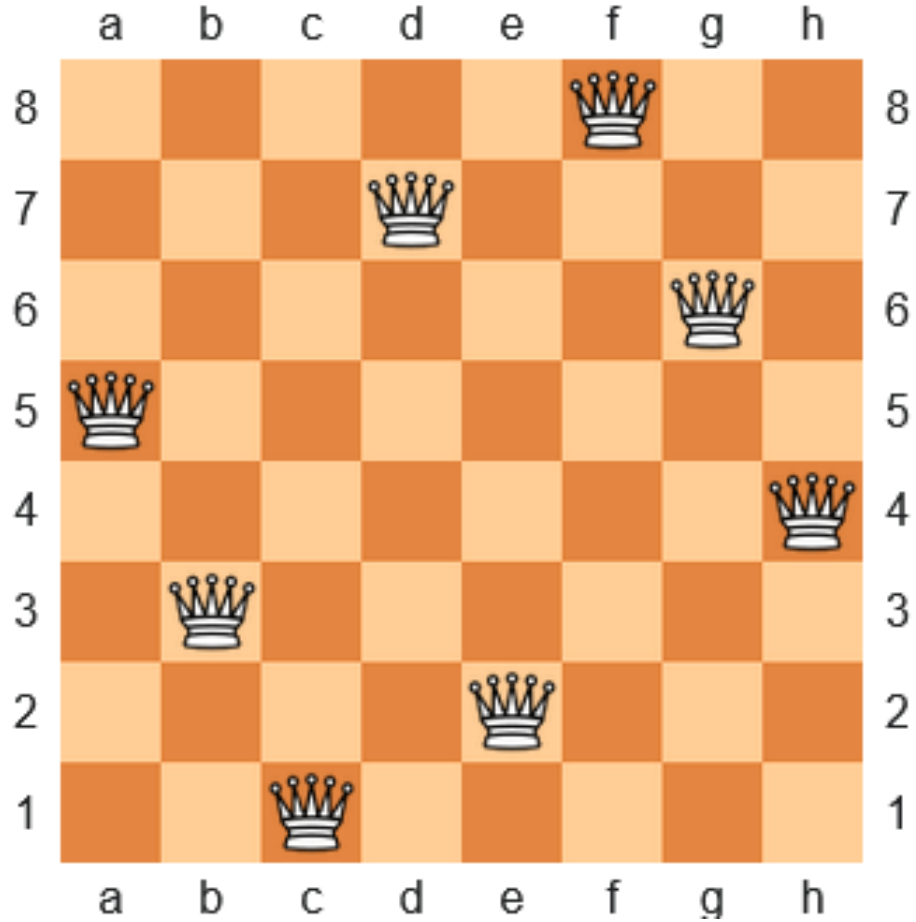
To find sum of the series

$1 - X + X^2 - X^3 \ldots X^N$



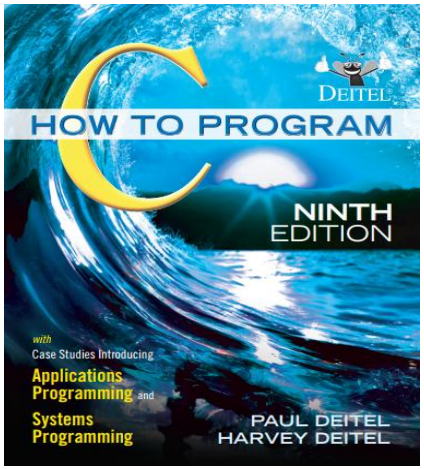
Advanced algorithms: Eight Queens Puzzle

- One possible solution



Summary

- There are **more than one algorithm** for a problem
  - Efficiency, Complexity, Clarity, …
- Algorithm (Programming Language) building blocks
  - Calculations (Lecture 4)
  - Input / Output (Lecture 5)
  - Decision Making (Lecture 6)
  - Repeating (Lecture 7)
  - Modular Programming (Lecture 8)
  - Arrays + Memory Management (Lectures 9 and 10)
  - Others (Complex data types and Files) (Lectures 11 and 12)



Reading Assignment

- Reading Assignment: Sections **3.1** to **3.4** of **Chapter 3** of "C How to Program"
- Read and practice the problems discussed: *https://www.programming9.com/raptor-flowcharts*