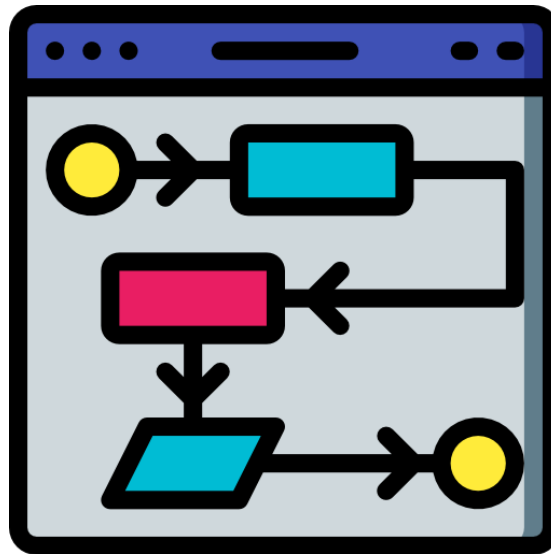




# Lecture 1: Algorithms and Pseudocode



Ms. Togzhan Nurtayeva  
Course Code: IT 235/A  
Semester 3  
Week 2-4  
Date: 08.10.2023

# Objectives

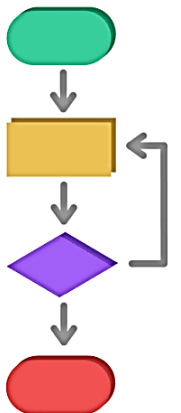
- ▶ Algorithms
- ▶ Flowcharts
- ▶ Types of Algorithms:
  - Searching (Linear, Binary)
  - Sorting (Bubble, Insertion)
  - Optimization (Greedy, Whale Optimization Algorithms (WOA))
- ▶ Recursive
- ▶ Big-O Notation



# Algorithm

- ▶ An *algorithm* is a finite sequence of precise instructions for performing a computation or for solving a problem.

WHAT IS AN  
ALGORITHM?



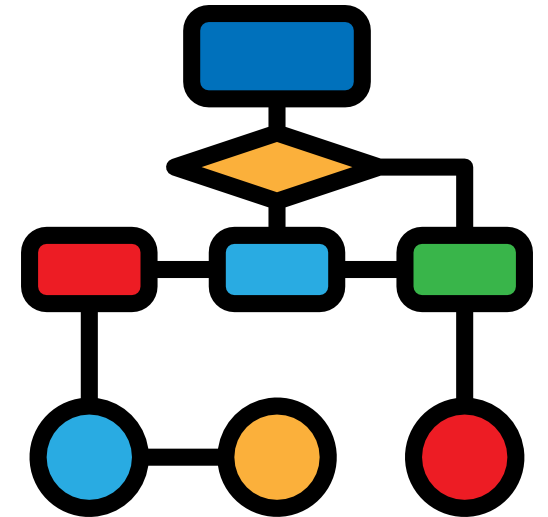
# Flowchart

A **flowchart** is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams.

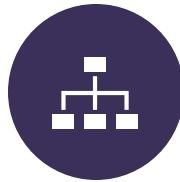
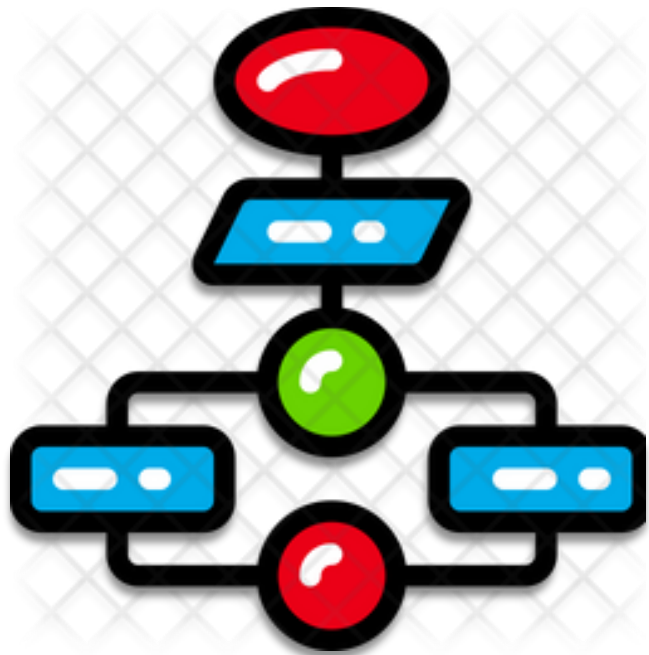


<https://app.diagrams.net/>

<https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial>



# Flowcharts can:



Demonstrate the way code is organized.



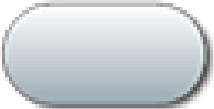

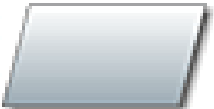
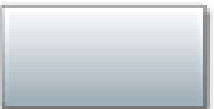

Visualize the execution of code within a program.



Show the structure of a website or application.

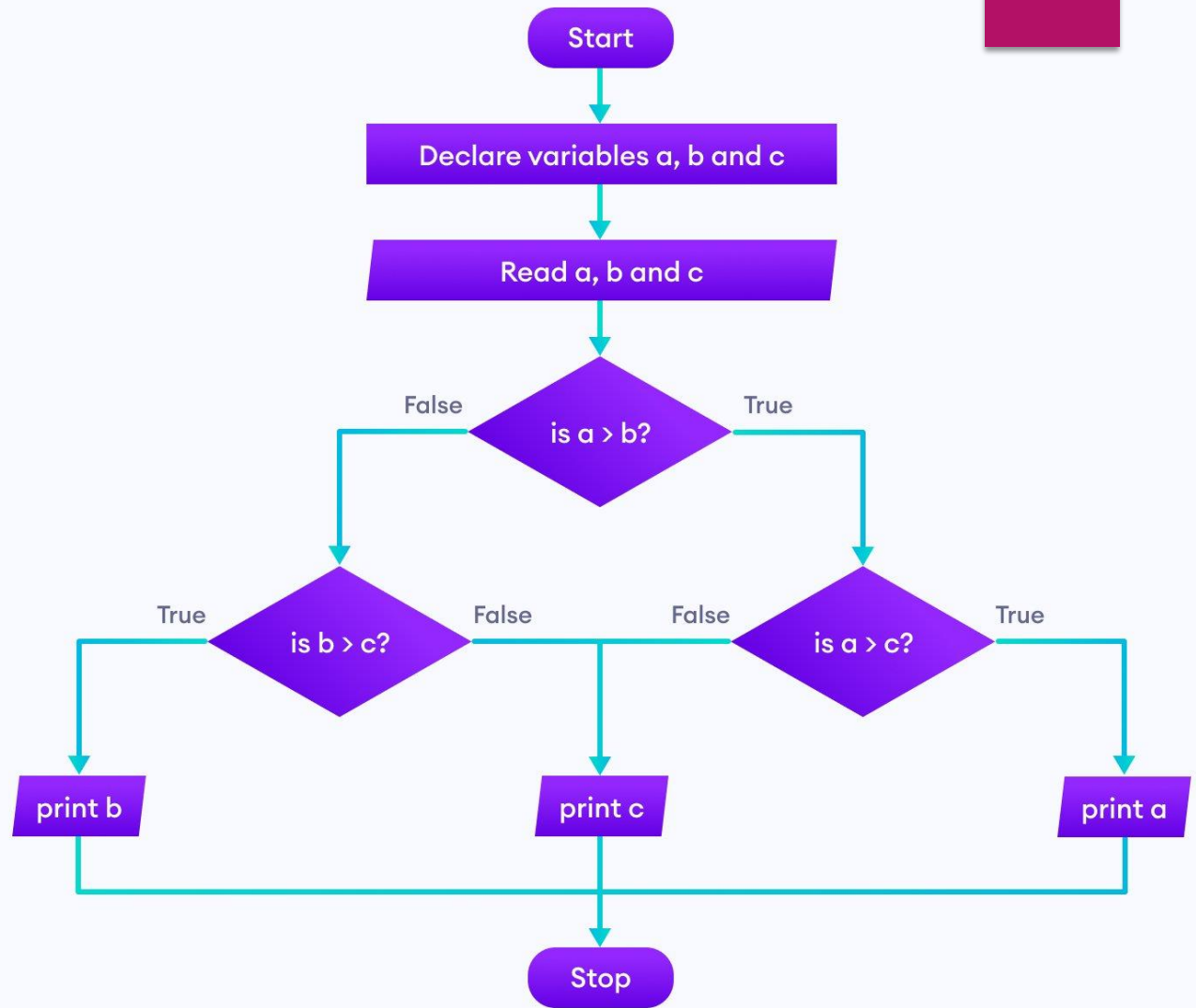


Understand how users navigate a website or program.

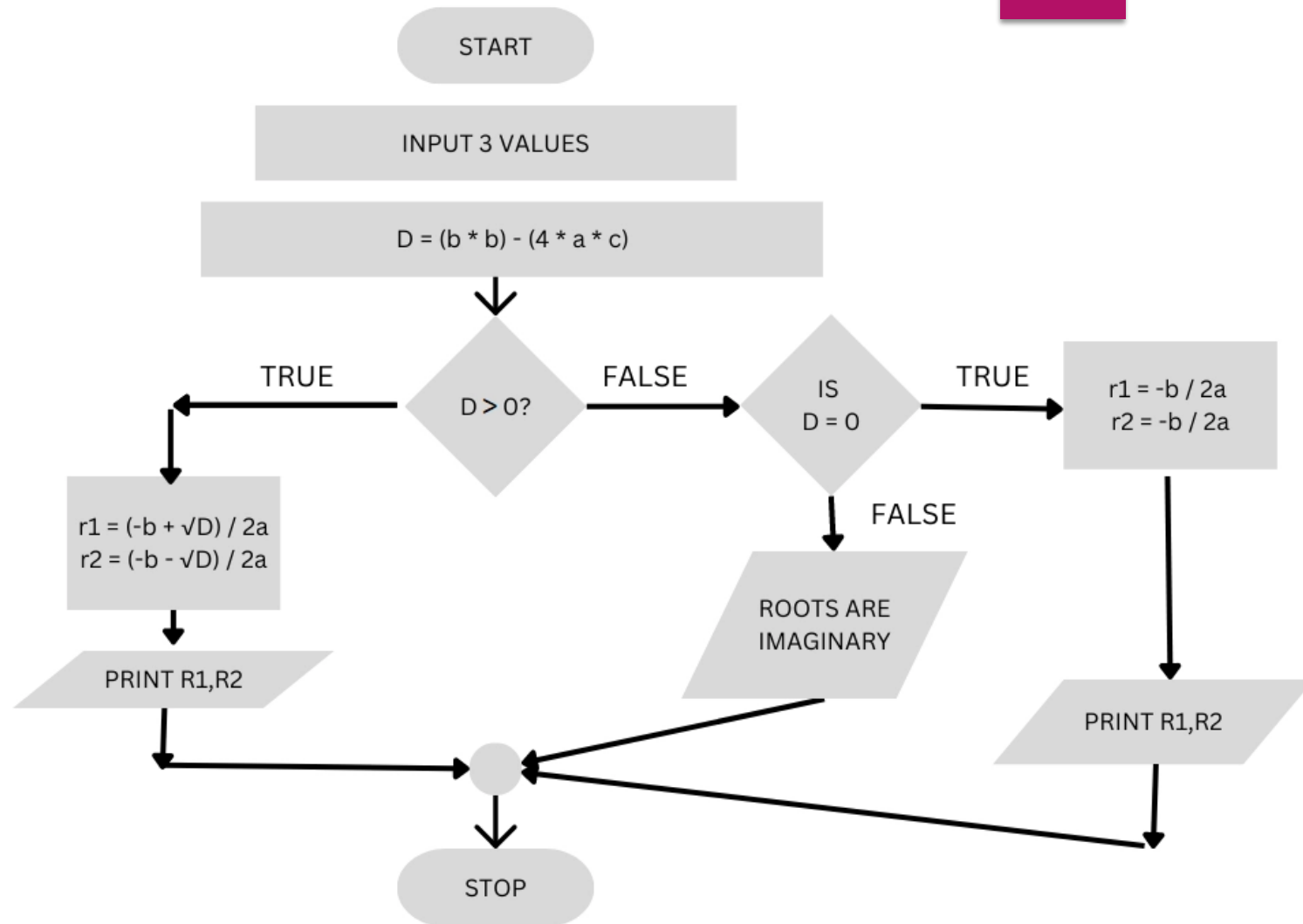
Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or output.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.

# Flowchart Symbols

# ► Finding the largest number



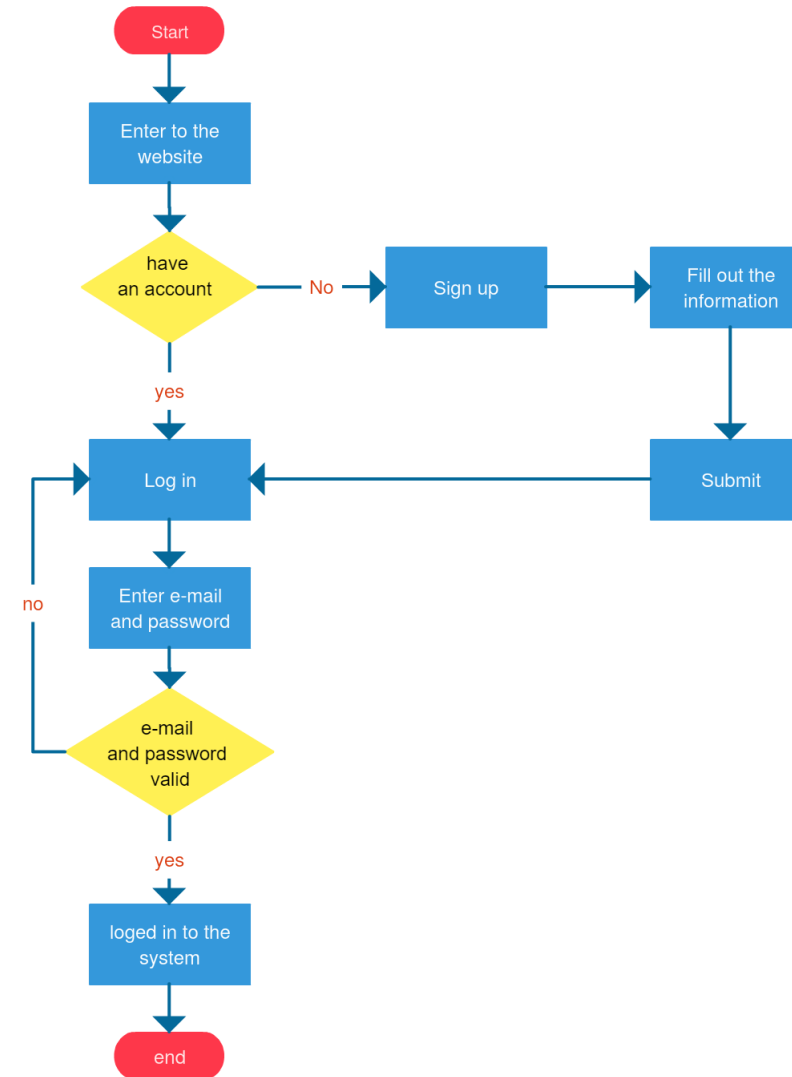
# ► Quadratic Equation





# ► Flowchart for the Website's Login Page

Log in Process Flowchart



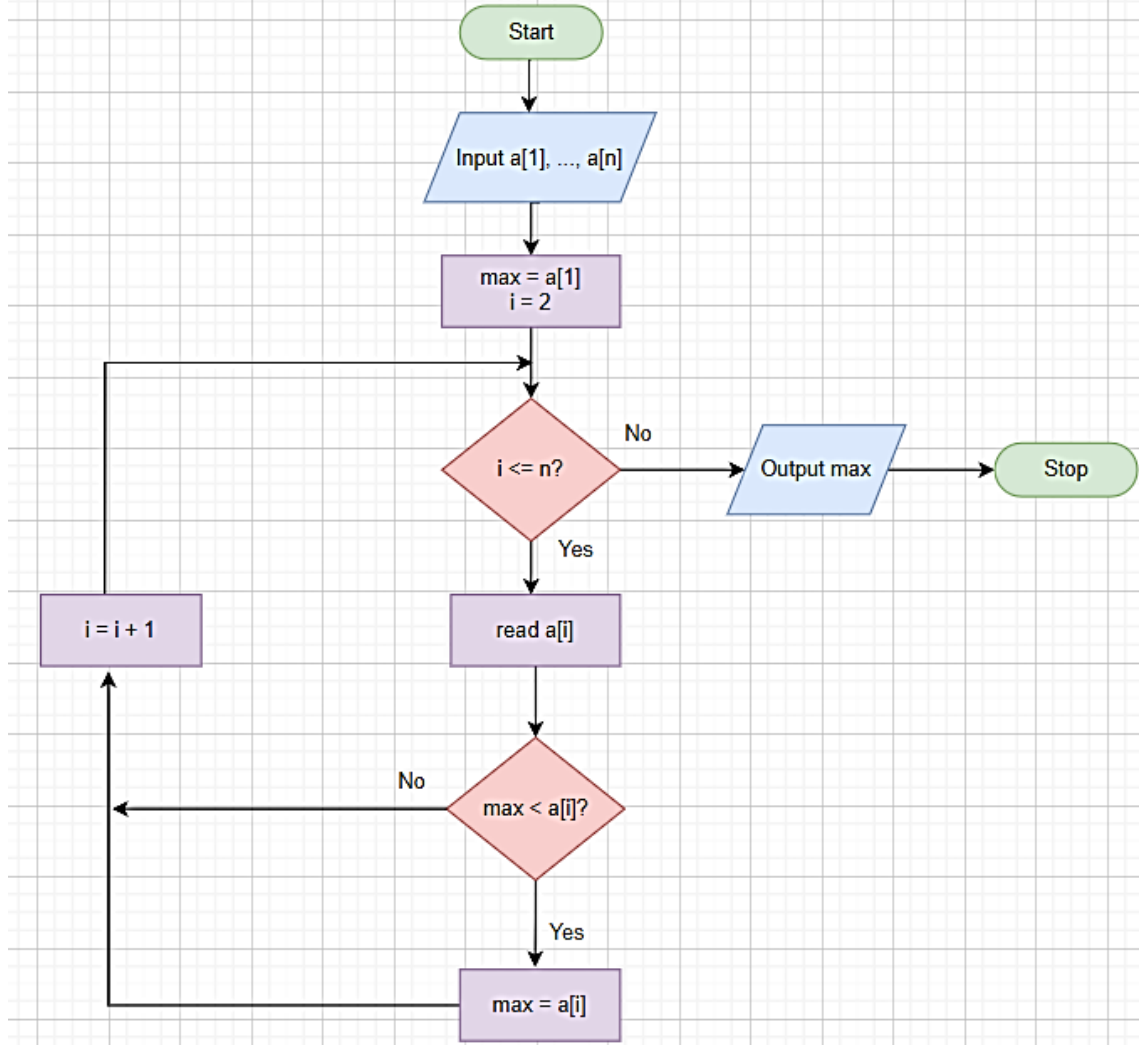
## EXAMPLE 1:

Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

- ▶ Set the temporary maximum equal to the first integer in the sequence. (The temporary maximum will be the largest integer examined at any stage of the procedure.)
- ▶ Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.
- ▶ Repeat the previous step if there are more integers in the sequence.
- ▶ Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

### ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

```
procedure max( $a_1, a_2, \dots, a_n$ : integers)
  max :=  $a_1$ 
  for  $i := 2$  to  $n$ 
    if  $max < a_i$  then  $max := a_i$ 
  return max{max is the largest element}
```



Input:  $arr[] = \{10, 324, 45, 90, 9808\}$ ; *Output: 9808*

Input:  $arr[] = \{20, 10, 100, 20, 4\}$ ; *Output : 100*

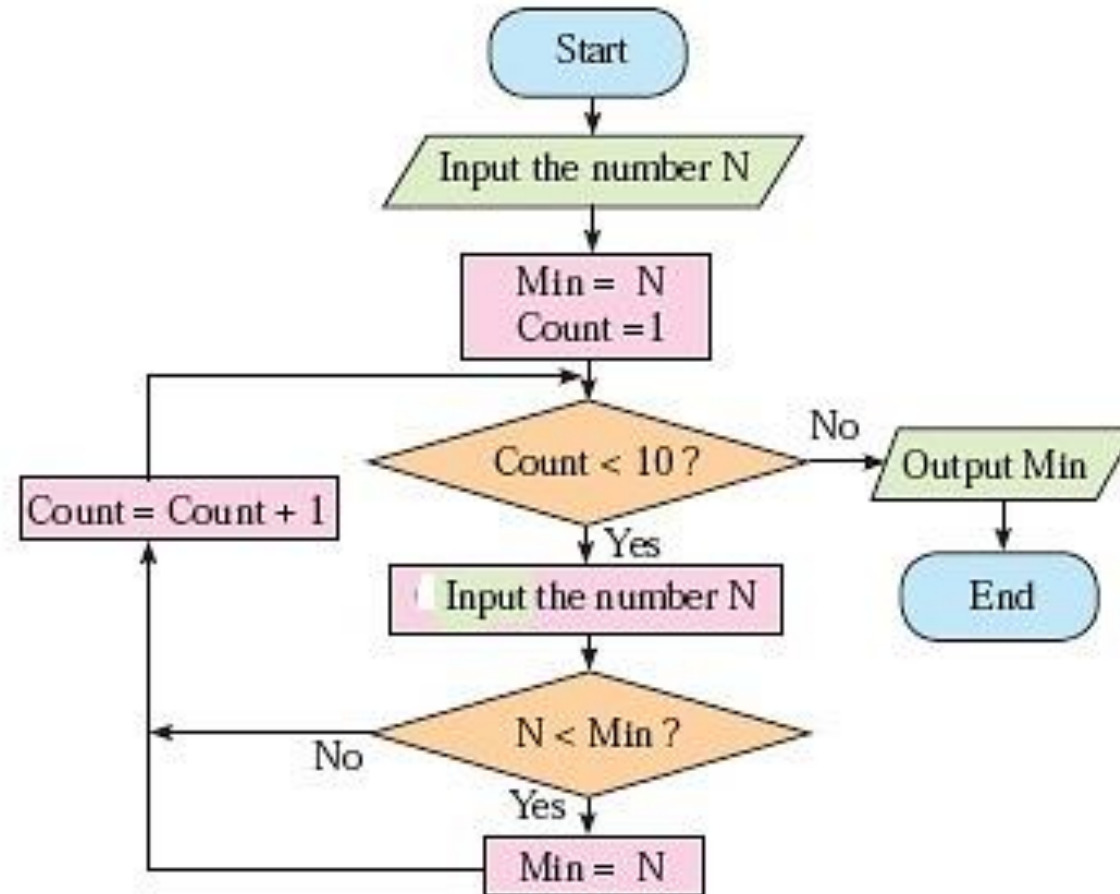
ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

```
procedure  $max(a_1, a_2, \dots, a_n$ : integers)  
   $max := a_1$   
  for  $i := 2$  to  $n$   
    if  $max < a_i$  then  $max := a_i$   
  return  $max$ { $max$  is the largest element}
```

Go through each step of the given algorithm to find the max value in array (show every step).



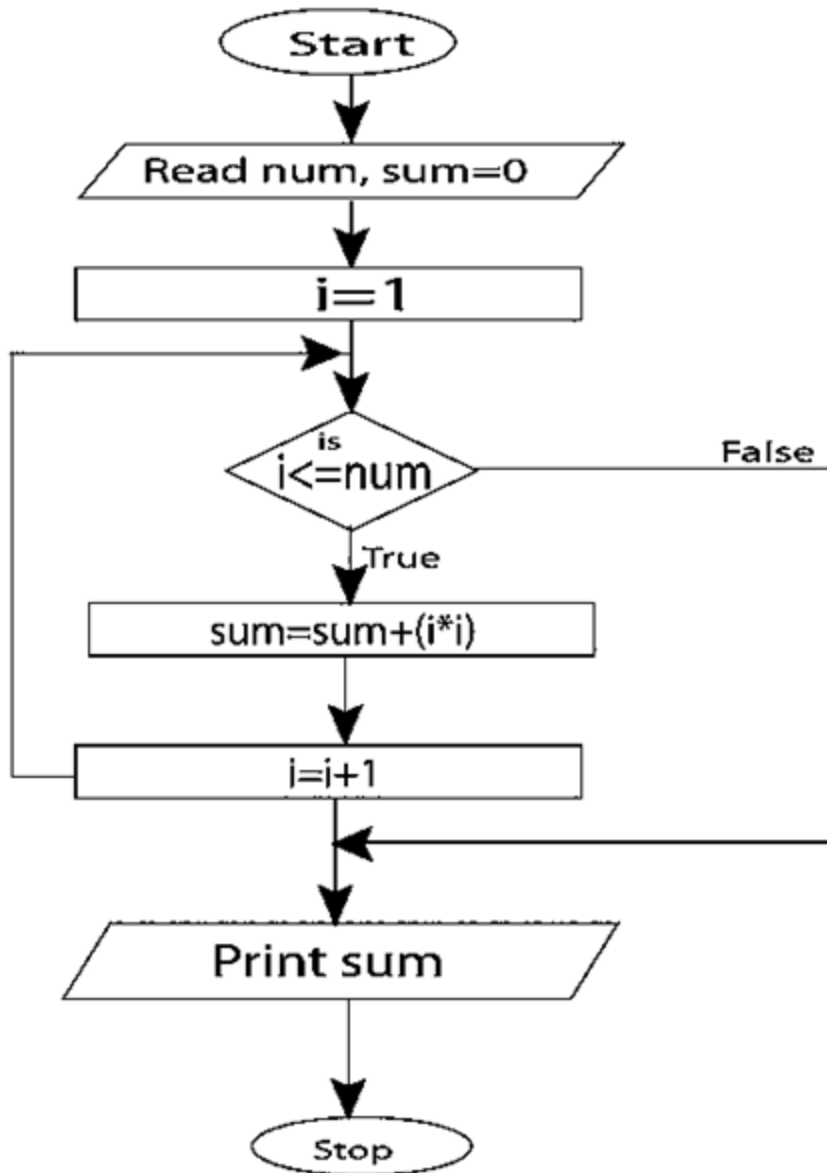
# Flowchart to find if $N$ is a minimum value among $n (=10)$ numbers





**PRACTICE**

Create a flowchart, where you have to build a program: sum of square of  $n$  natural numbers.



# Pseudocode (Appendix 3)

- ▶ **procedure** algorithm name (list \* description of input variables)
- ▶ {comments}
- ▶ variable := expression
- ▶ **if** condition **then** statement or block of statements
- ▶ **if** condition **then** statement 1  
**else** statement 2
- ▶ **if** condition 1 **then** statement 1  
**else if** condition 2 **then** statement 2  
**else if** condition 3 **then** statement 3 ...
- ▶ **for** *variable := initial value to final value*  
block of statements
- ▶ **while** condition  
statement or block of statements
- ▶ **return** output of algorithm



► In computer science, pseudocode is a plain language description of the steps in an algorithm or another system. Pseudocode often uses structural conventions of a normal programming language but is intended for human reading rather than machine reading.

Algorithm	Pseudocode
<pre>{   For i := 1 to n do   {     j := 1;     For k := i+1 to n do     If (a[k] &lt; a[j]) then       j := k;     t := a[i];     a[i] := a[j];     a[j] := t   } }</pre>	<pre>For i = 1 to n do {   Examine a[i] to a[n]   and suppose   The smallest   element is at a[j];   Interchange a[i] and   a[j]; }</pre>

# Types of Algorithm Problems

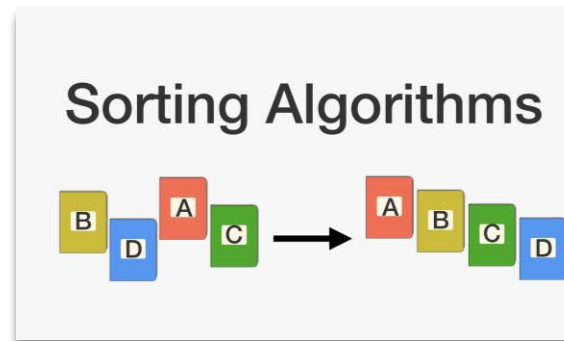
Binary Search

Search 23	0	1	2	3	4	5	6	7	8	9
	2	5	8	12	16	23	38	56	72	91
23 > 16 take 2 <sup>nd</sup> half	L=0	1	2	3	M=4	5	6	7	8	H=9
	2	5	8	12	16	23	38	56	72	91
23 > 56 take 1 <sup>st</sup> half	0	1	2	3	4	L=5	6	M=7	8	H=9
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5	0	1	2	3	4	L=5, M=5	H=6	7	8	9
	2	5	8	12	16	23	38	56	72	91

EG

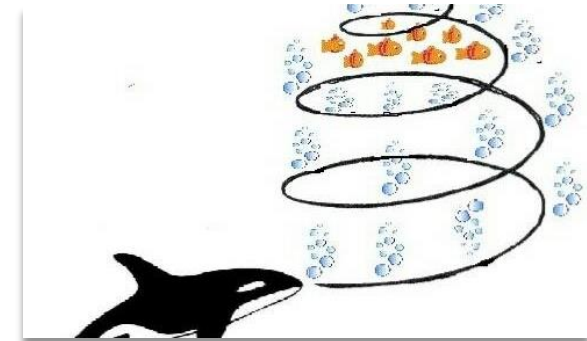
## Searching

Looking through a list to find something that needs a specific characteristic



## Sorting

Sort things from least to greatest or greatest to least or biggest to smallest



## Optimization

Looking for the least or the greatest

## Facial Recognition

Facial recognition is the mechanics behind iPhone logins and Snapchat filters, and it runs on an algorithm. The system works by using a biometrics map to plot facial features from a photo or video. It then takes this information and compares it to a known database of faces to find a match. This is how it can verify identity. When it is just used for Instagram or Snapchat filters, there is no database search. It simply makes a biometric map of the face and applies the filter to it.

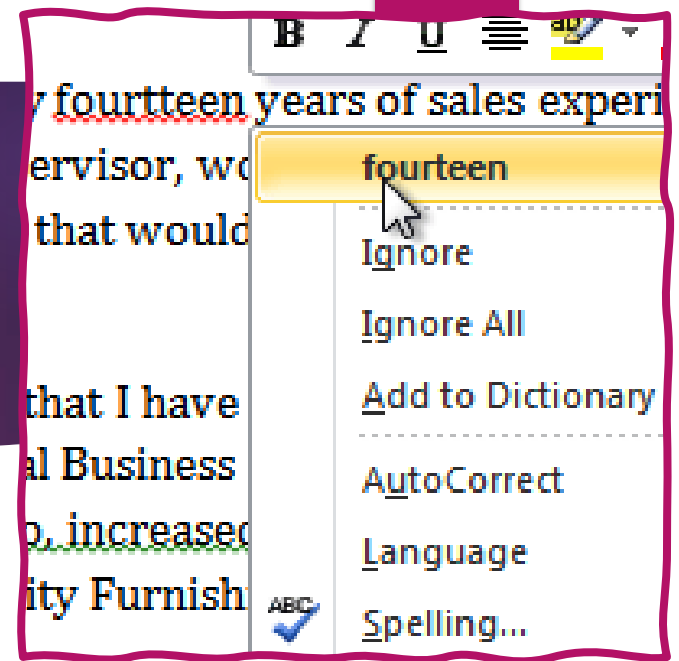
## Recipes

Just like sorting papers and even tying your shoes, following a recipe is a type of algorithm. The goal of course being to create a duplicated outcome. In order to complete a recipe, you have to follow a given set of steps. Say you are making bread. You need flour, yeast and water. After you have your ingredients, you need to combine them in a certain way that will create a predictable outcome, in this case a loaf of bread.



**Sorting Papers, Traffic Signals, Bus Schedules, GPS, Google Search, Facebook, etc.**

# Searching Algorithms

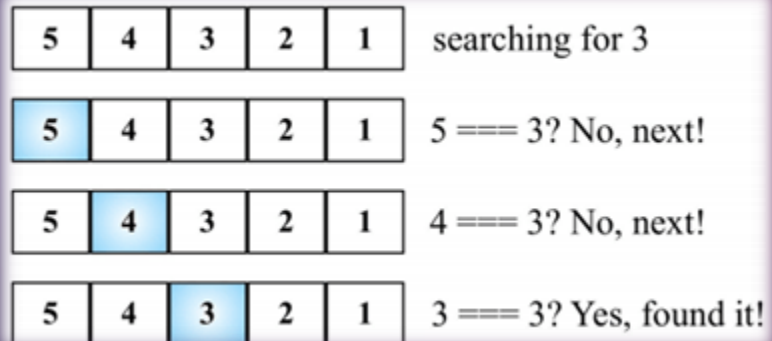


The goal is to locate an element  $x$  in the list of distinct elements or determine that it is not in the list.

The solution is the location of the term that equals  $x$  or 0 if  $x$  is not in the list.

Example: **Spell check**  
essentially looks through all items in the list, compares it to the dictionary and then the output: it is going to put squiggly little red line under something that is not spelled correctly, or it is not going to do anything if everything is spelled correctly

# Linear Search



- ▶ This algorithm checks each term in the sequence, in order, with the desired term,  $x$ .
- ▶ Let  $x = 9$

$$\begin{array}{cccc} & a_2 & a_4 & a_6 \\ 4, & 7, & 3, & 2, & 1, & 0, & 9 \\ a_1 & a_3 & a_5 & a_7 \end{array}$$

Output: 7

## ALGORITHM 2 The Linear Search Algorithm.

```
procedure linear search( $x$ : integer,  $a_1, a_2, \dots, a_n$ : distinct integers)
 $i := 1$ 
while ( $i \leq n$  and  $x \neq a_i$ )
     $i := i + 1$ 
if  $i \leq n$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript of the term that equals  $x$ , or is 0 if  $x$  is not found}
```

Trace the linear search algorithm to look for 12 in the list: 4, 7, 2, 1, 9, 11, 12, 8, 5

$a_1$   $a_3$   $a_5$   $a_7$   $a_9$   
 $a_2$   $a_4$   $a_6$   $a_8$

$x = 12$

$i = 1:$	$1 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 4?$ ✓ <small>TRUE</small>
$i = 2:$	$2 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 7?$ ✓ <small>TRUE</small>
$i = 3:$	$3 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 2?$ ✓ <small>TRUE</small>
$i = 4:$	$4 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 1?$ ✓ <small>TRUE</small>
$i = 5:$	$5 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 9?$ ✓ <small>TRUE</small>
$i = 6:$	$6 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 11?$ ✓ <small>TRUE</small>
$i = 7:$	$7 \leq 9?$ ✓ <small>TRUE</small>	$12 \neq 12?$ ✗ <small>FALSE</small>

```
procedure linear search (x: integer, a1, a2, ..., an: distinct integers)
  i := 1
  while ( i ≤ n and x ≠ ai) // if you haven't reached the end of the list and haven't found x
    i := i + 1 // increase i by 1 to move to the next number in the list

  if i ≤ n then location := i // if you haven't reached the end of the list, then you found x in location i
  else location := 0 // x wasn't found in the list, so we use location 0 to designate that it wasn't found
  return location
```

pseudocode from Discrete Mathematics and Its Applications, Rosen, 7e, McGraw-Hill















$7 \leq 9?$  ✓  
TRUE      *location = 7*






Trace the linear search algorithm to look for 3 in the list:

4, 7, 2, 1, 9, 11, 12, 8, 5

$a_1$   $a_3$   $a_5$   $a_7$   $a_9$   
 $a_2$   $a_4$   $a_6$   $a_8$

$x = 3$

$i = 1:$	$1 \leq 9?$	 <small>TRUE</small>	$3 \neq 4?$	 <small>TRUE</small>
$i = 2:$	$2 \leq 9?$	 <small>TRUE</small>	$3 \neq 7?$	 <small>TRUE</small>
$i = 3:$	$3 \leq 9?$	 <small>TRUE</small>	$3 \neq 2?$	 <small>TRUE</small>
$i = 4:$	$4 \leq 9?$	 <small>TRUE</small>	$3 \neq 1?$	 <small>TRUE</small>
$i = 5:$	$5 \leq 9?$	 <small>TRUE</small>	$3 \neq 9?$	 <small>TRUE</small>
$i = 6:$	$6 \leq 9?$	 <small>TRUE</small>	$3 \neq 11?$	 <small>TRUE</small>
$i = 7:$	$7 \leq 9?$	 <small>TRUE</small>	$3 \neq 12?$	 <small>TRUE</small>

$i = 8:$	$8 \leq 9?$	 <small>TRUE</small>	$3 \neq 8?$	 <small>TRUE</small>
$i = 9:$	$9 \leq 9?$	 <small>TRUE</small>	$3 \neq 5?$	 <small>TRUE</small>
$i = 10:$	$10 \leq 9?$	 <small>FALSE</small>		
	$10 \leq 9?$		<span style="border: 1px solid purple; padding: 2px;"><math>location = 0</math></span>	

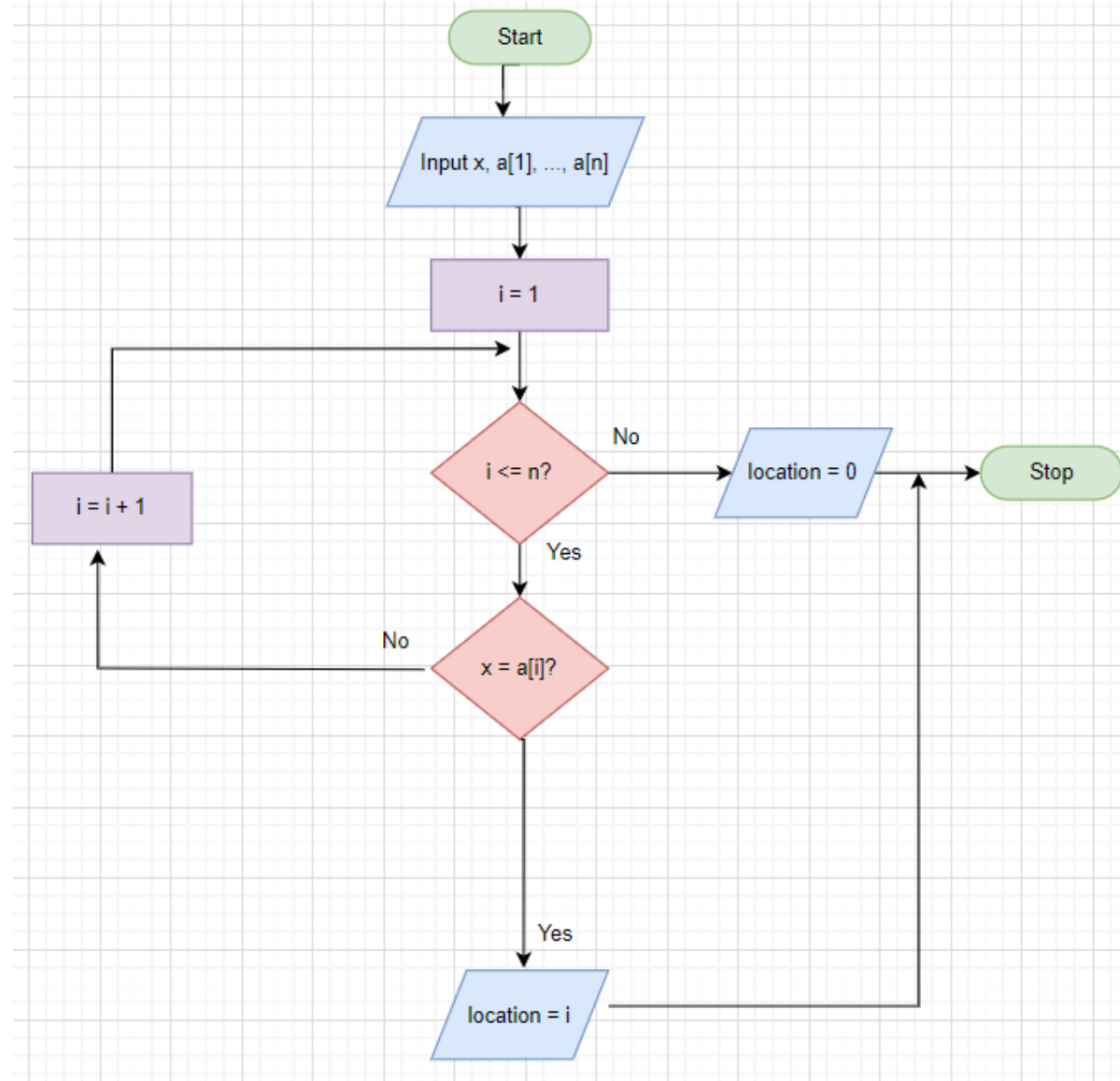
**ALGORITHM 2** The Linear Search Algorithm.

```

procedure linear_search( $x$ : integer,  $a_1, a_2, \dots, a_n$ : distinct integers)
 $i := 1$ 
while ( $i \leq n$  and  $x \neq a_i$ )
     $i := i + 1$ 
if  $i \leq n$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript of the term that equals  $x$ , or is 0 if  $x$  is not found}
    
```



# Flowchart for a Linear Search





**PRACTICE**  
*on your*  
**OWN**

Given a list of array:

22, 11, 5, 6, 7, 3, 9, 1

**Using a Linear Search Algorithm**

- a) Find the location of  $x$ , where  $x = 3$
- b) Find the location of  $x$ , where  $x = 10$
- c) Create a flowchart for a given problem.

# Binary Search

- ▶ In a list of items in increasing order, the binary search algorithm begins by comparing the desired element  $x$  with the middle value.
- ▶ The algorithm then chooses the upper or lower half of the data by comparing the desired value with the middle value.
- ▶ The process continues until a list of size 1 is found.

▶ Let  $x = 9$

$a_2$     $a_4$     $a_6$   
~~0, 1, 2, 3, 4, 7, 9~~  
 $a_1$     $a_3$     $a_5$     $a_7$

$$\frac{1 + 7}{2} = 4 \rightarrow a_4$$

$3 < 9 \rightarrow$  works only with the right side of the list

$$\frac{5 + 7}{2} = 6 \rightarrow a_6$$

$7 < 9 \rightarrow$  works only with the right side of the list

Output: 7

## ALGORITHM 3 The Binary Search Algorithm.

```
procedure binary search ( $x$ : integer,  $a_1, a_2, \dots, a_n$ : increasing integers)
 $i := 1$  { $i$  is left endpoint of search interval}
 $j := n$  { $j$  is right endpoint of search interval}
while  $i < j$ 
     $m := \lfloor (i + j) / 2 \rfloor$ 
    if  $x > a_m$  then  $i := m + 1$ 
    else  $j := m$ 
if  $x = a_i$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript  $i$  of the term  $a_i$  equal to  $x$ , or 0 if  $x$  is not found}
```

Search for the number 8 in the list using the binary search algorithm :

4, 7, 2, 1, 9, 11, 12, 8, 5



1, 2, 4, 5, 7, 8, 9, 11, 12

$i = 1$                        $j = 9$

$x = 8$

1 < 9? TRUE

$$m = \lfloor \frac{1+9}{2} \rfloor = 5$$

8 > 7? TRUE

$i = 6, \quad j = 9$

6 < 9? TRUE

$$m = \lfloor \frac{6+9}{2} \rfloor = 7$$

8 > 9? FALSE

$i = 6, \quad j = 7$

6 < 7? TRUE

$$m = \lfloor \frac{6+7}{2} \rfloor = 6$$

8 > 8? FALSE

$i = 6, \quad j = 6$

6 < 6? FALSE

8 = 8? TRUE

*location = 6*

Floor function used for m value in binary search:

$$\lfloor \_ 4 \_ \rfloor = 4$$

$$\lfloor \_ 4.01 \_ \rfloor = 4$$

$$\lfloor \_ 4.99 \_ \rfloor = 4$$

```

procedure binary search (x: integer, a1, a2, ..., an: incre
i := 1 // i is the left endpoint of the search interval
j := n // j is the right endpoint of the search interval
while i < j // as long as you are looking at a list of more than
    m := ⌊ (i + j)/2 ⌋ // m becomes the location of the mid

    if x > am then i := m + 1 // if x is bigger than the number i
    else j := m // otherwise, move the top down

if x = ai then location := i // if your list of 1 matches x, then m
else location := 0 // x wasn't found in the list, so we use locati
return location
    
```

pseudocode from Discrete Mathematic

Search for *the number 3* in the list using the **binary search algorithm** :

4, 7, 2, 1, 9, 11, 12, 8, 5




1, 2, 4, 5, 7, 8, 9, 11, 12


$i = 1$

$j = 9$


$x = 3$

$1 < 9?$   TRUE


$$m = \lfloor \frac{1+9}{2} \rfloor = 5$$

$3 > 7?$   FALSE


$i = 1, \quad j = 5$

$1 < 5?$   TRUE


$$m = \lfloor \frac{1+5}{2} \rfloor = 3$$

$3 > 4?$   FALSE


$i = 1, \quad j = 3$


$1 < 3?$   TRUE

$$m = \lfloor \frac{1+3}{2} \rfloor = 2$$

$3 > 2?$   TRUE

$i = 3, \quad j = 3$

$3 < 3?$   FALSE

$3 = 4?$   FALSE

$location = 0$

### ALGORITHM 3 The Binary Search Algorithm.

```

procedure binary search ( $x$ : integer,  $a_1, a_2, \dots, a_n$ : increasing integers)
 $i := 1$  { $i$  is left endpoint of search interval}
 $j := n$  { $j$  is right endpoint of search interval}
while  $i < j$ 
     $m := \lfloor (i + j)/2 \rfloor$ 
    if  $x > a_m$  then  $i := m + 1$ 
    else  $j := m$ 
if  $x = a_i$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript  $i$  of the term  $a_i$  equal to  $x$ , or 0 if  $x$  is not found}
    
```

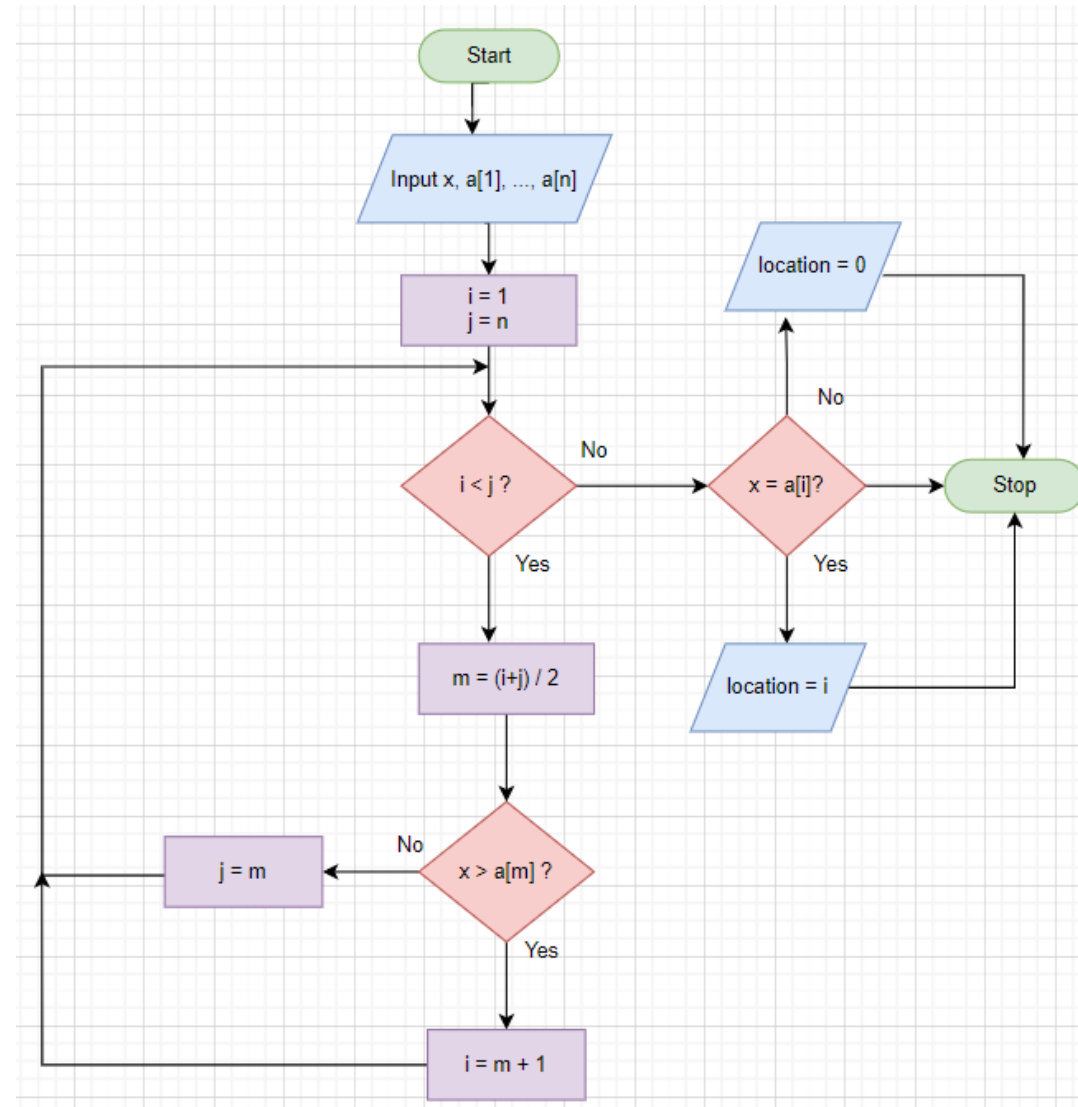
Floor function used for  $m$  value in binary search:

$$\lfloor \_ 4 \_ \rfloor = 4$$

$$\lfloor \_ 4.01 \_ \rfloor = 4$$

$$\lfloor \_ 4.99 \_ \rfloor = 4$$

# Flowchart for a Binary Search



**PRACTICE**  
*on your*  
**OWN**

Given a list of array:

22, 11, 5, 6, 7, 3, 9, 1

**Using a Binary Search Algorithm**

- a) Find the location of  $x$ , where  $x = 3$
- b) Find the location of  $x$ , where  $x = 10$
- c) Create a flowchart for a given problem.

# More Practice



Practice  
Makes  
Perfect

- ▶ Use linear and binary searching algorithms to locate  $n = 22$  and  $n = 5$  in set A.

$$A = \{0, 2, 4, 11, 22, 8, 14, 9, 27, 30\}$$

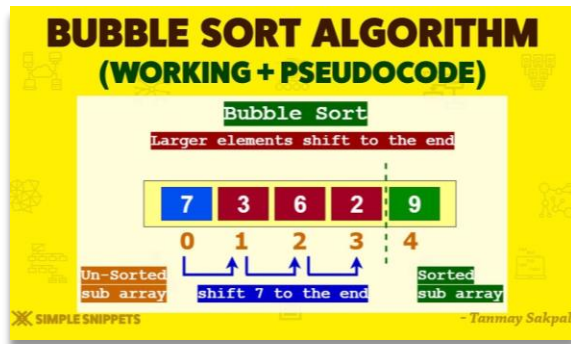
# Sorting Algorithms

An algorithm that sorts the elements of a list into increasing order (numerical, alphabetical, etc.)

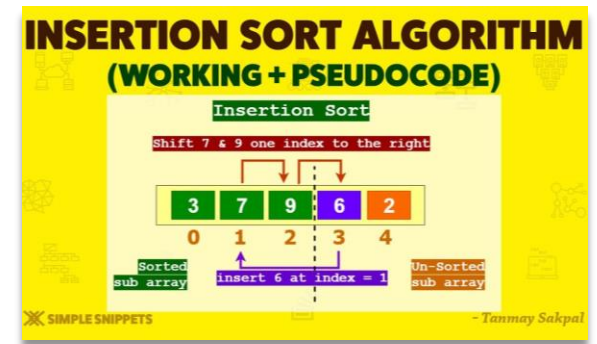
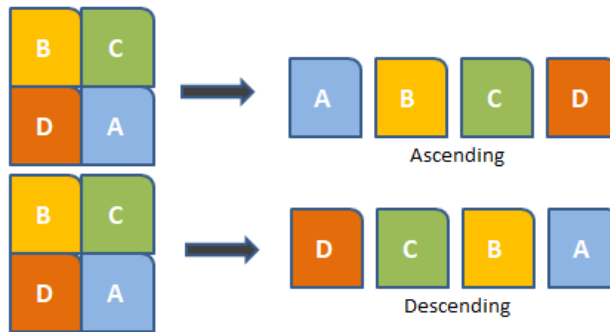
- ▶ Used often for large databases like sorting customer or part numbers, prices, etc.
- ▶ Phone directories (by last name)
- ▶ Over 100 sorting algorithms have been devised



# Sorting Algorithms



Bubble Sort



Insertion Sort

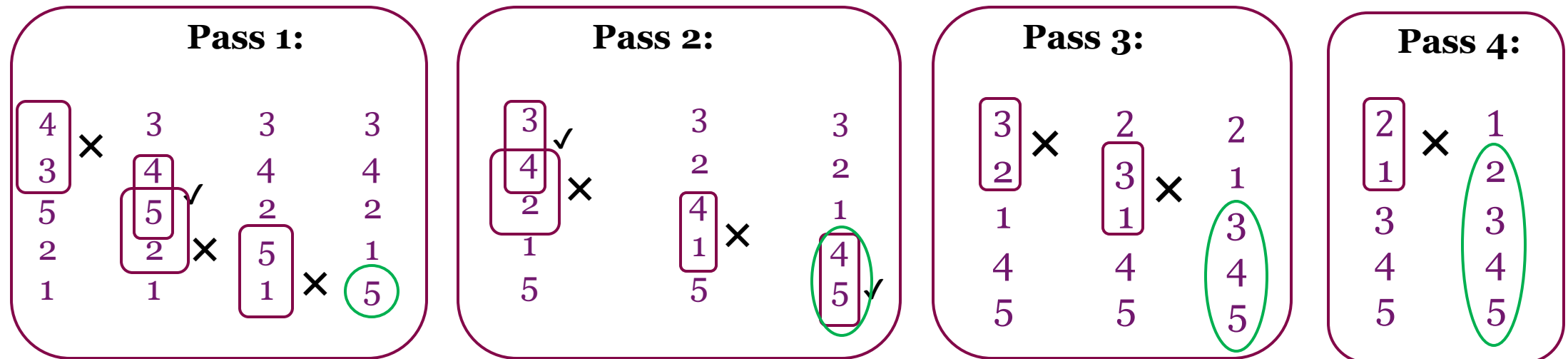
# Bubble Sort Algorithm

An example of a computer algorithm is bubble sort. This is a simple algorithm used for taking a list of jumbled up numbers and putting them into the correct order. The algorithm runs as follows:

- ▶ Look at the first number in the list.
- ▶ Compare the current number with the next number.
- ▶ Is the next number smaller than the current number? If so, swap the two numbers around. If not, do not swap.
- ▶ Move to the next number along in the list and make this the current number.
- ▶ Repeat from step 2 until the last number in the list has been reached.
- ▶ If any numbers were swapped, repeat again from step 1.
- ▶ If the end of the list is reached *without any swaps being made*, then the list is ordered, and the **algorithm can stop**.

# Bubble Sort Algorithm

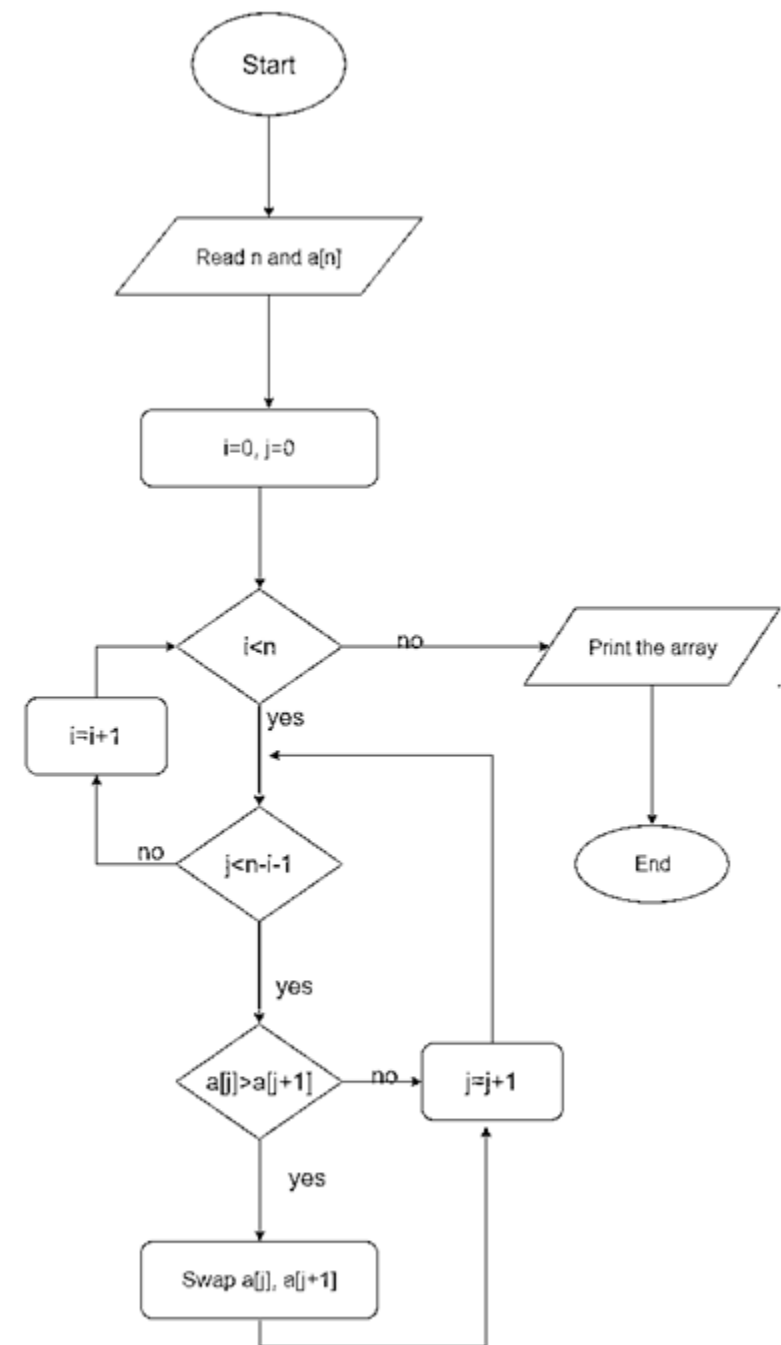
- ▶ A simple algorithm that successively compares adjacent elements, interchanging them if they are in the wrong order. This may require several passes.



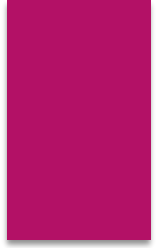
#### ALGORITHM 4 The Bubble Sort.

```
procedure bubblesort( $a_1, \dots, a_n$  : real numbers with  $n \geq 2$ )  
  for  $i := 1$  to  $n - 1$   
    for  $j := 1$  to  $n - i$   
      if  $a_j > a_{j+1}$  then interchange  $a_j$  and  $a_{j+1}$   
  { $a_1, \dots, a_n$  is in increasing order}
```

5 4 3 1 2 | 




5, 1, 4, 2, 8





0, 5, 7, 6, 9, 4, 3, 2





Sort the list using Bubble Sort: 3, 1, 7, 5, 2


3, 1, 7, 5, 2  



1, 3, 7, 5, 2  



1, 3, 7, 5, 2  



1, 3, 5, 7, 2  



1, 3, 5, 2, 7  



1, 3, 5, 2, 7  



1, 3, 5, 2, 7  



1, 3, 5, 2, 7  



1, 3, 2, 5, 7  


1, 3, 2, 5, 7  


1, 3, 2, 5, 7  


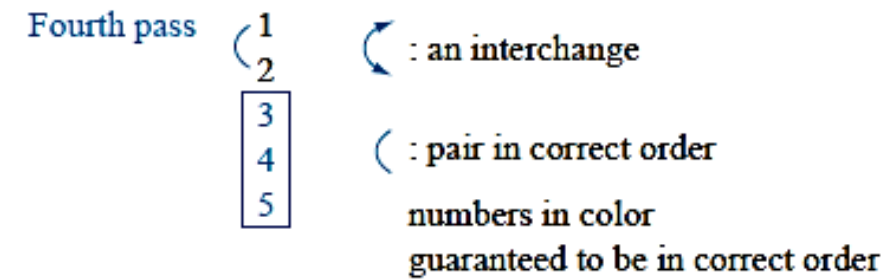
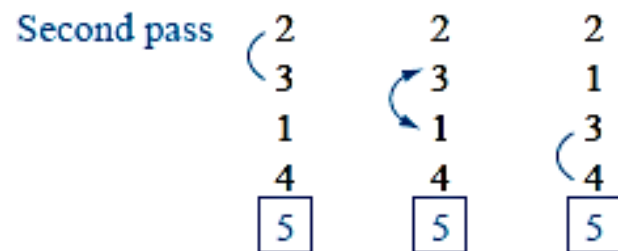
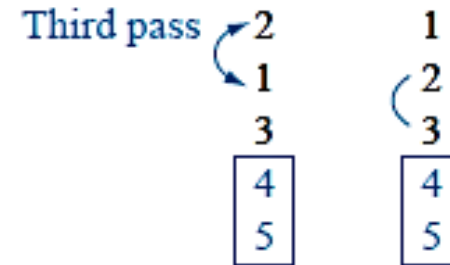
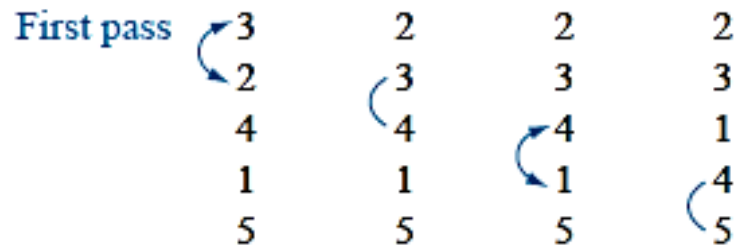
1, 2, 3, 5, 7  


1, 2, 3, 5, 7  


1, 2, 3, 5, 7  


# Try!

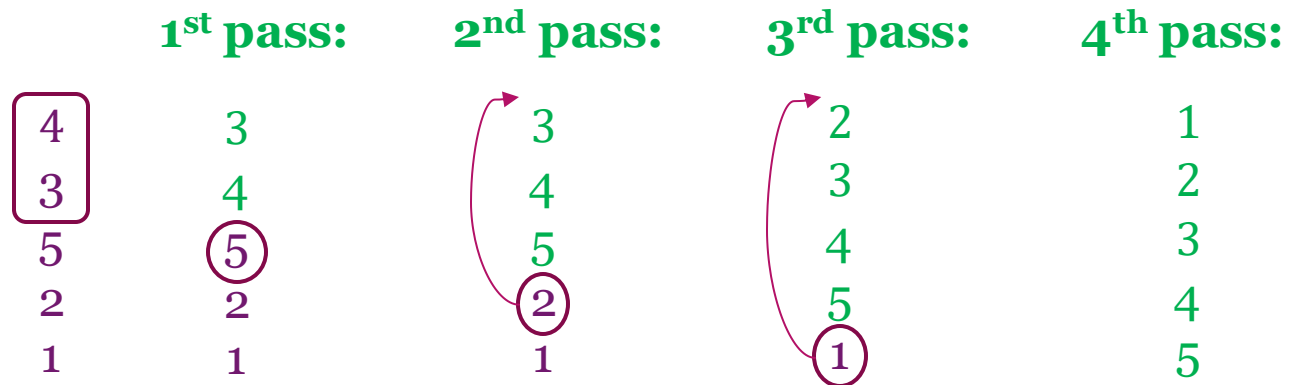
- Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order.





# Insertion Sort Algorithm

- ▶ Another simple sorting algorithm that begins with the second element, comparing it to the first and ordering it appropriately. The third element is then compared with the first and, if necessary, the second to order it. This pattern repeats.



### ALGORITHM 5 The Insertion Sort.

**procedure** *insertion sort*( $a_1, a_2, \dots, a_n$ : real numbers with  $n \geq 2$ )

**for**  $j := 2$  **to**  $n$

$i := 1$

**while**  $a_j > a_i$

$i := i + 1$

$m := a_j$

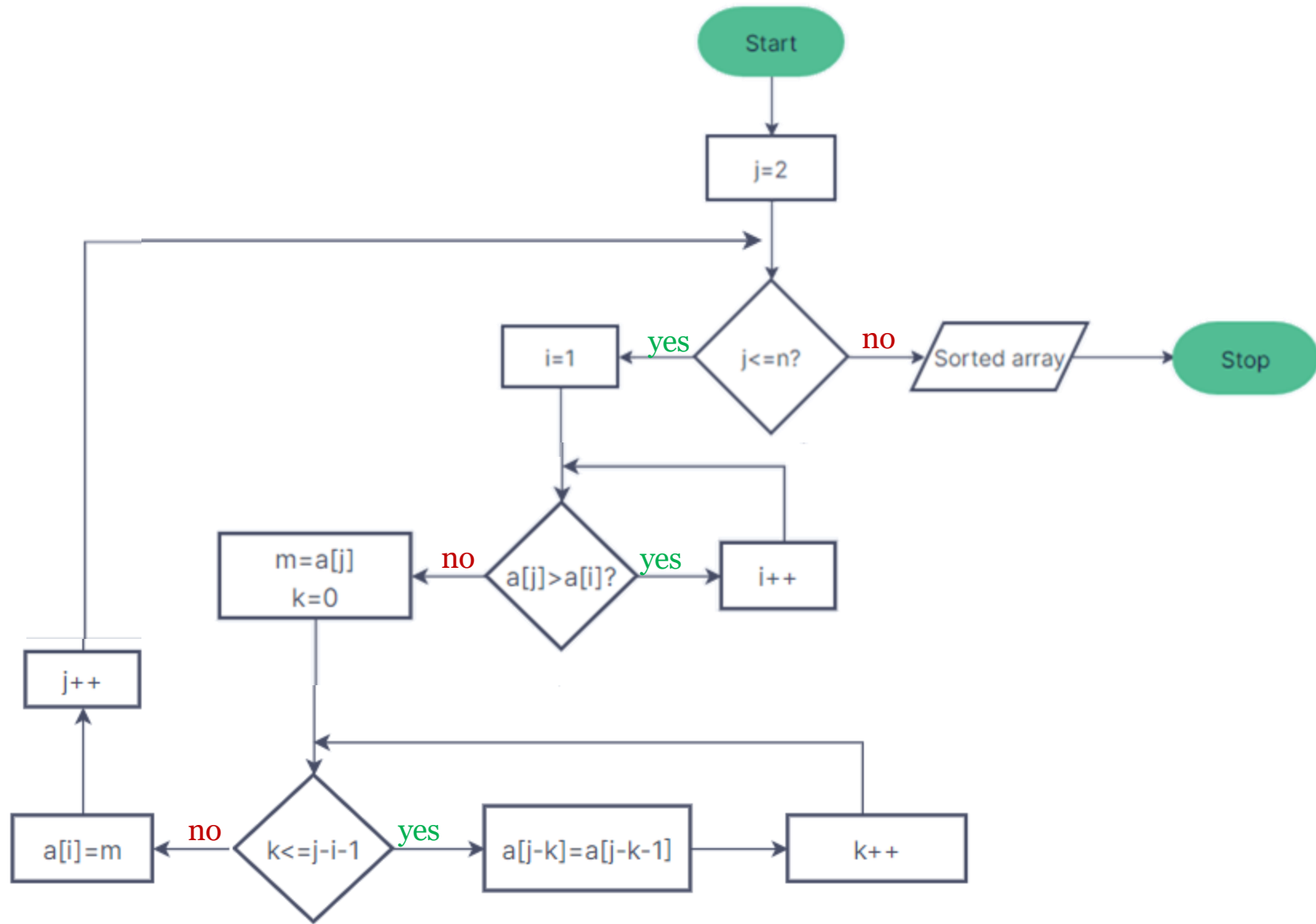
**for**  $k := 0$  **to**  $j - i - 1$

$a_{j-k} := a_{j-k-1}$

$a_i := m$

{ $a_1, \dots, a_n$  is in increasing order}

6 5 3 1 8 7 2 4



# Insertion Sort Example



Sorted Array



Unsorted Array

## Insertion Sort in C++



- Since,  $3 < 5$



- Since,  $1 < 3$



- 9 is at correct position now



- Since,  $8 < 9$



- Since,  $2 < 3$



- Since,  $4 < 5$



- Since,  $7 < 8$



Traverse leftwards wherever you find the first greater item insert before that

3 gets inserted before 5  
5 moves 1 position rightwards

1 gets inserted before 3  
3, 5 each move 1 position rightwards

No Insertion needed or  
No rightward movement needed

8 gets inserted after 5  
9 moves 1 position rightwards

2 gets inserted after 1  
3 to 9 each moves 1 position rightwards

4 gets inserted after 3  
5, 8, 9 each moves 1 position rightwards

7 gets inserted after 5  
8, 9 each moves 1 position rightwards

Final Sorted Array

7, 4, 5, 2

Sort the list using Insertion Sort: 3, 1, 7, 5, 2

sorted          unsorted  
3, 1, 7, 5, 2

1, 3, 7, 5, 2

1, 3, 7, 5, 2

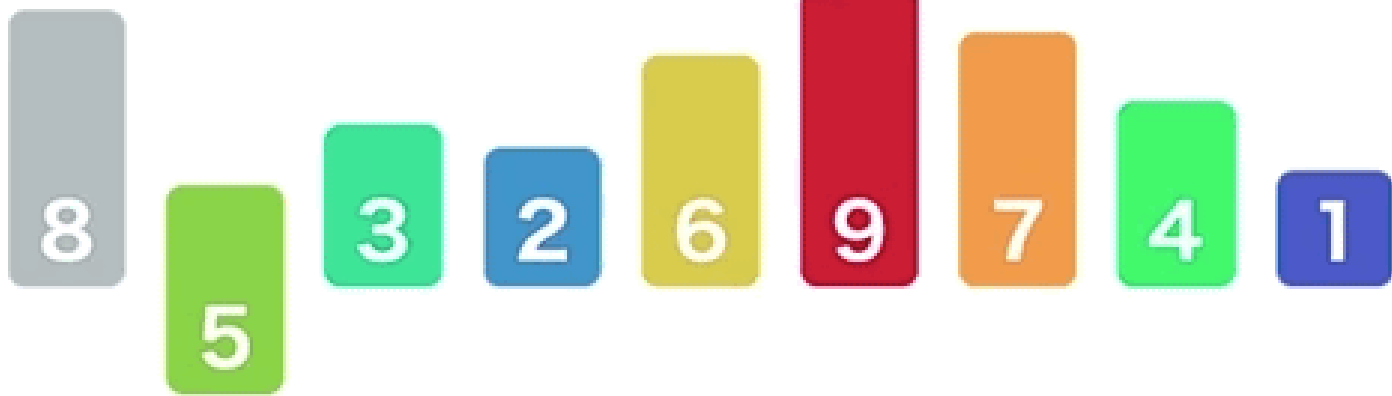
1, 3, 5, 7, 2

1, 2, 3, 5, 7

#### ALGORITHM 5 The Insertion Sort.

```
procedure insertion sort( $a_1, a_2, \dots, a_n$ : real numbers with  $n \geq 2$ )  
for  $j := 2$  to  $n$   
     $i := 1$   
    while  $a_j > a_i$   
         $i := i + 1$   
     $m := a_j$   
    for  $k := 0$  to  $j - i - 1$   
         $a_{j-k} := a_{j-k-1}$   
     $a_i := m$   
{ $a_1, \dots, a_n$  is in increasing order}
```

- $m$  temporarily store the number you are inserting in
- at the end after sliding numbers bigger than  $a[j]$  down 1 spot, there will be an empty value at  $a[i]$ 's location (as values move upwards, value of  $a[j]$  will be wiped at the very beginning), where  $m$  (wiped  $a[j]$  value) will be inserted.



# Practice

- ▶ Use the Bubble and Insertion Sorting algorithms to sort the elements of set A.

$$A = \{42, 19, 32, 11, 8, 1\}$$



# Answer: Bubble Sort

- ▶ Use the Bubble and Insertion Sorting algorithms to sort the elements of set A.

$A = \{42, 19, 32, 11, 8, 1\}$

Pass 1:

19, 42, 32, 11, 8, 1



19, 32, 42, 11, 8, 1



19, 32, 11, 42, 8, 1



19, 32, 11, 8, 42, 1



19, 32, 11, 8, 1, 42

Pass 2:

19, 32, 11, 8, 1, 42



19, 11, 32, 8, 1, 42



19, 11, 8, 32, 1, 42



19, 11, 8, 1, 32, 42

Pass 3:

19, 11, 8, 1, 32, 42



11, 19, 8, 1, 32, 42



11, 8, 19, 1, 32, 42



11, 8, 1, 19, 32, 42

Pass 4:

11, 8, 1, 19, 32, 42



8, 11, 1, 19, 32, 42



8, 1, 11, 19, 32, 42

Pass 5:

8, 1, 11, 19, 32, 42



1, 8, 11, 19, 32, 42

# Answer: Insertion Sort

- ▶ Use the Bubble and Insertion Sorting algorithms to sort the elements of set A.

$$A = \{42, 19, 32, 11, 8, 1\}$$

Pass 1:

19, 42, 32, 11, 8, 1

Pass 2:

19, 32, 42, 11, 8, 1

Pass 3:

11, 19, 32, 42, 8, 1

Pass 4:

8, 11, 19, 32, 42, 1

Pass 5:

1, 8, 11, 19, 32, 42

# Try!

- Use the insertion sort to put the elements of the list 3, 2, 4, 1, 5 in increasing order.

3 > 2,            2, 3, 4, 1, 5

4 > 2 & 4 > 3,            2, 3, 4, 1, 5

1 < 2,            1, 2, 3, 4, 5

5 > 4,            1, 2, 3, 4, 5



Try it Out



# Optimization Algorithms: Greedy Algorithms

- ❑ These algorithms makes the “best” choice at each step. We must specify what that “best” choice is.
  - ▶ Finding shortest path between two points
  - ▶ Connect network using least amount of fiber-optic cable
  - ▶ Scheduling
  
- ❑ There are perhaps hundreds of popular optimization algorithms

- ▶ Design a greedy algorithm for making change of n U.S. cents with **quarters**, **dimes**, **nickels** and **pennies**.

## \$0.97 – Making Change

- ▶ At each step, use the largest possible value coin that does not exceed the amount of change left.

\$0.97	Quarter	\$0.97 – \$0.25
\$0.72	Quarter	\$0.72 – \$0.25
\$0.47	Quarter	\$0.47 – \$0.25
\$0.22	Dime	\$0.22 – \$0.10
\$0.12	Dime	\$0.12 – \$0.10
\$0.02	Penny	\$0.02 – \$0.01
\$0.01	Penny	\$0.01 – \$0.01

quarter	→	25 cents
dime	→	10 cents
nickel	→	5 cents
penny	→	1 cent



Make a Change for 397 750 IQR using Greedy Optimization Algorithm.



# The backpack problem

Let's imagine, a thief broke into a jewelry store, in which there are three jewelry weighing 2 kg, 4 kg, and 6 kg cost \$3000, \$5000, \$6000, respectively. That is to say, \$1500/kg, \$1250/kg, \$1000/kg.

The thief can take only 10 kg – that is how much his backpack holds. The thief wants to maximize the profit. What will be the optimal solution for the thief?

$$\begin{aligned} \$5000 + \$6000 &= \$11000 \\ &\text{(10 kg)} \end{aligned}$$



### ALGORITHM 6 Greedy Change-Making Algorithm.

**procedure** *change*( $c_1, c_2, \dots, c_r$ : values of denominations of coins, where

$c_1 > c_2 > \dots > c_r$ ;  $n$ : a positive integer)

**for**  $i := 1$  **to**  $r$

$d_i := 0$  { $d_i$  counts the coins of denomination  $c_i$  used}

**while**  $n \geq c_i$

$d_i := d_i + 1$  {add a coin of denomination  $c_i$ }

$n := n - c_i$

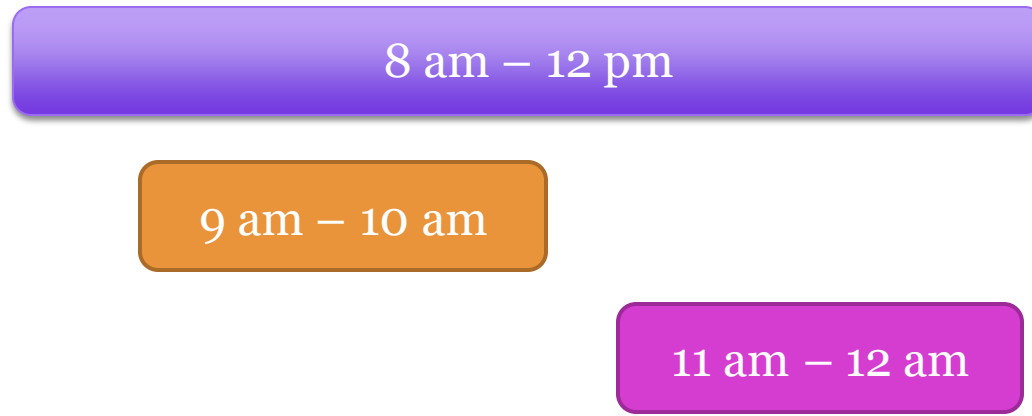
{ $d_i$  is the number of coins of denomination  $c_i$  in the change for  $i = 1, 2, \dots, r$ }

# Greedy Algorithm: Scheduling

- ▶ A greedy algorithm makes the best choice at each step according to a specified criterion. However, it also can be difficult to determine which of many possible criteria to choose:

To use a greedy algorithm to schedule the most talks, that is, an optimal schedule, we need to decide how to choose which talk to add at each step. **There are many criteria** we could use to select a talk at each step, where we chose from the talks that do not overlap talks already selected. For example, we could add talks in order of *earliest start time*, we could add talks in order of *shortest time*, we could add talks in order of *earliest finish time*, or we could use some other criterion.

- ▶ Talk 1 starts at 8 a.m. and ends at 12 noon,
- ▶ Talk 2 starts at 9 a.m. and ends at 10 a.m.,
- ▶ Talk 3 starts at 11 a.m. and ends at 12 noon.



8 am – 12 pm

9 am – 10 am

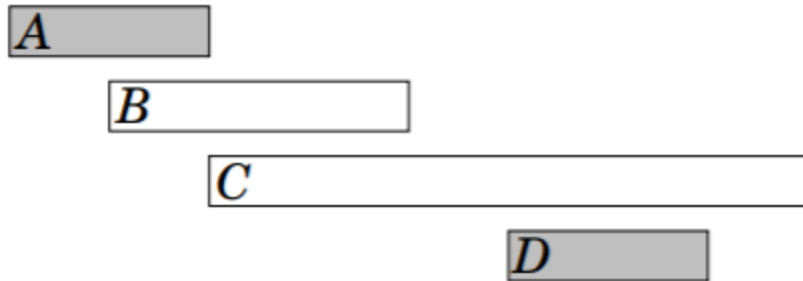
11 am – 12 am

We first select the *Talk 1* because it starts earliest. But once we have selected *Talk 1* we cannot select either *Talk 2* or *Talk 3* because both overlap *Talk 1*. Hence, this greedy algorithm selects only one talk. This is not optimal because we could schedule *Talk 2* and *Talk 3*, which do not overlap.

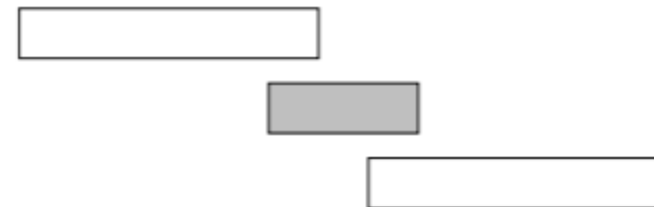
event	starting time	ending time
<i>A</i>	1	3
<i>B</i>	2	5
<i>C</i>	3	9
<i>D</i>	6	8

### Algorithm 1:

•The first idea is to select as short events as possible. In the example, this algorithm selects the following events:



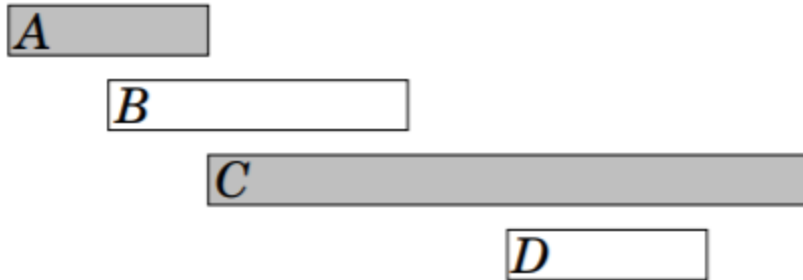
•However, selecting short events is not always a correct strategy. For example, the algorithm fails in the below case:



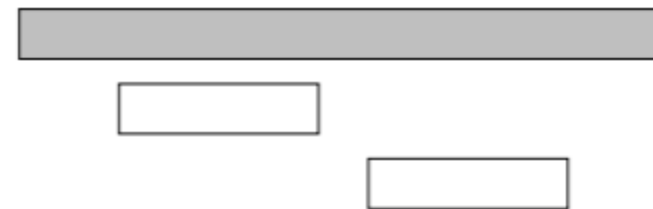
event	starting time	ending time
<i>A</i>	1	3
<i>B</i>	2	5
<i>C</i>	3	9
<i>D</i>	6	8

### Algorithm 2:

•Another idea is to always select the next possible event that begins as early as possible. This algorithm selects the following events:



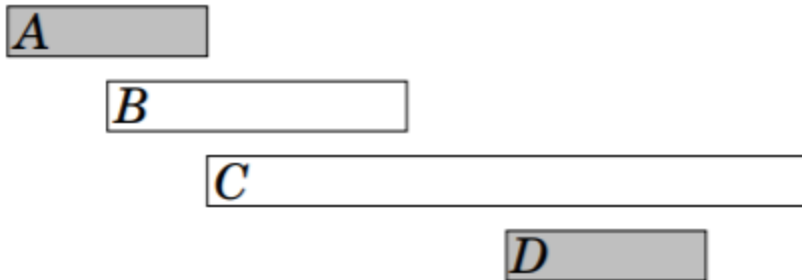
•However, given a counter example for this algorithm. In this case, the algorithm only selects one event:



event	starting time	ending time
<i>A</i>	1	3
<i>B</i>	2	5
<i>C</i>	3	9
<i>D</i>	6	8

### Algorithm 3:

•The third idea is to always select the next possible event that ends as early as possible. This algorithm selects the following events:



•It turns out that this algorithm **always produces an optimal solution**.

•The reason for this is that it is always an optimal choice to first select an event that ends as early as possible.

•After this, it is an optimal choice to select the next event using the same strategy, etc., until any other event can't be selected.

•One way the algorithm works is to consider what happens if first select an event that ends later than the event that ends as early as possible.

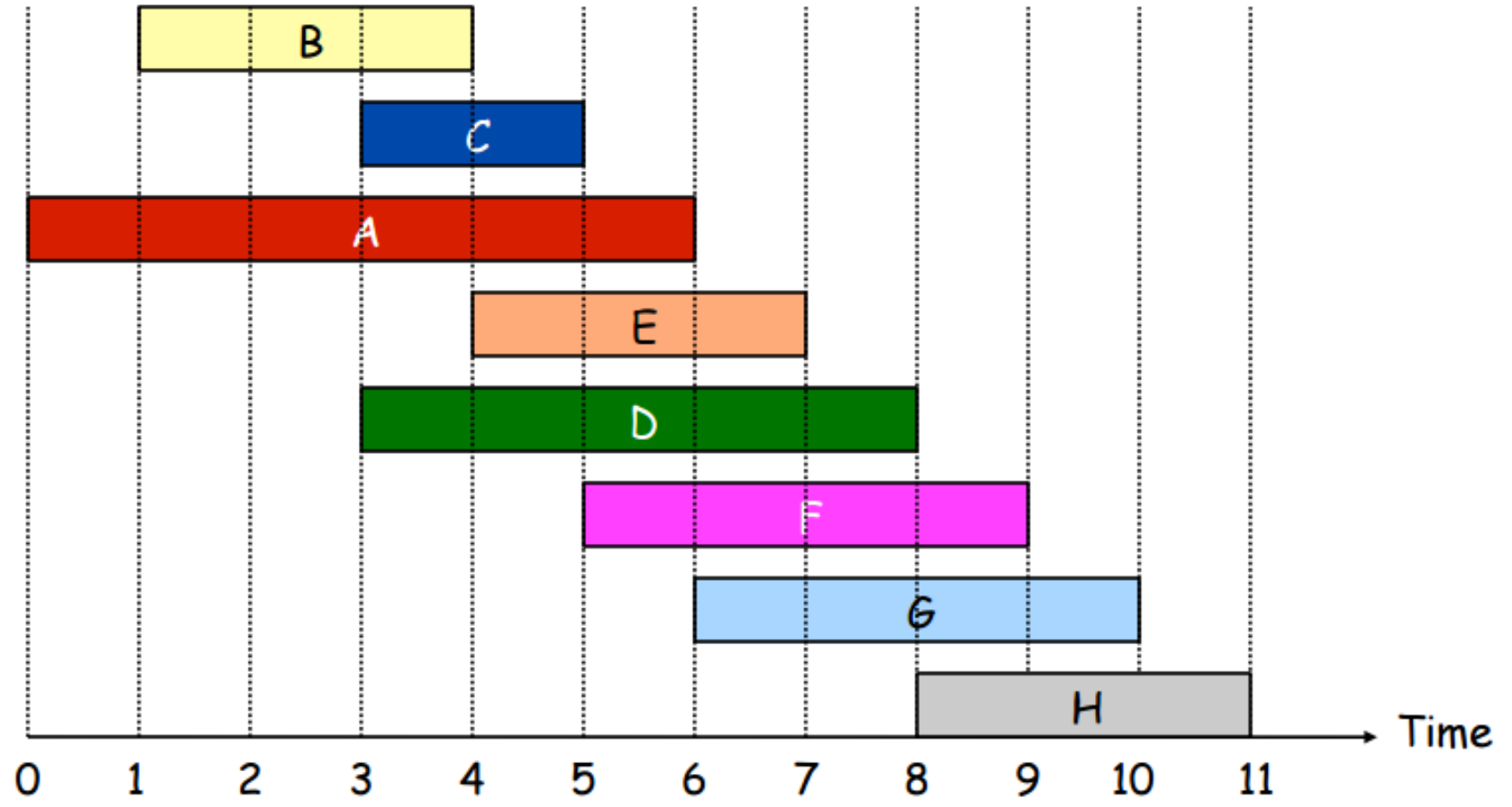
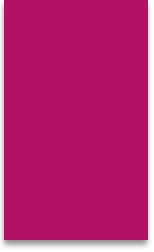
•Now, with having at most an equal number of choices how the next event can be selected.

•Hence, selecting an event that ends later can never yield a better solution, and the greedy algorithm is correct.

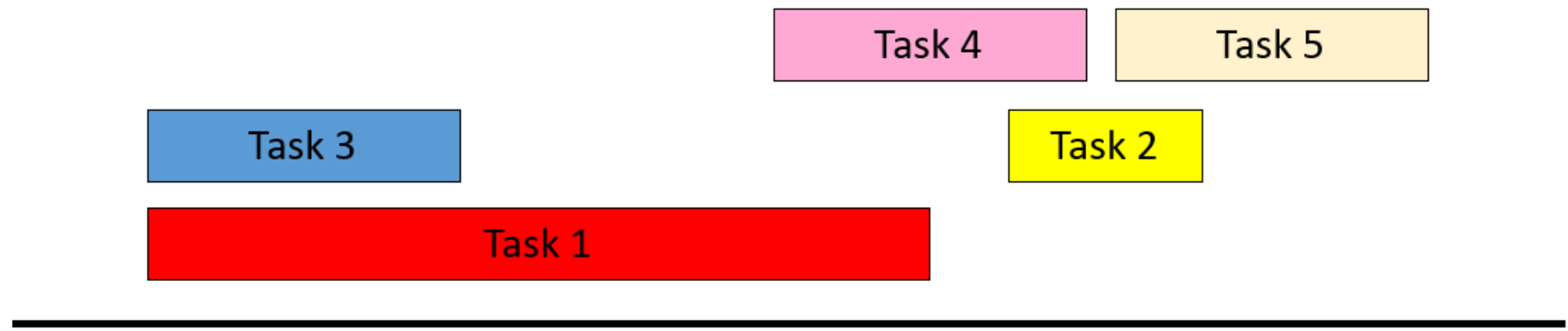
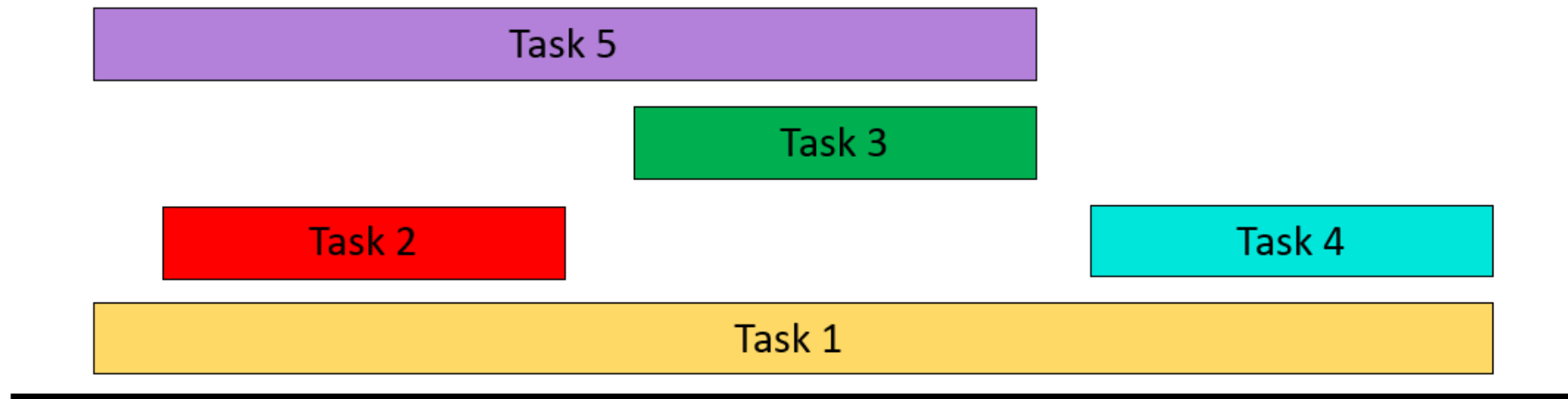


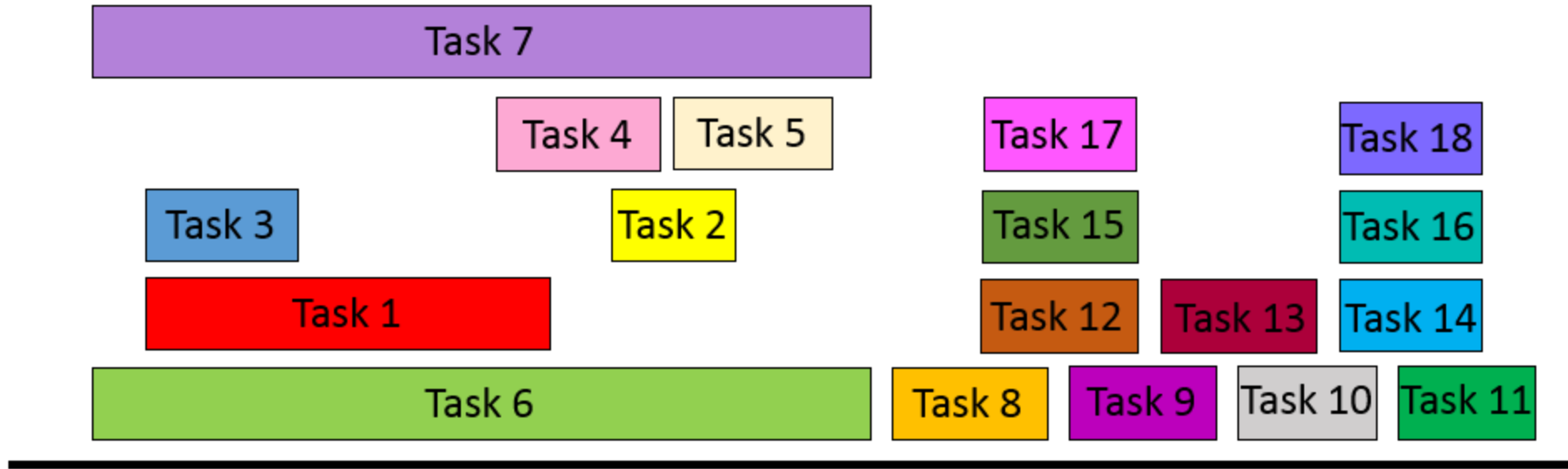
### ALGORITHM 7 Greedy Algorithm for Scheduling Talks.

**procedure** *schedule*( $s_1 \leq s_2 \leq \dots \leq s_n$ : start times of talks,  
 $e_1 \leq e_2 \leq \dots \leq e_n$ : ending times of talks)  
sort talks by finish time and reorder so that  $e_1 \leq e_2 \leq \dots \leq e_n$   
 $S := \emptyset$   
**for**  $j := 1$  **to**  $n$   
    **if** talk  $j$  is compatible with  $S$  **then**  
         $S := S \cup \{\text{talk } j\}$   
**return**  $S$  { $S$  is the set of talks scheduled}







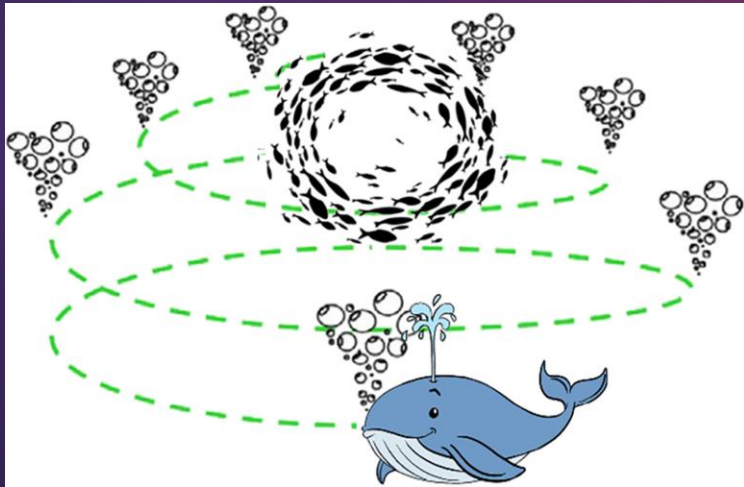


talk 1: start 8 a.m., end 10 a.m.  
talk 2: start 9 a.m., end 11 a.m.  
talk 3: start 10:30 a.m., end 12 noon  
talk 4: start 9:30 a.m., end 1 p.m.  
talk 5: start 8:30 a.m., end 2 p.m.  
talk 6: start 11 a.m., end 2 p.m.  
talk 7: start 1 p.m., end 2 p.m.



## Hunting strategy of Humpback Whale

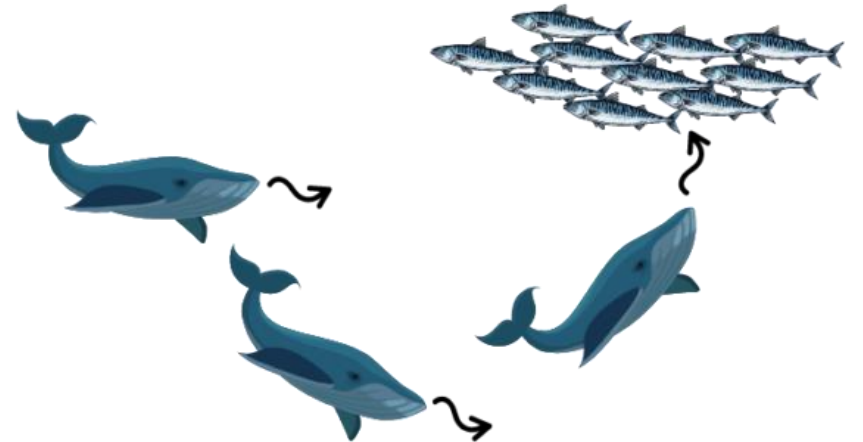
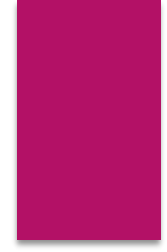
To catch their prey, whales use  
Bubble Net Feeding method



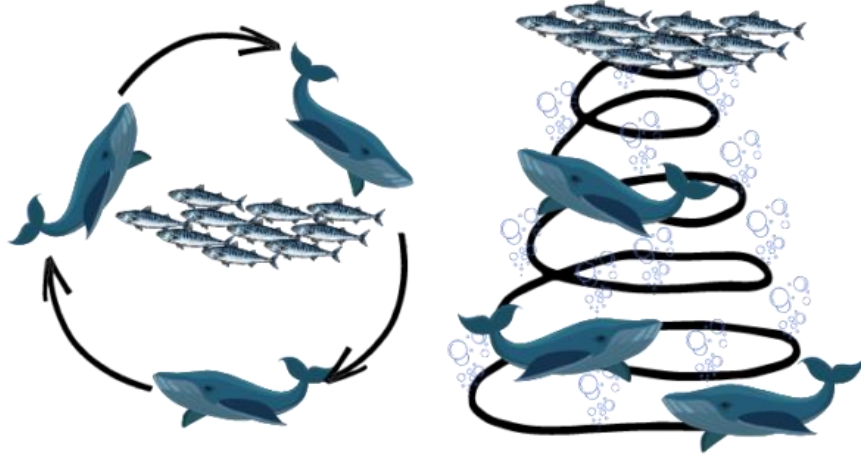
## Whale Optimization Algorithm (WOA)

- ▶ Humpback Whale dive 12 meters deep.
- ▶ They create bubble around the target.
- ▶ Target will be captured by bubbles.
- ▶ Ready to eat.

# Humpback Whale: Hunting Technique



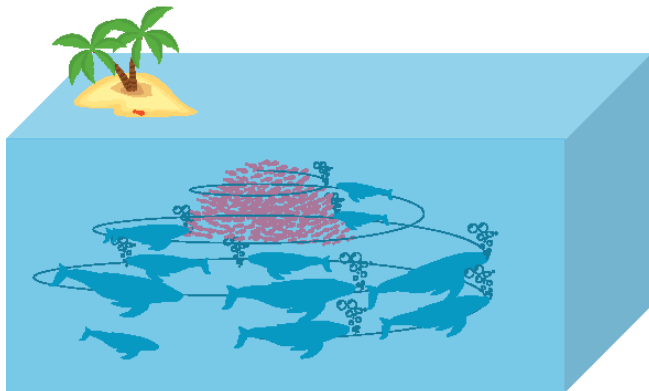
Whales search randomly for a prey



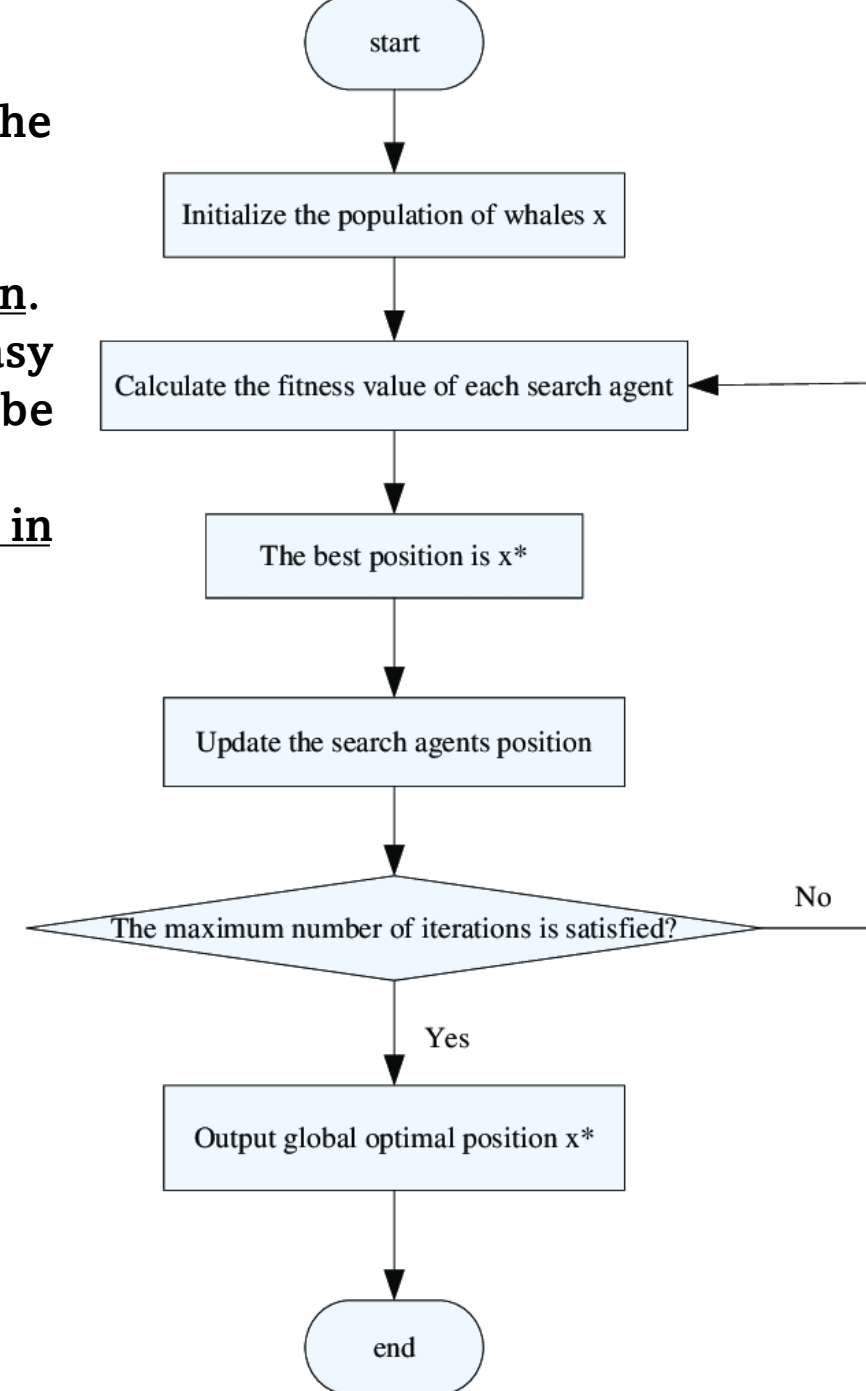
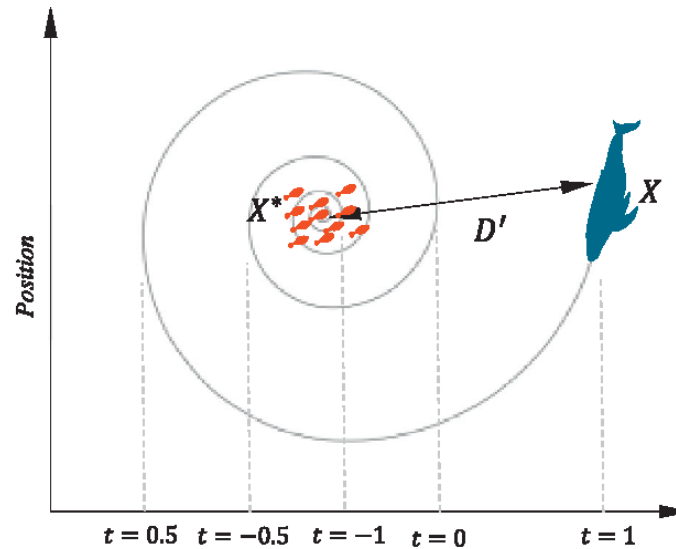
Whales create spiral-shaped bubbles with shrinking circles



- WOA was proposed based on the bubble-net hunting behaviour of the humpback whales.
- WOA is one of the nature- inspired heuristic optimization algorithms.
- WOA is widely used in engineering applications and medical application.
- WOA is efficient algorithm the key characters are, it is simple and easy to implement, gradient information is not required, local optima can be bypassed and used in a wide range of disciplinary challenges.
- WOA is used to find optimal solutions in highly complex constraints in reasonable time period.



$$\vec{X}(t + 1) = D' \cdot e^{bt} \cdot \cos(2\pi t) + \vec{X}^*(t)$$



# Applications of WOA

The WOA is a new swarm intelligence optimization algorithm, which was proposed by Australian scholars Mirjalili and Lewis in 2016. Inspired by the hunting behavior of humpback whales in nature, the algorithm simulates the shrinking encircling, spiral updating position, and random hunting mechanisms of the humpback whale population. The algorithm includes three stages: encircling prey, bubble net attack and search for prey.

S. No.	Application
	<i>Problem</i>
1.	Economic and Emission Dispatch using WOA
2.	Multi-Objective Optimal Vehicle Fuel Consumption based on WOA
3.	Multi-objective optimal mobile robot path planning base on WOA
4.	An Ameliorative WOA for Multi-Objective Optimal Allocation of Water Resources
5.	A MOWOA for Solving Engineering Design Problems
6.	WOA for combined heat and power economic dispatch

Using the WOA, brain tumour can be classified.

Opinion leader detection Opinion in online social network using WOA.

For the community detection algorithm, the whale optimization algorithm is used. WOCDA: A whale optimization-based community detection algorithm.

Optimum position and size of the battery storage unit to minimize losses using WOA.

01

02

03

04

05

06

07

08

Classification of sentiments within online social media based on the theory of social impact using whale optimization algorithm.

A guided population archive whale optimization algorithm for solving multiobjective optimization problems.

A whale optimization algorithm is used to solve global optimization problems.

Positioning of charging stations for electric vehicles with service ability using WOA.



# Research on WOA

- <https://www.mdpi.com/1424-8220/21/13/4579>
- <https://www.scirp.org/journal/paperinformation.aspx?paperid=101268>







**REVISION**



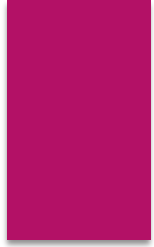
# Practice


1. List all the steps used by [Algorithm 1 \(procedure max\)](#) to find the maximum of the list: 1, 8, 12, 9, 11, 2, 14, 5, 10, 4.

2. Devise an algorithm that finds the sum of all the integers in a list.



3. Describe an algorithm that takes as input a list of  $n$  integers and finds the location of the last even integer in the list or returns 0 if there are no even integers in the list.





► 4. Describe an algorithm that locates the first occurrence of the largest element in a finite list of integers, where the integers in the list are not necessarily distinct.

5) Sort these lists using the insertion sort.

a) 3, 5, 4, 1, 2

b) 5, 4, 3, 2, 1

c) 1, 2, 3, 4, 5

6) Use the bubble sort to sort 3, 1, 5, 7, 4, showing the lists obtained at each step.





7) Use the greedy algorithm to make change using quarters, dimes, nickels, and pennies for:

- a) 51 cents
- b) 69 cents
- c) 76 cents
- d) 60 cents

8) Use the greedy algorithm to make change using quarters, dimes, and pennies (but no nickels) for each of the amounts given in 7) *question above*. For which of these amounts does the greedy algorithm use the fewest coins of these denominations possible?

9) Use Algorithm 7 to schedule the largest number of talks in a lecture hall from a proposed set of talks, if the starting and ending times of the talks are 9:00 A.M. and 9:45 A.M.; 9:30 A.M. and 10:00 A.M.; 9:50 A.M. and 10:15 A.M.; 10:00 A.M. and 10:30 A.M.; 10:10 A.M. and 10:25 A.M.; 10:30 A.M. and 10:55 A.M.; 10:15 A.M. and 10:45 A.M.; 10:30 A.M. and 11:00 A.M.; 10:45 A.M. and 11:30 A.M.; 10:55 A.M. and 11:25 A.M.; 11:00 A.M. and 11:15 A.M.





CHALLENGE YOURSELF

Sort these lists using the selection sort.

a) 3, 5, 4, 1, 2

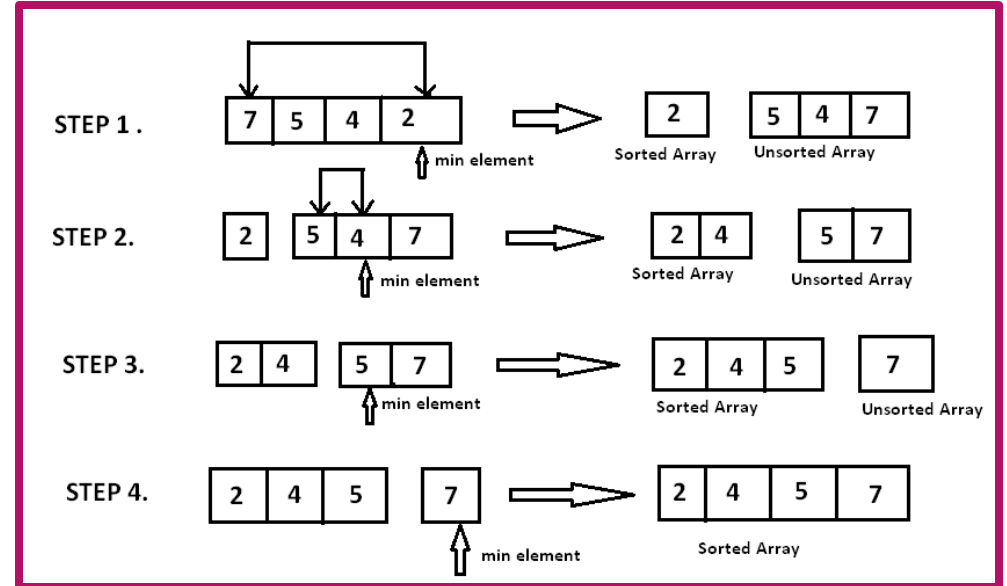
b) 5, 4, 3, 2, 1

c) 1, 2, 3, 4, 5

**Selection sort** is an effective and efficient sort algorithm based on comparison operations. It adds one element in each iteration. You need to select the smallest element in the array and move it to the beginning of the array by swapping with the front element.

**Algorithm 4** Selection Sort

```
1: for  $i = 1$  to  $n - 1$  do
2:    $min = i$ 
3:   for  $j = i + 1$  to  $n$  do
4:     // Find the index of the  $i^{th}$  smallest element
5:     if  $A[j] < A[min]$  then
6:        $min = j$ 
7:     end if
8:   end for
9:   Swap  $A[min]$  and  $A[i]$ 
10: end for
```



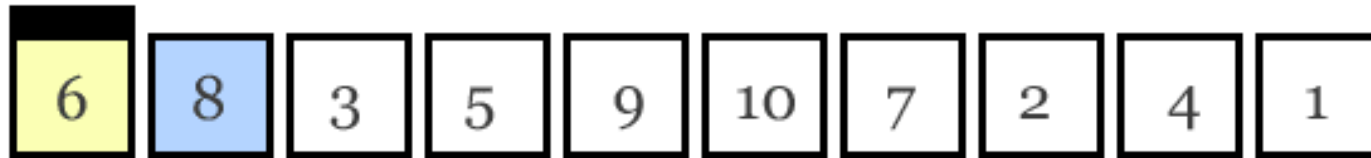
a) 1, 5, 4, 3, 2;  
1, 2, 4, 3, 5;  
1, 2, 3, 4, 5;  
1, 2, 3, 4, 5

b) 1, 4, 3, 2, 5;  
1, 2, 3, 4, 5;  
1, 2, 3, 4, 5;  
1, 2, 3, 4, 5

c) 1, 2, 3, 4, 5;  
1, 2, 3, 4, 5;  
1, 2, 3, 4, 5;  
1, 2, 3, 4, 5



# Selection Sort



Yellow is smallest number found

Blue is current item

Green is sorted list



## Links for additional and detailed information about Optimization



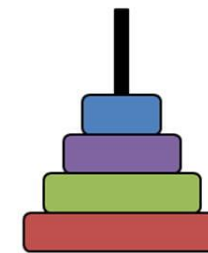
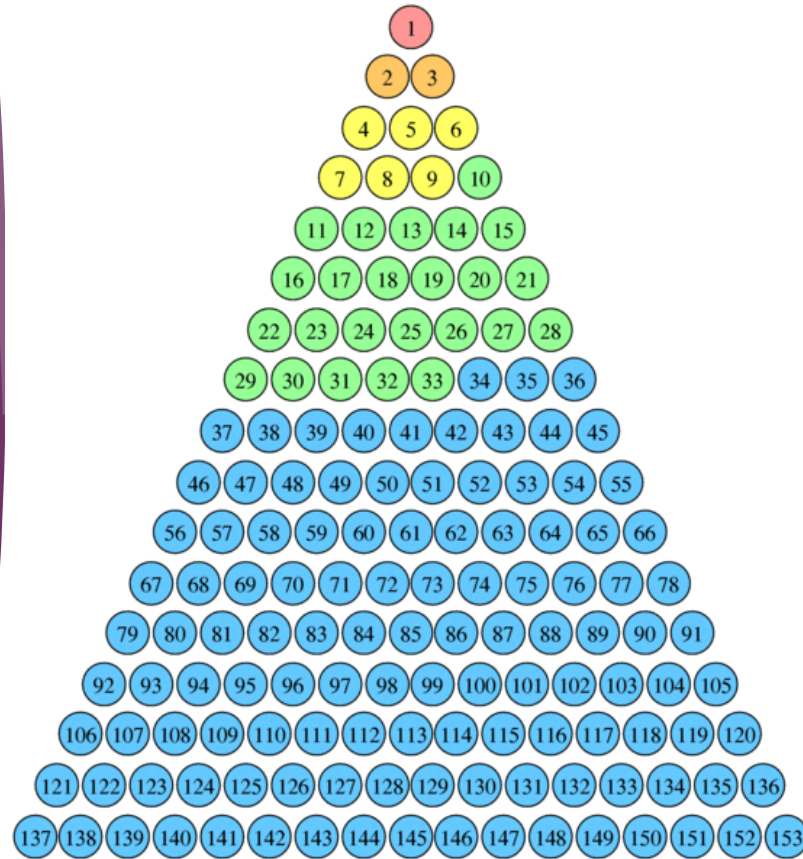
<https://machinelearningmastery.com/tour-of-optimization-algorithms/>



<https://algorithmsbook.com/optimization/files/optimization.pdf>

# Recursive

- ▶ Towers of Hanoi
- ▶ Triangular Numbers
- ▶ Fibonacci
- ▶ Sequences
- ▶ Factorial



(A) Start



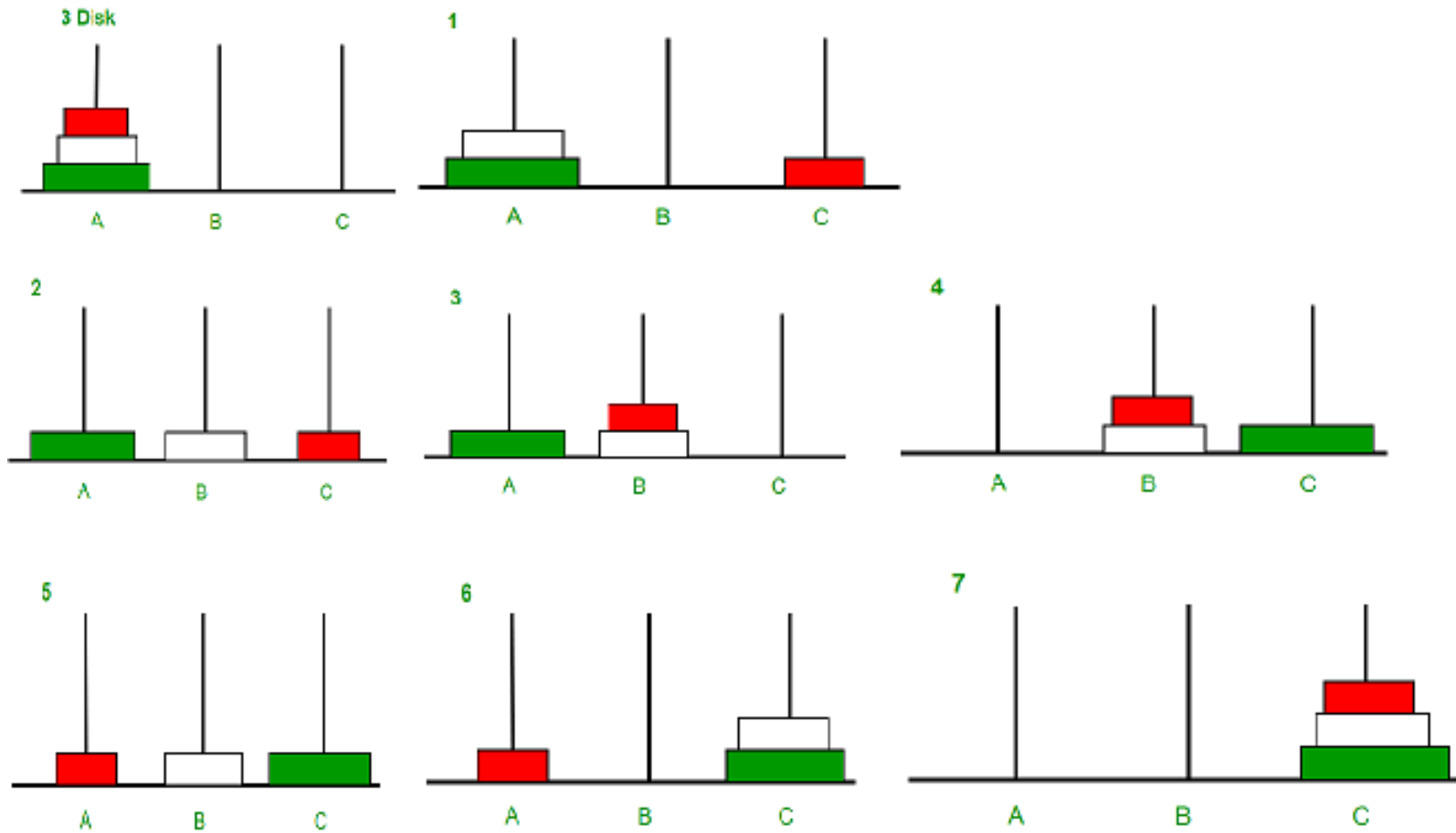
(B) Middle



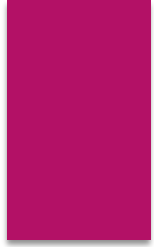
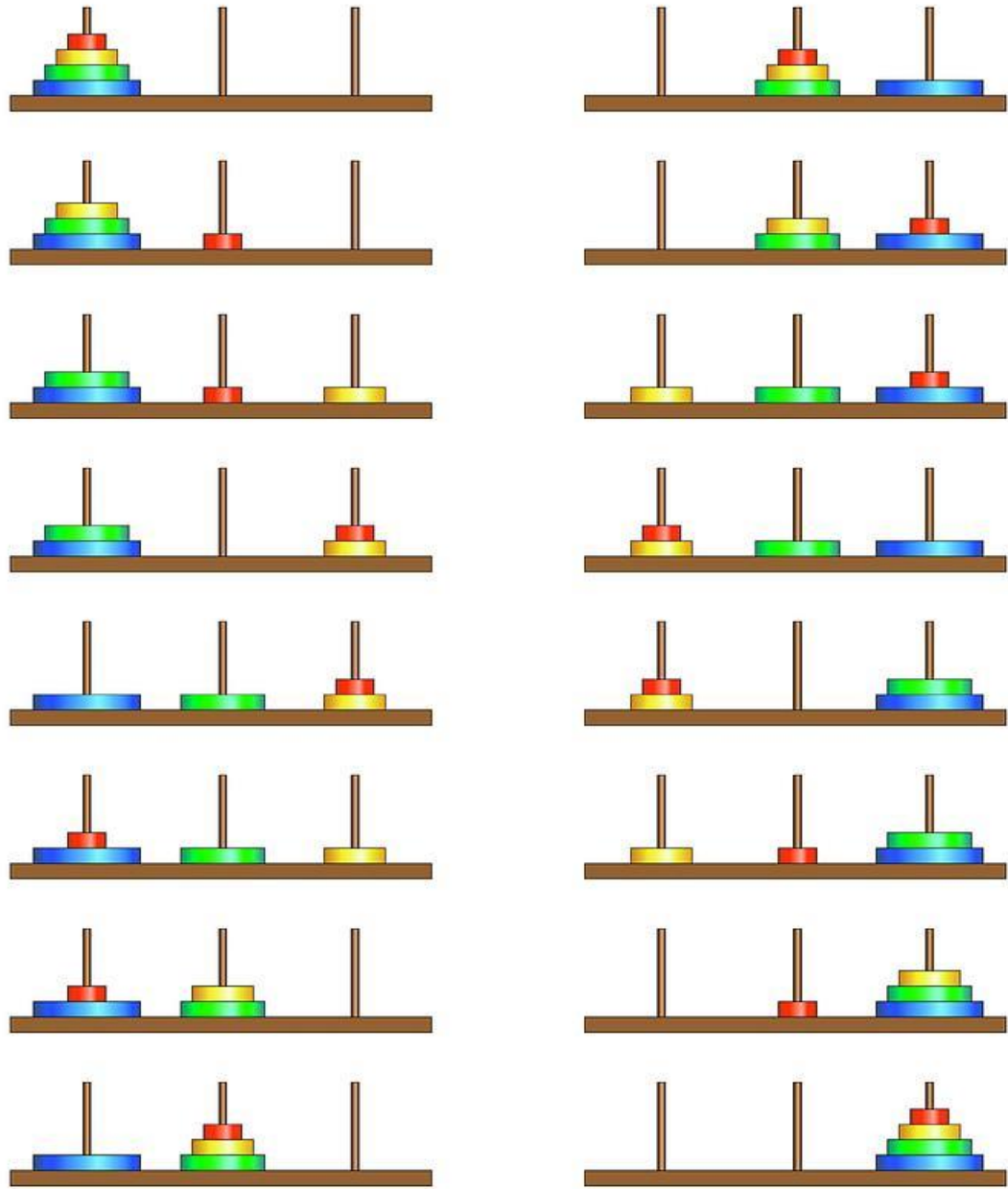
(C) Goal

# Towers of Hanoi

$n = 3$ :



$n = 4$ :



# Towers of Hanoi

1,3,7,15,31,63,127,255 ...

$$2(a_{n-1}) + 1$$

What if n is large?

$$2^n - 1$$

If you had 64 golden disks you would have to use a minimum of  $2^{64} - 1$  moves. If each move took one second, it would take around 585 *billion* years to complete the puzzle!

N \ k	1	2	3	4	5	6	7	8	SUM
1	1								1
2	1	2							3
3	1	2	4						7
4	1	2	4	8					15
5	1	2	4	8	16				31
6	1	2	4	8	16	32			63
7	1	2	4	8	16	32	64		127
8	1	2	4	8	16	32	64	128	255

# Towers of Hanoi animations

Play and make sure you get minimum moves for certain number of discs:

<https://www.mathsisfun.com/games/towerofhanoi.html>

You can check your solutions comparing with this one:

<http://towersofhanoi.info/Animate.aspx>





1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

$$x_n = \frac{n(n+1)}{2}$$

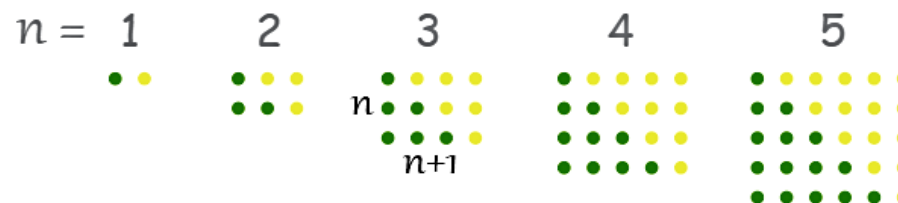
## A Rule

We can make a "Rule" so we can calculate any triangular number.

First, rearrange the dots like this:



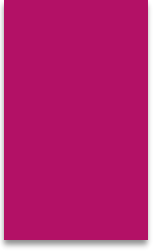
Then double the number of dots, and form them into a rectangle:



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, ...

$$F_n = F_{n-1} + F_{n-2}$$





➤ {5, 11, 17, 23, ... }

➤ {3, 6, 12, 24, ... }

➤ {2, 6, 14, 30, 62, ... }

**Find the Factorial of a Number and Fibonacci series  
using Recursion (PseudoCode or Algorithm)**

➤ {1, 2, 6, 24, 120, 720, ... }

➤ {2, 4, 16, 256, 65536, ... }

➤ {2, 3, 6, 18, 108, 1944, 209952, ... }

# Find the Factorial of a Number using Recursion

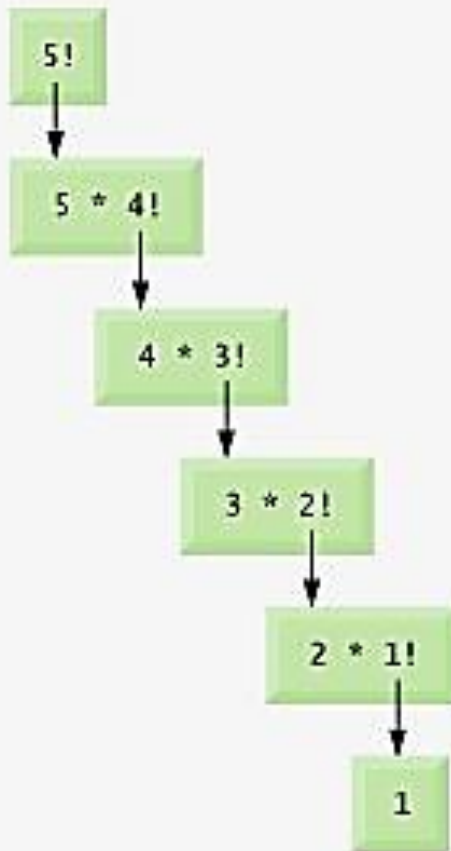
```
#include <iostream>
using namespace std;

unsigned int factorial(unsigned int n)
{
    if (n == 0 || n == 1)
        return 1;
    return n * factorial(n - 1);
}

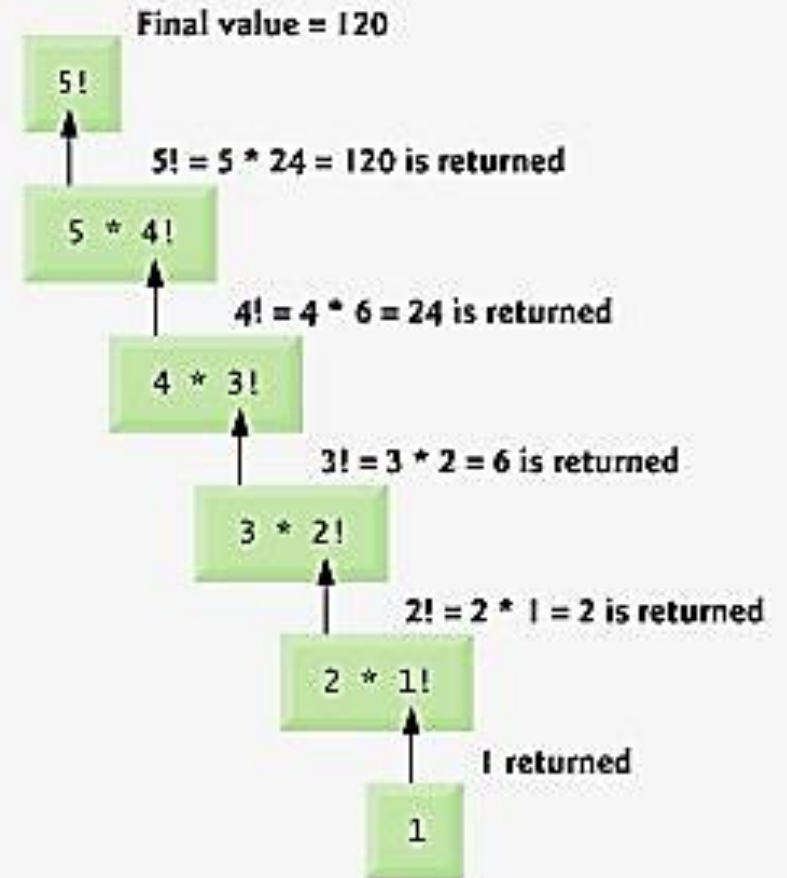
int main()
{
    int num = 5;
    cout << "Factorial of "
         << num << " is " << factorial(num) << endl;
    return 0;
}
```

n	n!		
1	1	1	1
2	2 × 1	= 2 × 1!	= 2
3	3 × 2 × 1	= 3 × 2!	= 6
4	4 × 3 × 2 × 1	= 4 × 3!	= 24
5	5 × 4 × 3 × 2 × 1	= 5 × 4!	= 120
6	etc	etc	

# Recursive Demonstration of Factorial



(a) Sequence of recursive calls.



(b) Values returned from each recursive call.

## // Fibonacci Series using Recursion

```
#include <iostream>
using namespace std;

int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}

int main()
{
    int n = 9;
    cout << fib(n);
    return 0;
}
```

## // Fibonacci Series using Space Optimized Method

```
#include <bits/stdc++.h>
using namespace std;

int fib(int n)
{
    int a = 0, b = 1, c, i;
    if (n == 0)
        return a;
    for (i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

int main()
{
    int n = 9;

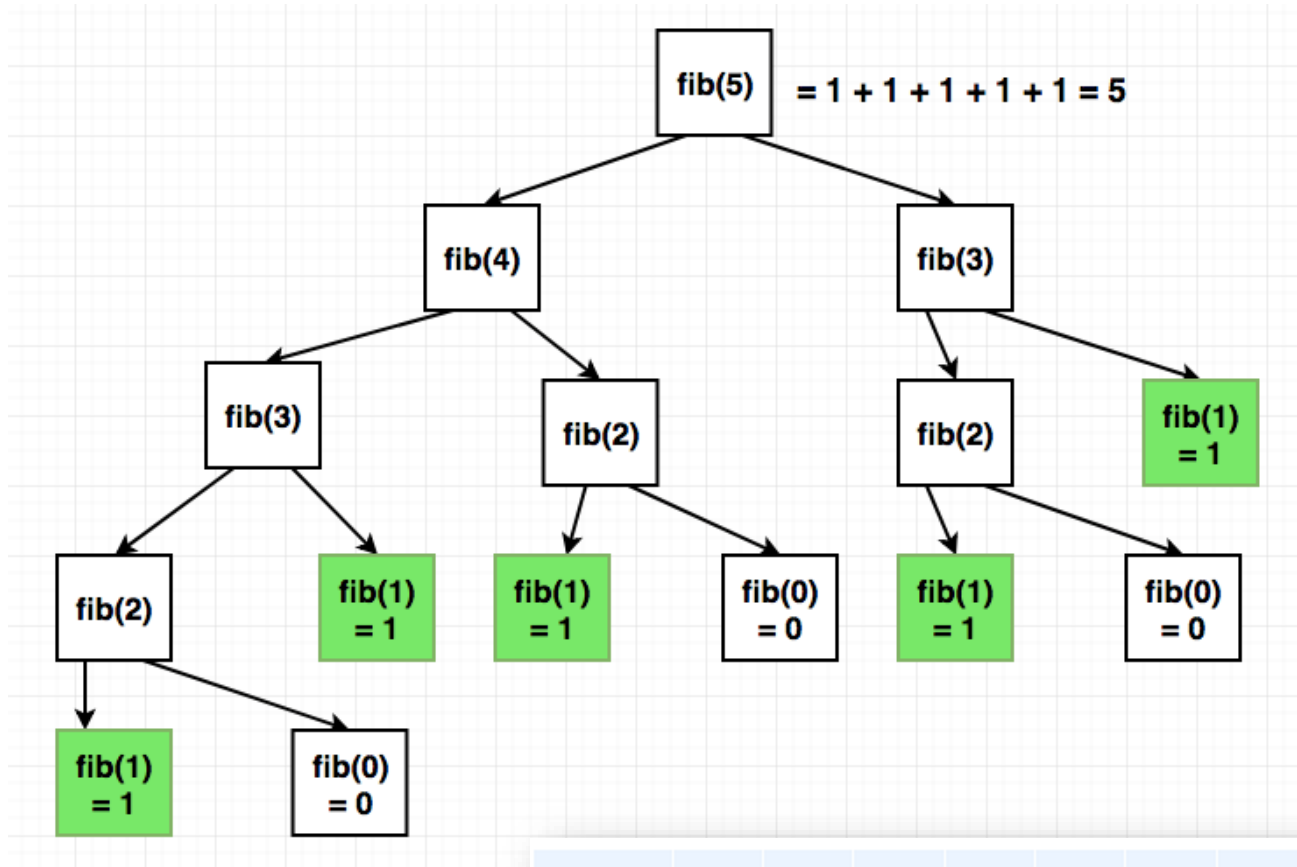
    cout << fib(n);
    return 0;
}
```

```
#include <iostream>
using namespace std;
```

```
class GFG{
public:
int fib(int n)
{
    int f[n + 2];
    int i;
    f[0] = 0;
    f[1] = 1;
    for (i = 2; i <= n; i++)
    {
        f[i] = f[i - 1] + f[i - 2];
    }
    return f[n];
}
};

int main ()
{
    GFG g;
    int n = 4;
    cout << g.fib(n);
    return 0;
}
```

# Recursive Demonstration of the Fibonacci Sequence

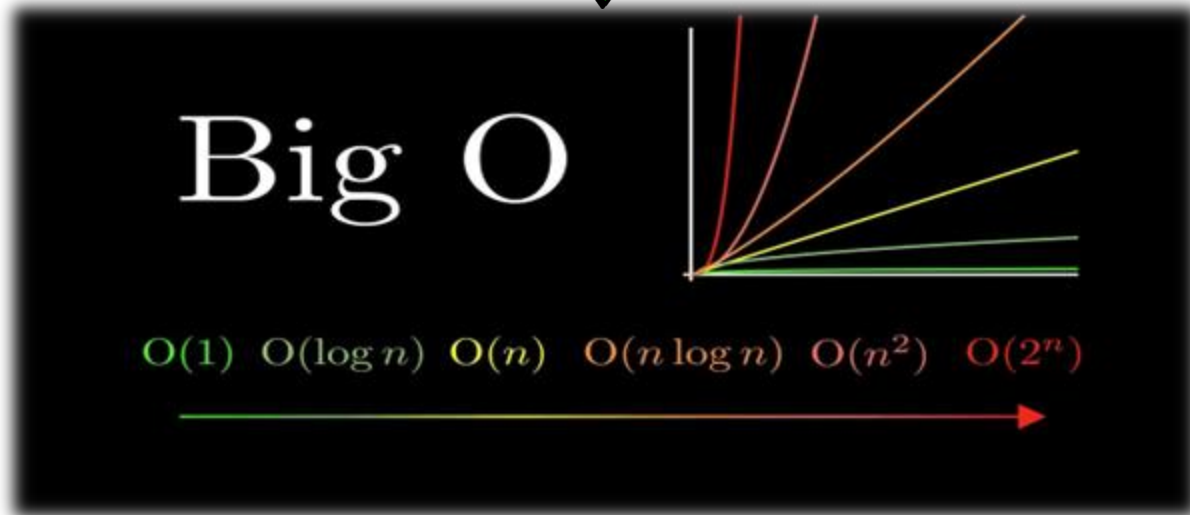


$n =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
$x_n =$	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	...

# Big-O Notation



```
32 self.file = open('log.txt', 'a')
33 self.fingerprints = set()
34 self.logdups = True
35 self.debug = debug
36 self.logger = logging.getLogger(__name__)
37 if path:
38     self.file = open(os.path.join(path, 'requests.log'), 'a')
39     self.file.seek(0)
40     self.fingerprints.update(request)
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('DEBUG', False)
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```



# Big-O Notation

## Time Complexity

In the time complexity analysis, we define the time as a function of the problem size and try to estimate the growth of the execution time with the growth in problem size.



# Big-O Notation

## Memory Space

The space require by the program to save input data and results in memory (RAM).

# Big-O Notation

## History

Big-O notation (with a capital letter O, not a zero), also called Landau's symbol, is a symbolism used in complexity theory, computer science, and mathematics to describe the basic behavior of functions. Basically, it tells you how fast a function grows or declines.

Landau's symbol comes from the name of the German mathematician Edmund Landau who invented the notation.

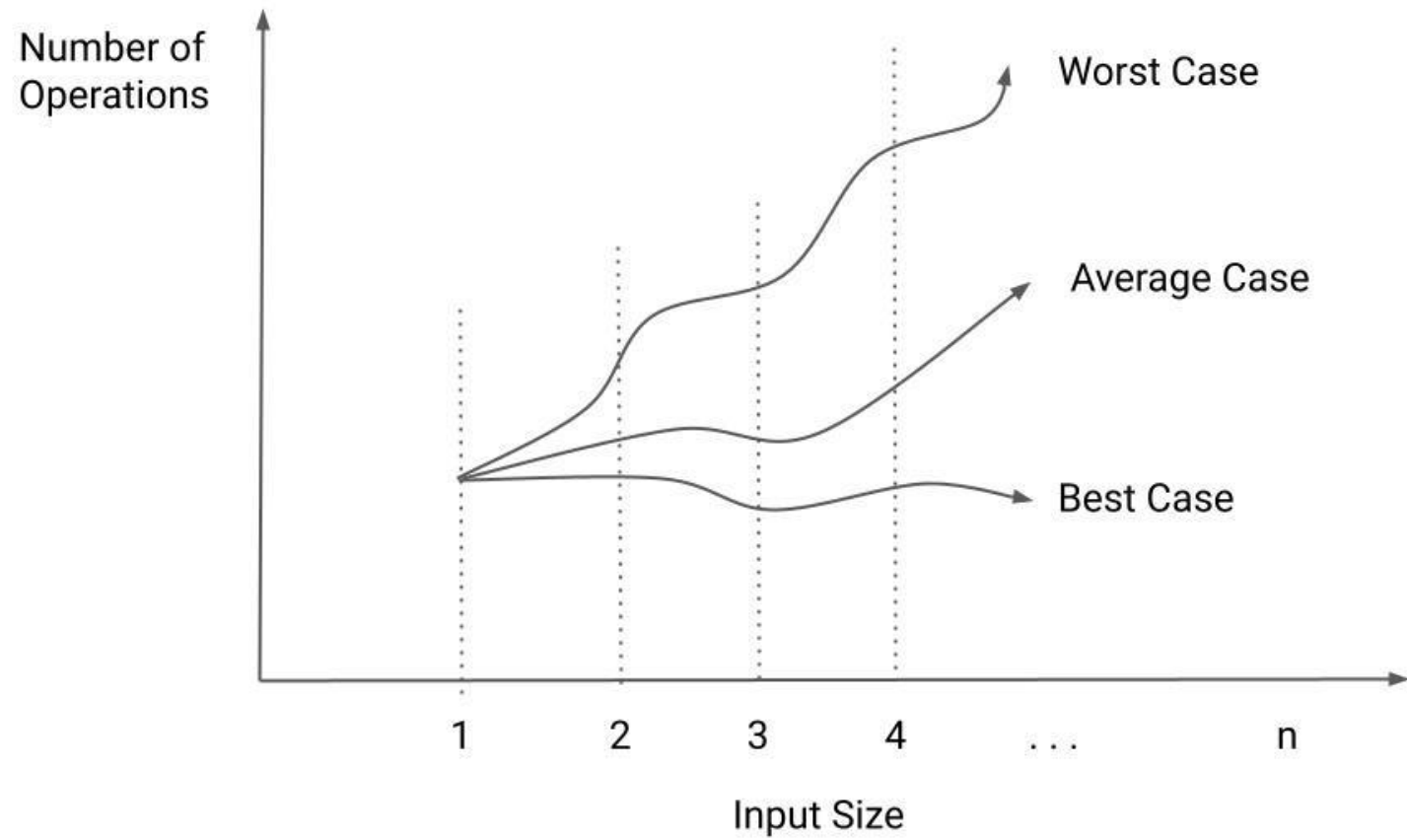
The letter O is used because the rate of growth of a function is also called its order.

# Big-O Notation

## It Describes:

- ✓ Efficiency of an Algorithm.
- ✓ Time Factor of an Algorithm.
- ✓ Space Complexity of an Algorithm.

It is represented by  $O(n)$ , where  $n$  is the number of operations.



# Execution Time

Let us consider that every operation can be executed in 1 ns ( $10^{-9}$ s).

Function	Time		
	$(n = 10^3)$	$(n = 10^4)$	$(n = 10^5)$
$\log_2 n$	10 ns	13.3 ns	16.6 ns
$\sqrt{n}$	31.6 ns	100 ns	316 ns
$n$	1 $\mu$ s	10 $\mu$ s	100 $\mu$ s
$n \log_2 n$	10 $\mu$ s	133 $\mu$ s	1.7 ms
$n^2$	1 ms	100 ms	10 s
$n^3$	1 s	16.7 min	11.6 days
$n^4$	16.7 min	116 days	3171 yr
$2^n$	$3.4 \cdot 10^{284}$ yr	$6.3 \cdot 10^{2993}$ yr	$3.2 \cdot 10^{30086}$ yr

# Big-O Notation

- ▶ We use Big-O notation to classify algorithms based on the number operations or comparisons they use.
- ▶ For large values of  $x$ :  $x^2$ ,  $3x^2 + 25$ ,  $4x^2 + 7x + 10$  are all very similar, so we will consider them of the same **order:  $O(x^2)$**
- ▶  $f(x)$  is  $O(g(x))$  if there exist constants (witnesses)  $C$  and  $k$  such that  $|f(x)| \leq C|g(x)|$  whenever  $x > k$ .

# Example 1 (Version 1)

Show that  $3x^2 + 25$  is  $O(x^2)$

$$\text{Let: } x = 5: \quad 3x^2 + 25 = 3(5)^2 + 25 = 75 + 25 = 100$$

$k$

$$C(5)^2 \geq 100$$

$$25C \geq 100$$

$$C \geq 4$$

$$\text{Let } C = 4, k = 5$$

$$|3x^2 + 25| \leq 4|x^2| \quad \text{when } x > 5.$$

$$x = 6:$$

$$3x^2 + 25 = 133$$

$$4x^2 = 144$$

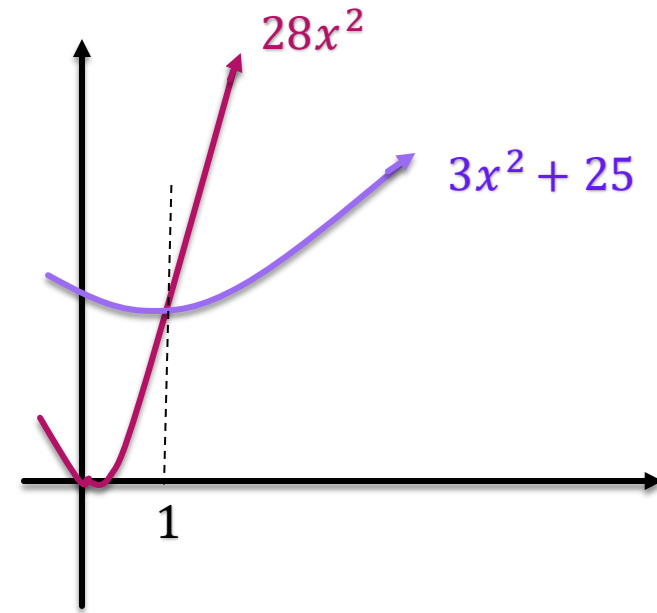
# Example 1 (Version 2)

Show that  $3x^2 + 25$  is  $O(x^2)$

$$3x^2 + 25 \leq 3x^2 + 25x^2, \quad x > 1$$

$$3x^2 + 25 \leq 28x^2, \quad x > 1$$

$$C = 28, \quad k = 1$$





## Example 2

Show that  $4x^3 + 7x^2 + 12$  is  $O(x^3)$  by finding the witnesses,  $C$  and  $k$ .

$$4x^3 + 7x^2 + 12 \leq 4x^3 + 7x^3 + 12x^3, \quad x > 1$$

$$|4x^3 + 7x^2 + 12| \leq |23x^3|, \quad x > 1$$

$$C = 23, \quad k = 1$$

# Example 3

Show that  $x^3 + 5x$  is not  $O(x^2)$

$$|x^3 + 5x| \leq C|x^2|, \quad x > k$$

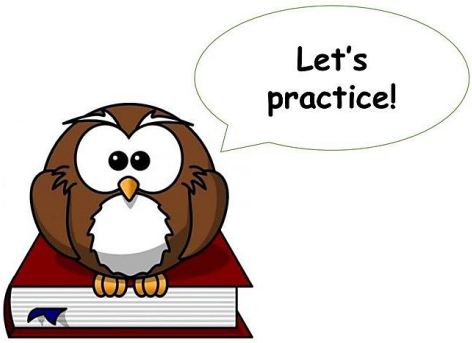
$$\text{if } C = x: \quad x^3 + 5x > x \cdot x^2, \quad x > 0$$

No fixed  $C$  that will keep

$$x^3 + 5x \leq Cx^2$$

Once  $x > C$ , calculation will be wrong

$C$  has to be a constant and this inequality has to hold for all  $x > k$ . So, no matter what you pick for  $C$ ,  $x$  can always get large enough to overcome that.



Prove that:

➔  $2n + 10 = O(n)$

➔  $1000n^2 + 1000n = O(n^2)$

# Input Size calculations

$$\log 10 = 1$$

Suppose a computer can perform  $10^{12}$  bit operations per second. Find the largest problem size that could be solved in 1 second if an algorithm requires:

➔  $n^4$  bit operations

$$n^4 = 10^{12}$$

$$(n^4)^{\frac{1}{4}} = (10^{12})^{\frac{1}{4}}$$

$$n = 10^3 = 1000$$

➔  $2^n$  bit operations

$$2^n = 10^{12}$$

$$2^{39} \approx 5.5 \cdot 10^{11}$$

$$2^{40} \approx 1.1 \cdot 10^{12}$$

$$n = 39$$

$$2^n = 10^{12}$$

$$\log 2^n = \log 10^{12}$$

$$n \cdot \log 2 = 12$$

$$n = \frac{12}{\log 2} \approx 39.86$$

# Time calculations

$$\log n = \log_2 n$$

Suppose a computer can perform  $10^{12}$  bit operations per second. Find the time it would take an algorithm that requires  $n^3 + \log n$  operations with a problem size of:



$$n = 1000$$

$$f(n) = n^3 + \log n$$

$$1000^3 + \log_2 1000 \approx 10^9 + 10 \text{ bit operations}$$

$$10^9 \text{ b.o.} \cdot \frac{1 \text{ sec}}{10^{12} \text{ b.o.}} = \frac{1}{1000} \text{ sec.}$$



$$n = 10^8$$

$$f(n) = n^3 + \log n$$

$$(10^8)^3 + \log_2 10^8 \approx 10^{24} + 26.6 \text{ bit operations}$$

$$10^{24} \text{ b.o.} \cdot \frac{1 \text{ sec}}{10^{12} \text{ b.o.}} = 10^{12} \text{ sec.}$$

$$10^{12} \text{ sec.} \cdot \frac{1 \text{ min}}{60 \text{ sec}} \cdot \frac{1 \text{ hr}}{60 \text{ min}} \cdot \frac{1 \text{ day}}{24 \text{ hour}} \cdot \frac{1 \text{ yr}}{365 \text{ days}} \approx 31710 \text{ years}$$



Given that supercomputer takes  $10^{-15}$  seconds per bit operation, how long will it take that supercomputer to solve a problem of size  $n = 100$  if the algorithm requires:

- $n^3$  bit operations?
- $n \log n$  operations?

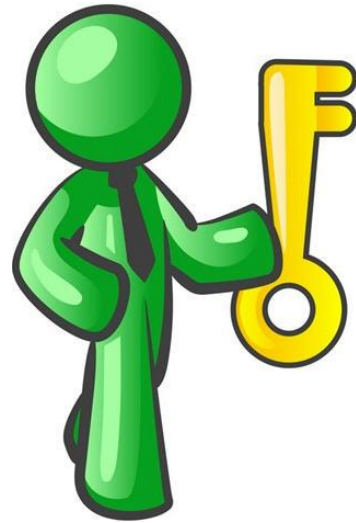


What is the largest problem size  $n$  that we can solve in no more than one hour using an algorithm that requires  $f(n)$  operations, where each operation takes  $10^{-9}$  seconds (this is close to a today's computer), with the following  $f(n)$  ? Below are examples of the given algorithms.

01  $\gg$   $\log_2(n)$

02  $\gg$   $\log_2^4(n)$

03  $\gg$   $3n$



$$f(n) \times 10^{-9} s = 1 \text{ hour} = 3600s$$
$$f(n) = 3.6 \times 10^{12}$$

01  $\gg$   $\log_2 n = 3.6 \times 10^{12}$   
 $\implies n = 2^{3.6 \times 10^{12}}$

02  $\gg$   $(\log_2 n)^4 = 3.6 \times 10^{12}$   
 $n = 2^{1.37 \times 10^3}$

03  $\gg$   $n = 1.2 \times 10^{12}$



Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size  $n$ .) Suppose you have a computer that can perform  $10^{10}$  operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size  $n$  for which you would be able to get the result within an hour?

- a)  $n^2$
- b)  $n^3$
- c)  $100n^2$
- d)  $2^n$



a) For  $n^2$ , the running is  $n^2$  operations. Since the computer can perform  $10^{10}$  operations per second, the largest input size  $n$  for which the result can be computed within an hour is:

$n^2 = 10^{10} \times 3600$ . Simplifying, we get:

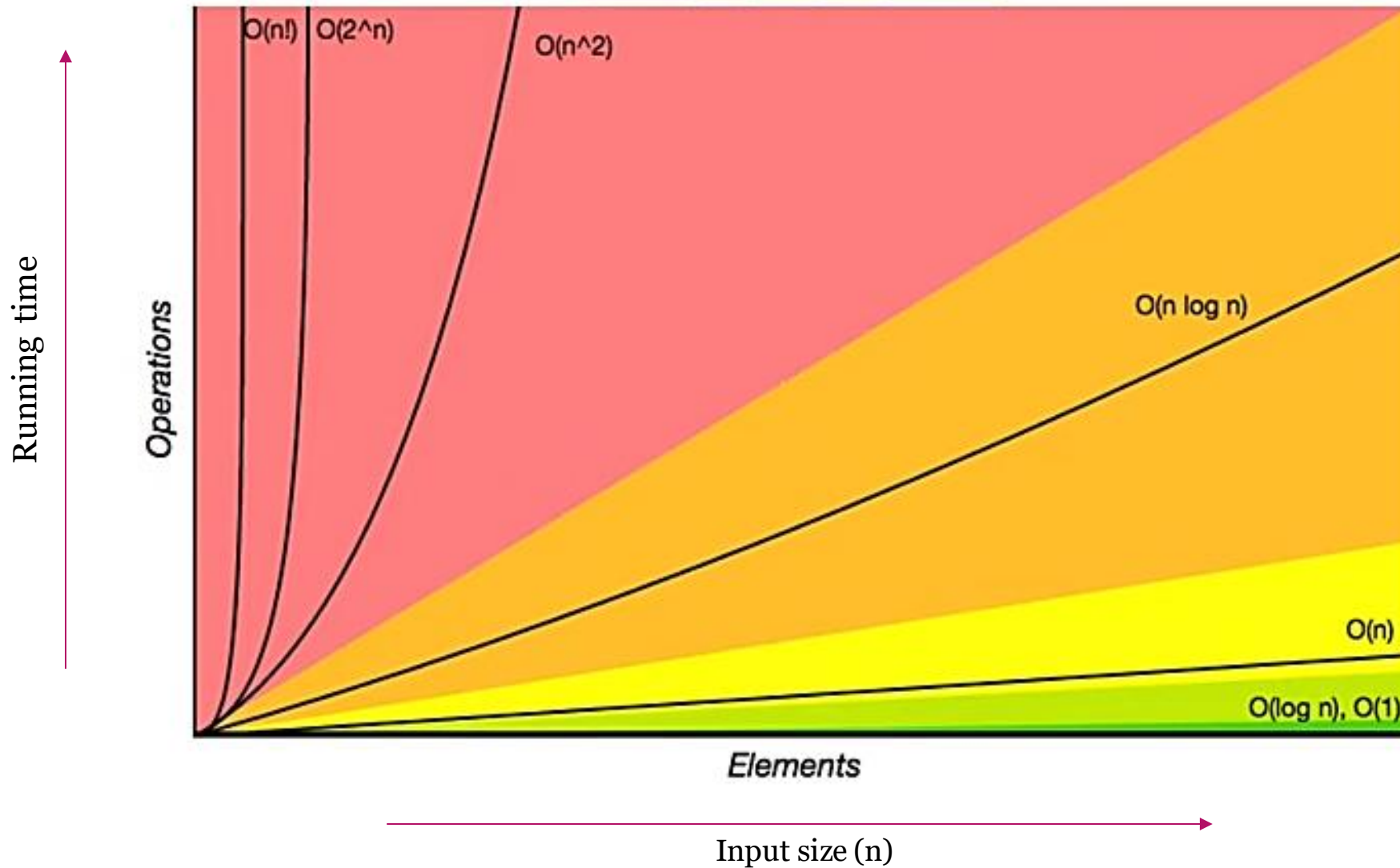
$$n = \sqrt{10^{10} \times 3600}, n = 6000000.$$

So, the largest input size  $n$  is 6000000.



# Big-O Complexity Chart

Horrible Bad Fair Good Excellent



## ***Examples of Algorithms and their big-O complexity***

Big-O Notation	Examples of Algorithms
$O(1)$	Push, Pop, Enqueue (if there is a tail reference), Dequeue, Accessing an array element
$O(\log(n))$	Binary search
$O(n)$	Linear search
$O(n \log(n))$	Heap sort, Quick sort (average), Merge sort
$O(n^2)$	Selection sort, Insertion sort, Bubble sort
$O(n^3)$	Matrix multiplication
$O(2^n)$	Towers of Hanoi

# References

<https://www.enjoyalgorithms.com/blog/time-complexity-analysis-in-data-structure-and-algorithms>

Discrete Mathematics explained in Kurdish:

- ▶ <https://www.youtube.com/watch?v=A4dq1rVwcF4&list=PLxIvc-MGOs6gZlMVYOOEtUHJmfUquCjwz&index=17>

Discrete Mathematics and its Applications by Kenneth H. Rosen, Chapter 3, page 191:

- ▶ <https://www.houstonisd.org/cms/lib2/TX01001591/Centricity/Domain/26781/DiscreteMathematics.pdf>

