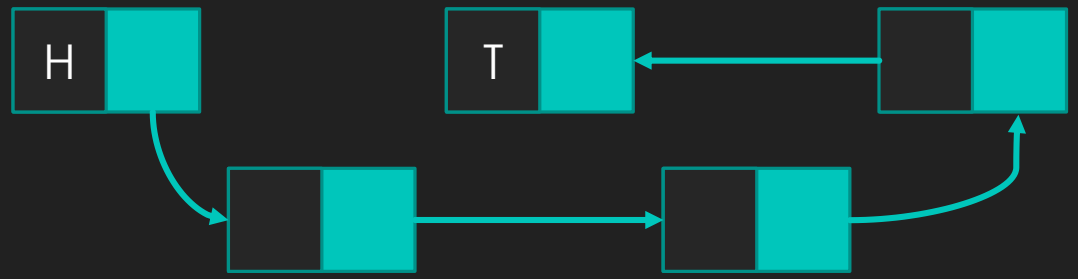
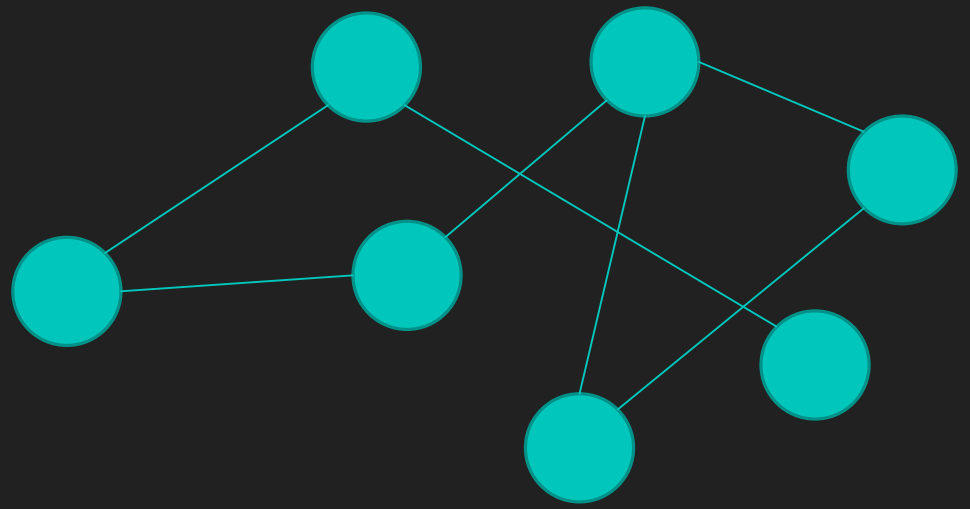
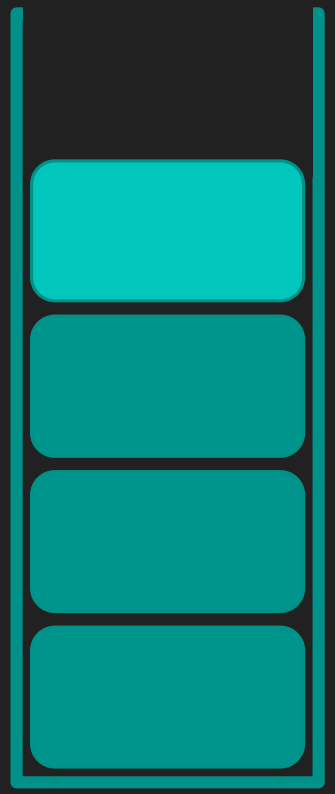
Data structure And Algorithm

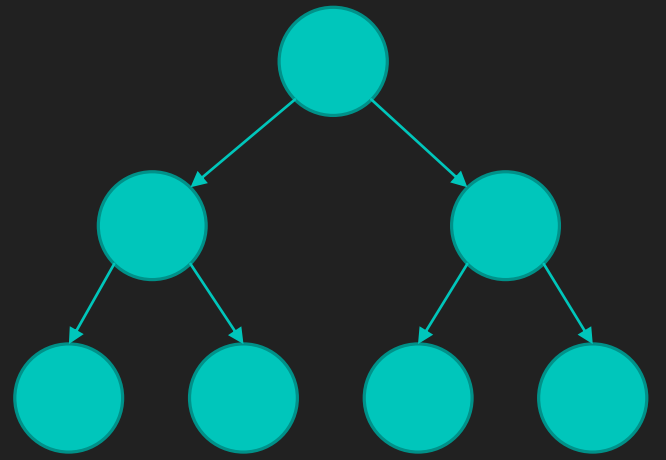
Mohammad Ghoddosi

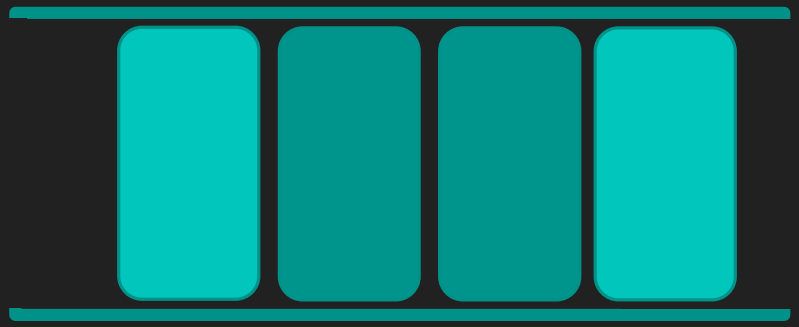






Data structure





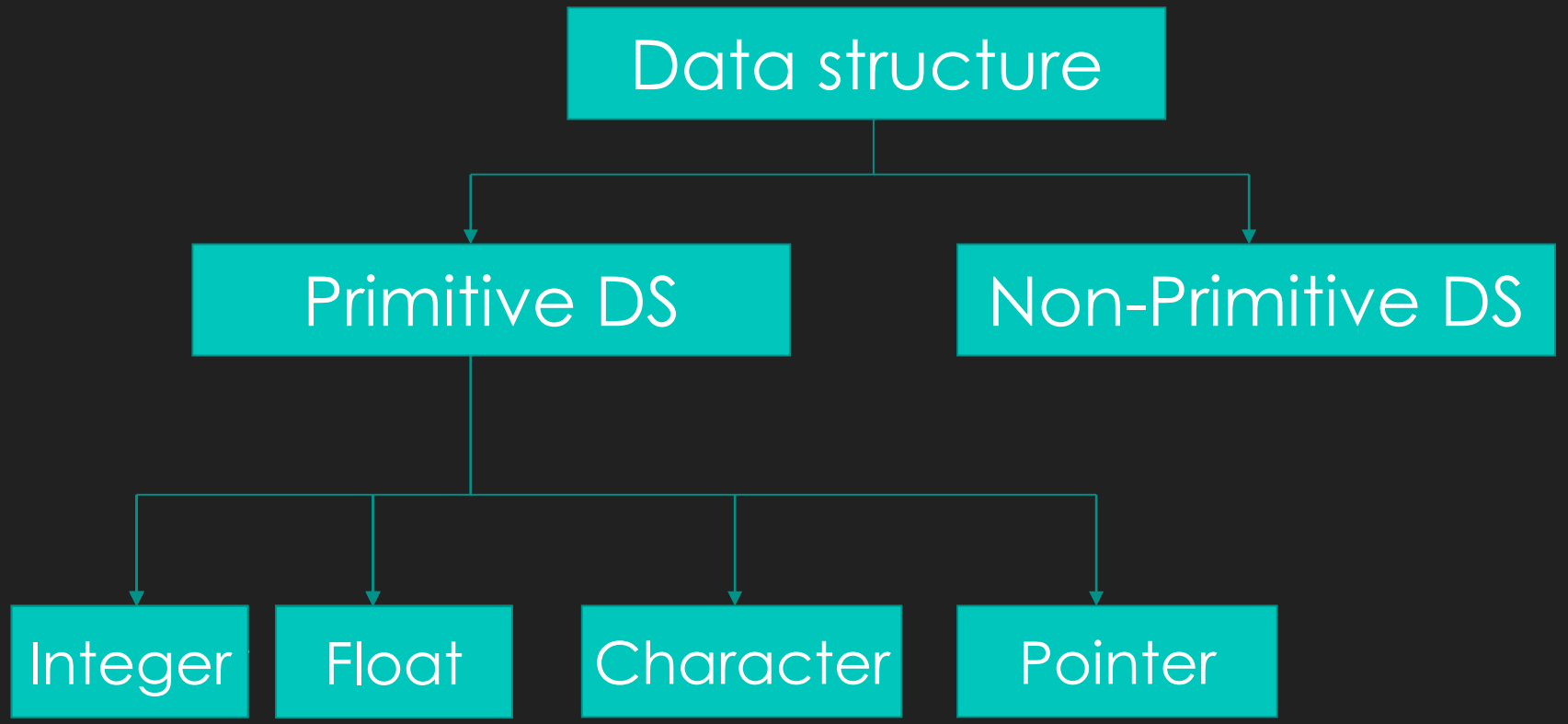
What is Data structure

- Data structure is representation of the logical relationship existing between individual elements of data.
- A way of organizing all data items
  - Store elements
  - Store relationships between data
- One of the most important parts of programming

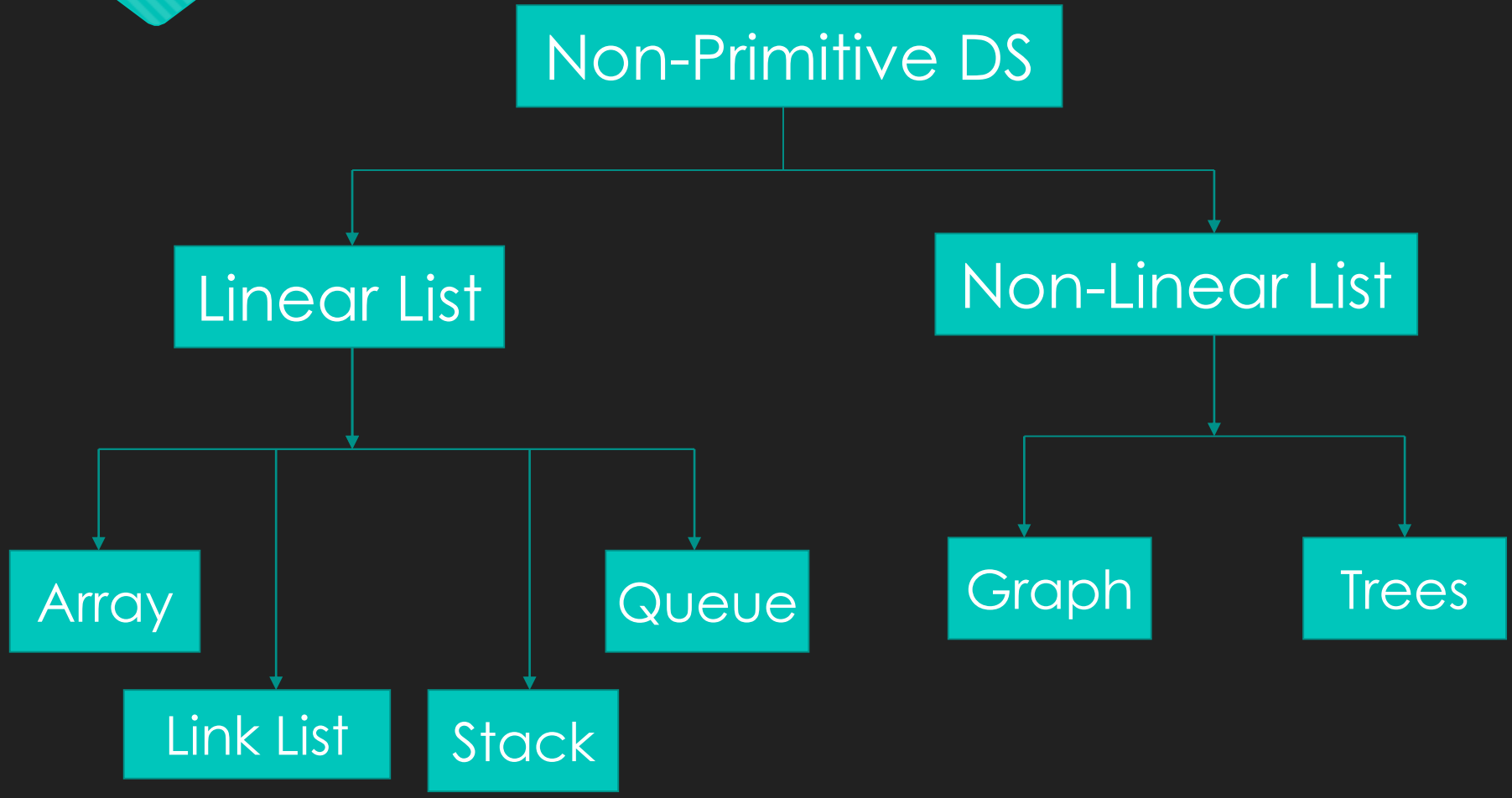
Algorithm and Data Structure

- Program = Algorithm + Data Structure (both implemented by a programming language)
- Algorithm
  - Step by step procedure to solve a particular problem
  - Set of instructions
- Data Structure
  - The way you store data
- To develop a program of an algorithm, we should select an appropriate data structure for that algorithm.
- Therefore algorithm and its associated data structures form a program.

Types of DS


Data structure
Primitive DS
Non-Primitive DS
Integer
Float
Character
Pointer

Types of DS


Non-Primitive DS
Linear List
Non-Linear List
Array
Link List
Stack
Queue
Graph
Trees

Primitive Data Structure

- There are basic structures and directly operated upon by the machine instructions.
  - Integer
  - Floating-point number
  - Character
  - String
  - Pointers
  - ….
- In general, there are different representation on different computers.

Non-Primitive Data Structure

- There are more sophisticated data structures.
- These are derived from the primitive data structures.
- The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

Non-Primitive Data Structure

- Array
- Stack
- Queue
- Linked list
- Graph
- Tree
- Heap
- …

How to compare Data Structures

- Operations
  - Create
  - Insert
  - Select
  - Update
  - Search
  - Sort
  - Merge
  - Delete
  - Destroy

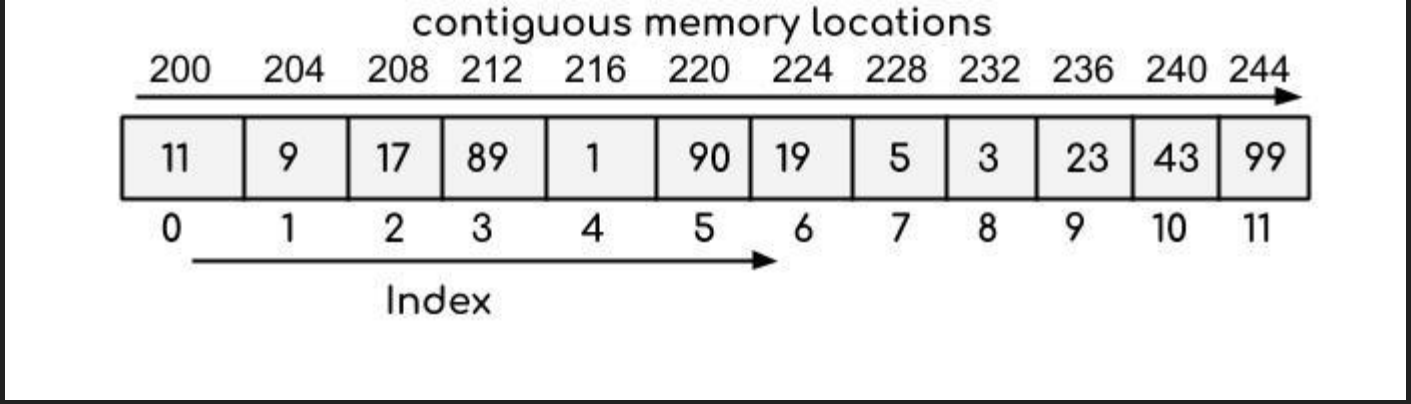
Example

- Imagine we write our data
  - in a normal Array
  - In a sorted Array

- Which one is faster?
  - Insert new element
  - Find specific element
  - Merge 2 different data sets

Arrays

- Like lists in python
- Some elements is order
- Continues in memory
- Array in DS has fixed size
- Each element can be accessed directly



Arrays

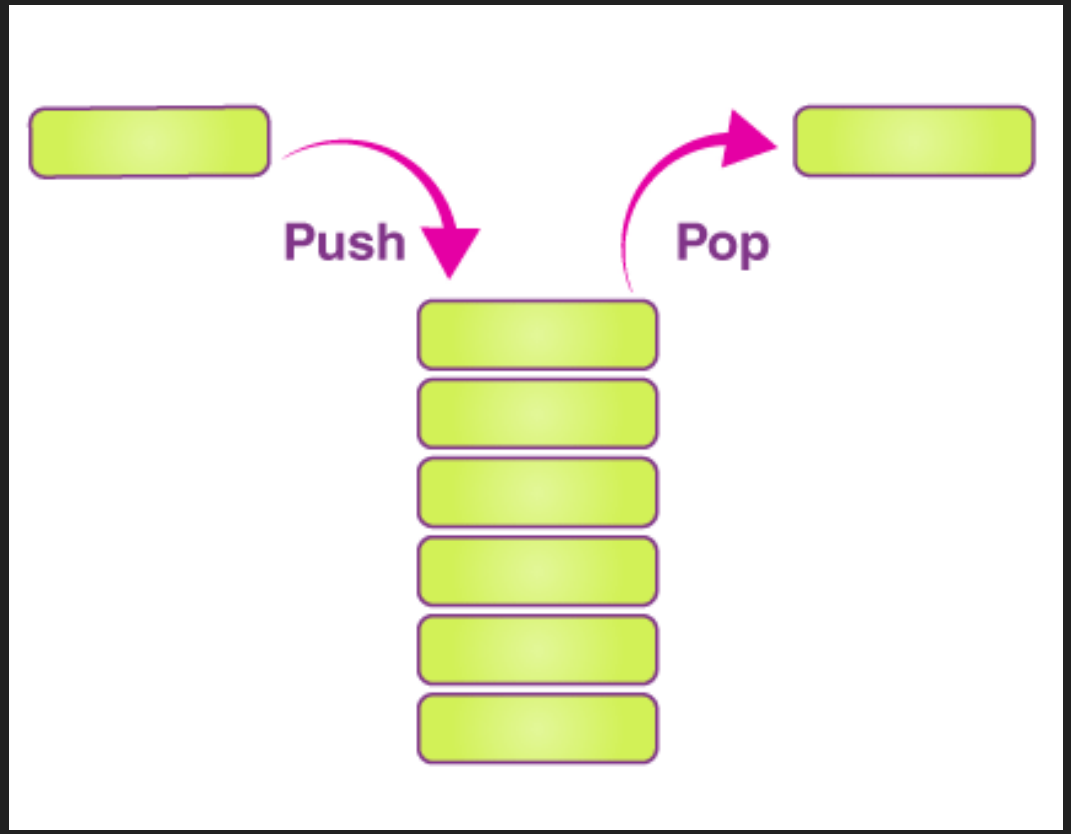
- Insertion of new elements
- Deletion of required element
- Modification of an element
- Reordering elements
- Merging two arrays

Stack

- Like arrays
- Deletion and insertion can be done only from one end
- Last In First Out (LIFO)
- Stack of dishes

Stack

- Insertion is called PUSH
- Deletion is called POP



Queue

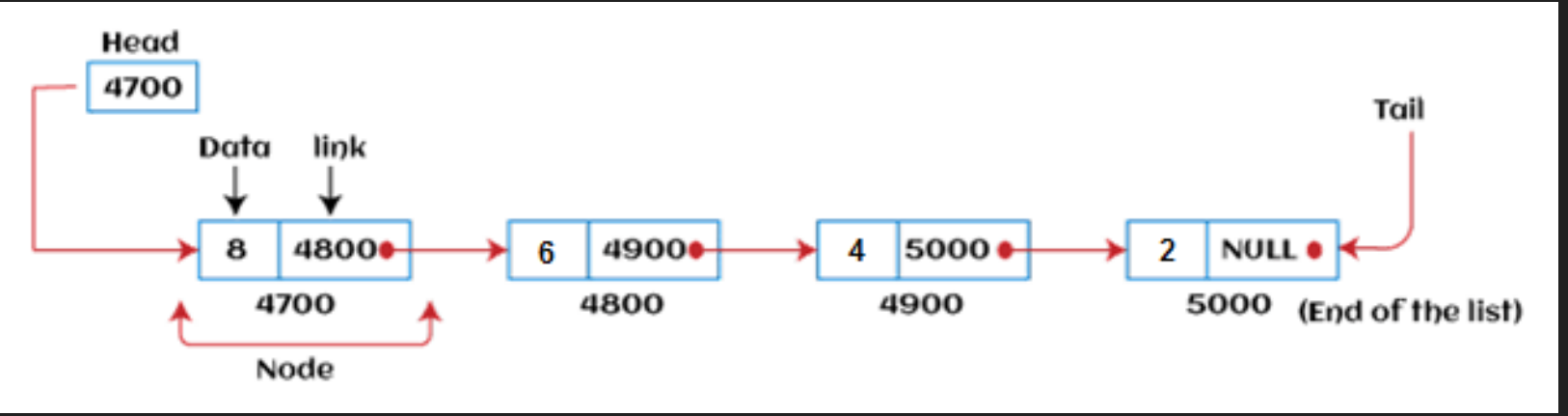
- Like arrays
- Deletion can be done only from one end
- Insertion can be done only from the other end
- First In First Out (FIFO)
- People standing in a line to buy stuff

Queue

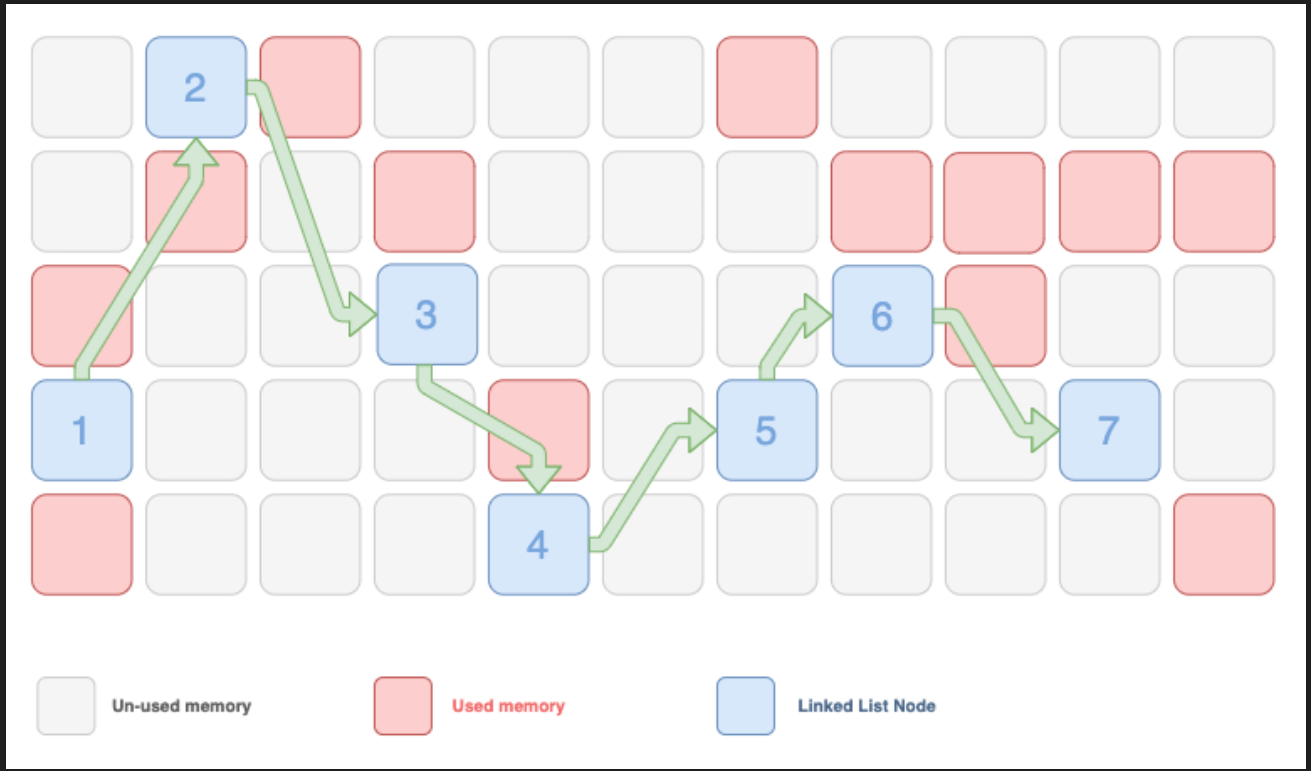
- We have 2 positions
  - Front -> new elements go to front
  - Rear -> elements can be deleted from rear

Linked Lists (list)

- Collection of variable number of data items
- Each element has 2 fields
  - Data / Information
  - Address of the next element
- Each element is called a node



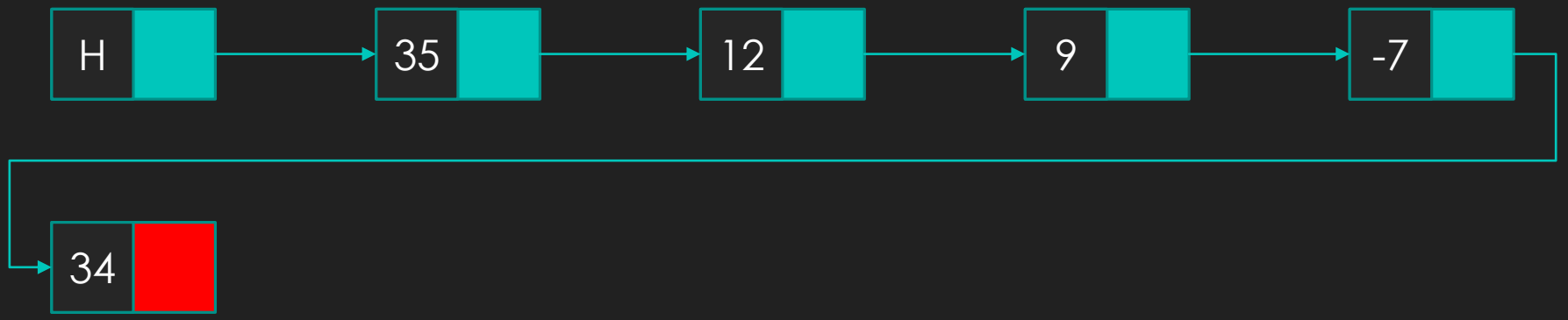
Linked list in RAM



Linked Lists Insert new data at the end

34

Linked Lists Insert new data at the end

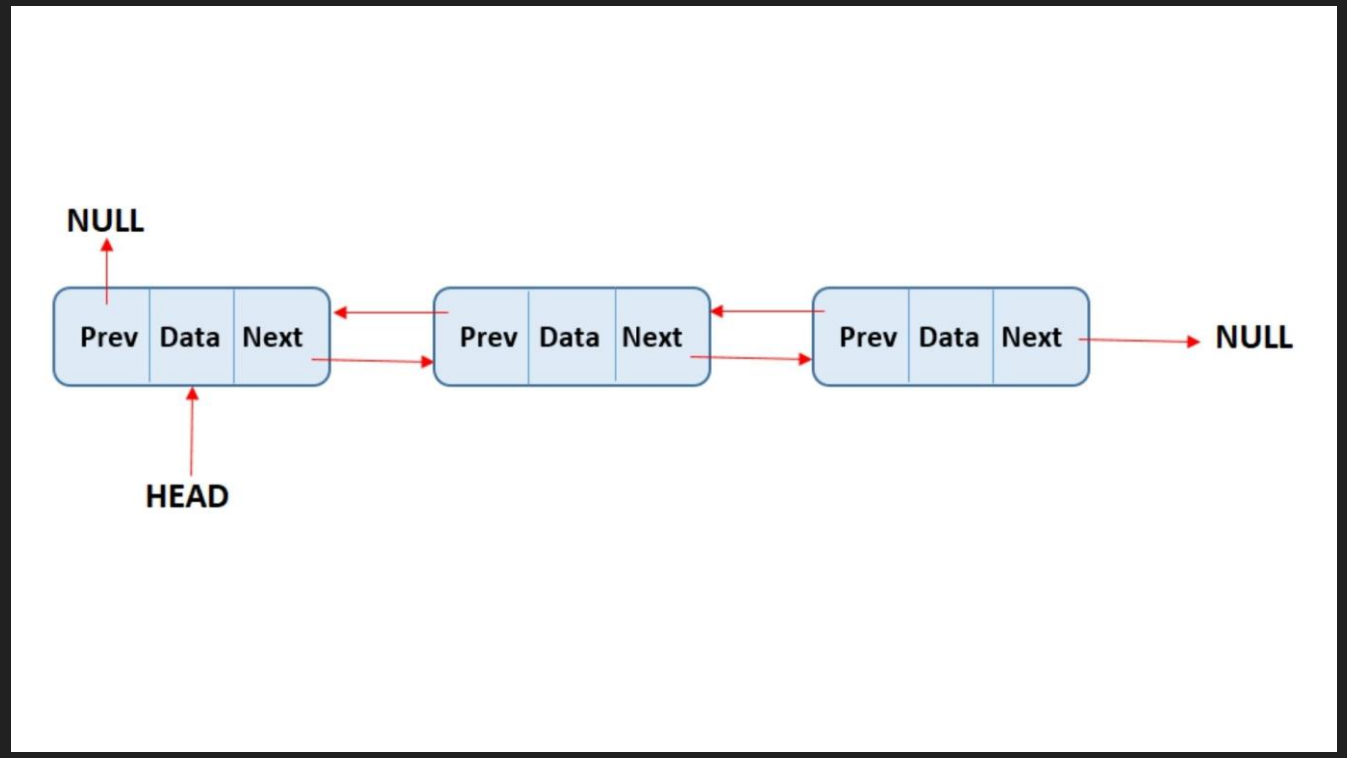


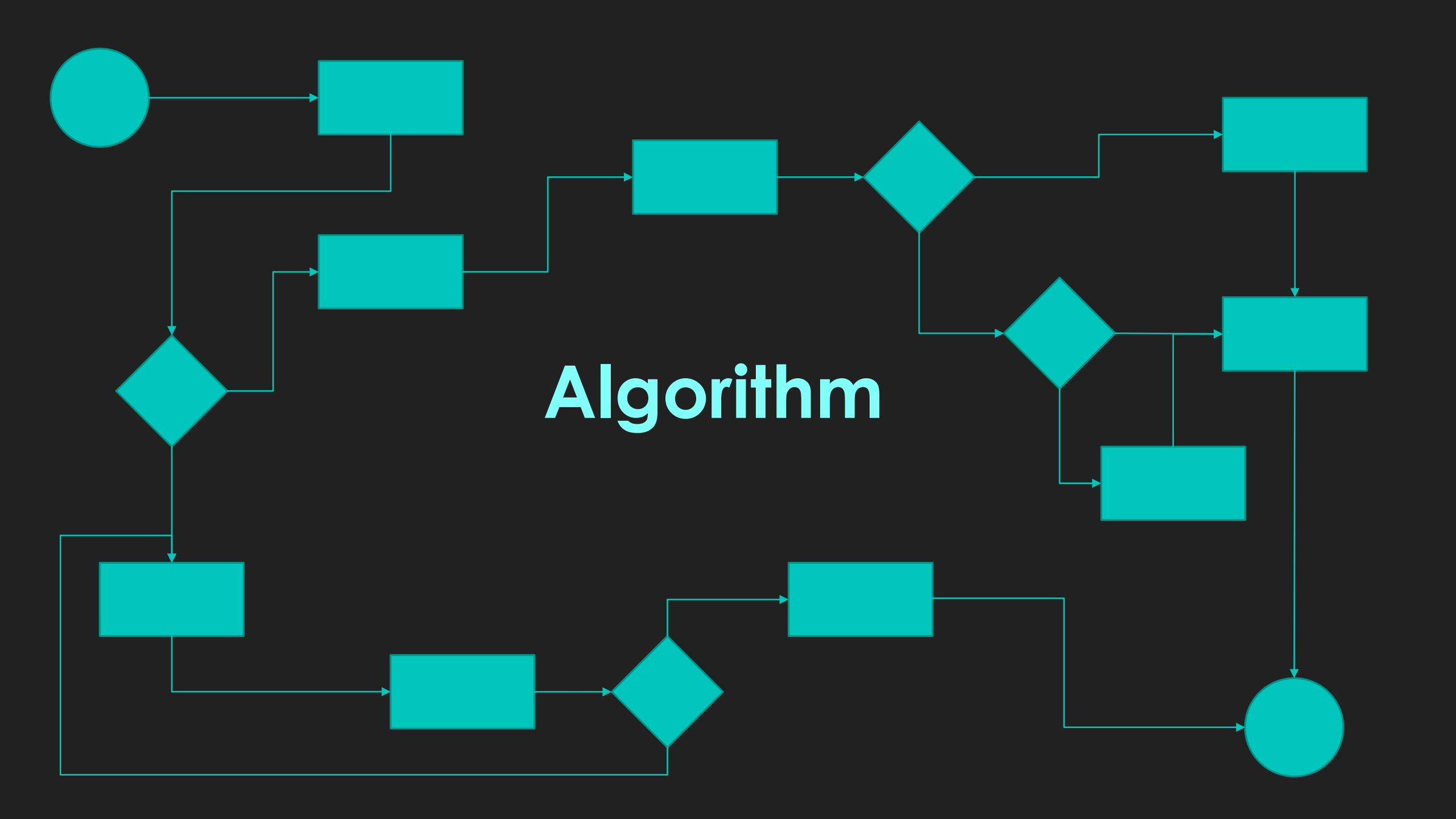
Exercises

- Find specific element in linked list
- Insert new value after specific element in linked list
- Remove a value from linked list
- Merge 2 linked lists
- Reverse a linked list

- https://antoniosarosi.github.io/Linked-List-Visualization/

Doubly linked list




Algorithm

Algorithm and Data Structure

- Program = Algorithm + Data Structure (both implemented by a programming language)
- Algorithm
  - Step by step procedure to solve a particular problem
  - Set of instructions
- Data Structure
  - The way you store data
- To develop a program of an algorithm, we should select an appropriate data structure for that algorithm.
- Therefore algorithm and its associated data structures form a program.

Algorithm

- The first step in programming is to solve the problem
- Solving a problem
  - In math
  - In physics
  - In computer science

Characteristics of an Algorithm

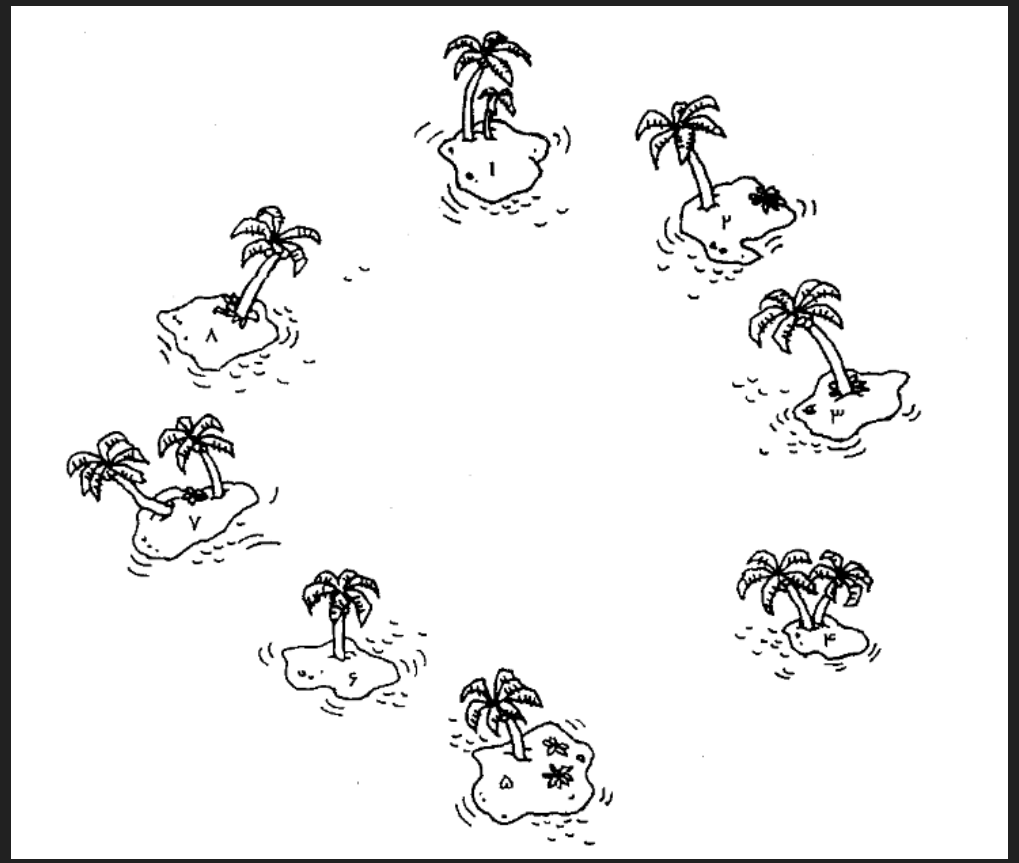
- Well-defined inputs
- Well-defined outputs
- Clear and Unambiguous
- Finite
- Language Independent
- Feasible

Properties of Algorithm

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic
  - giving the same output for the same input case.
- Every step in the algorithm must be effective
  - every step should do some work.

Problem solving example

- How to get a fare on one route only?
- 2 new ships resulting in maximum 3 rides?

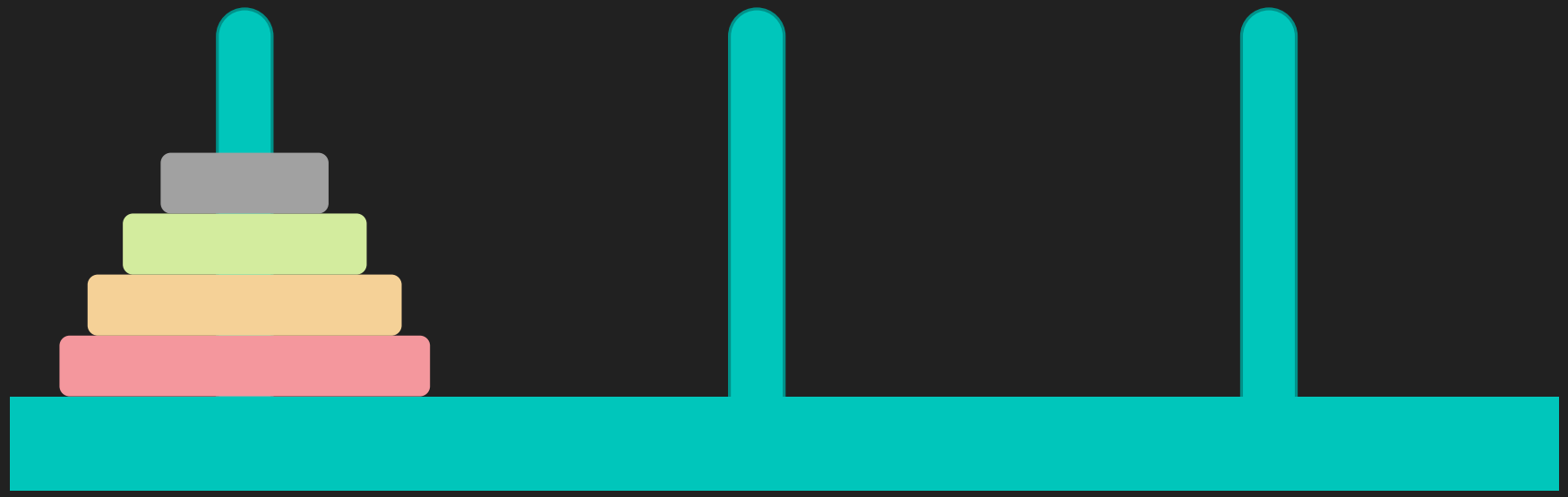


Problem solving example

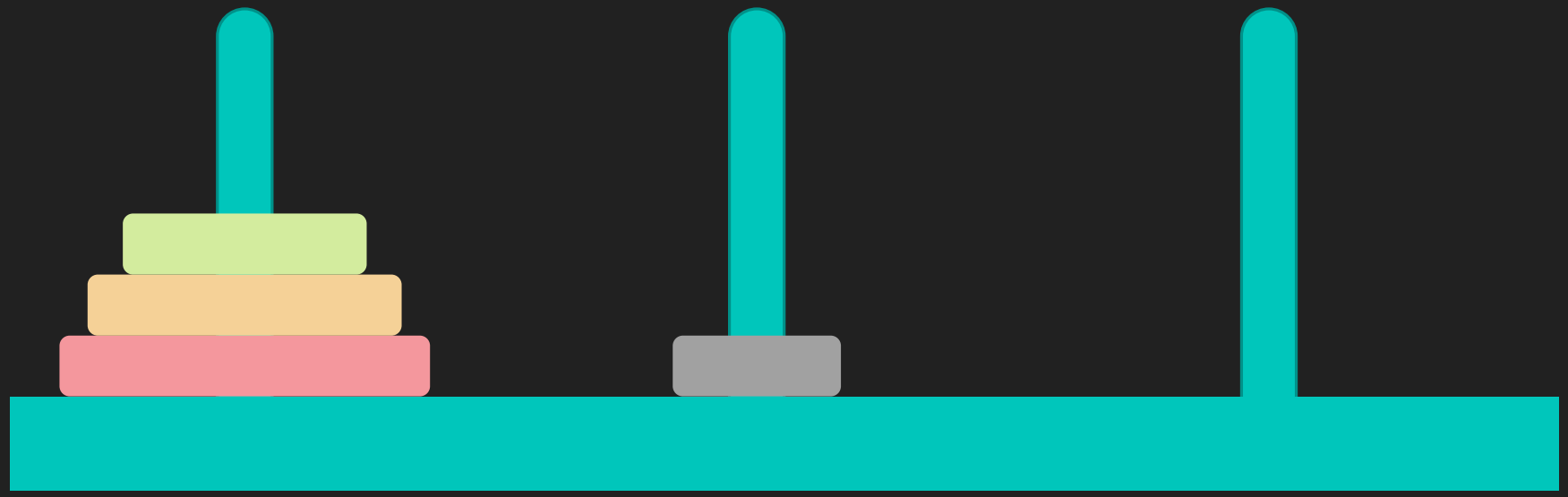
- New programming language with only 2 commands
  - Assignment (=)
  - XOR (⊗)
- Each variable is a 7 bit binary number
- How to swap 2 numbers without any other variables?

A	B	A⊗B
0	0	0
0	1	1
1	0	1
1	1	0

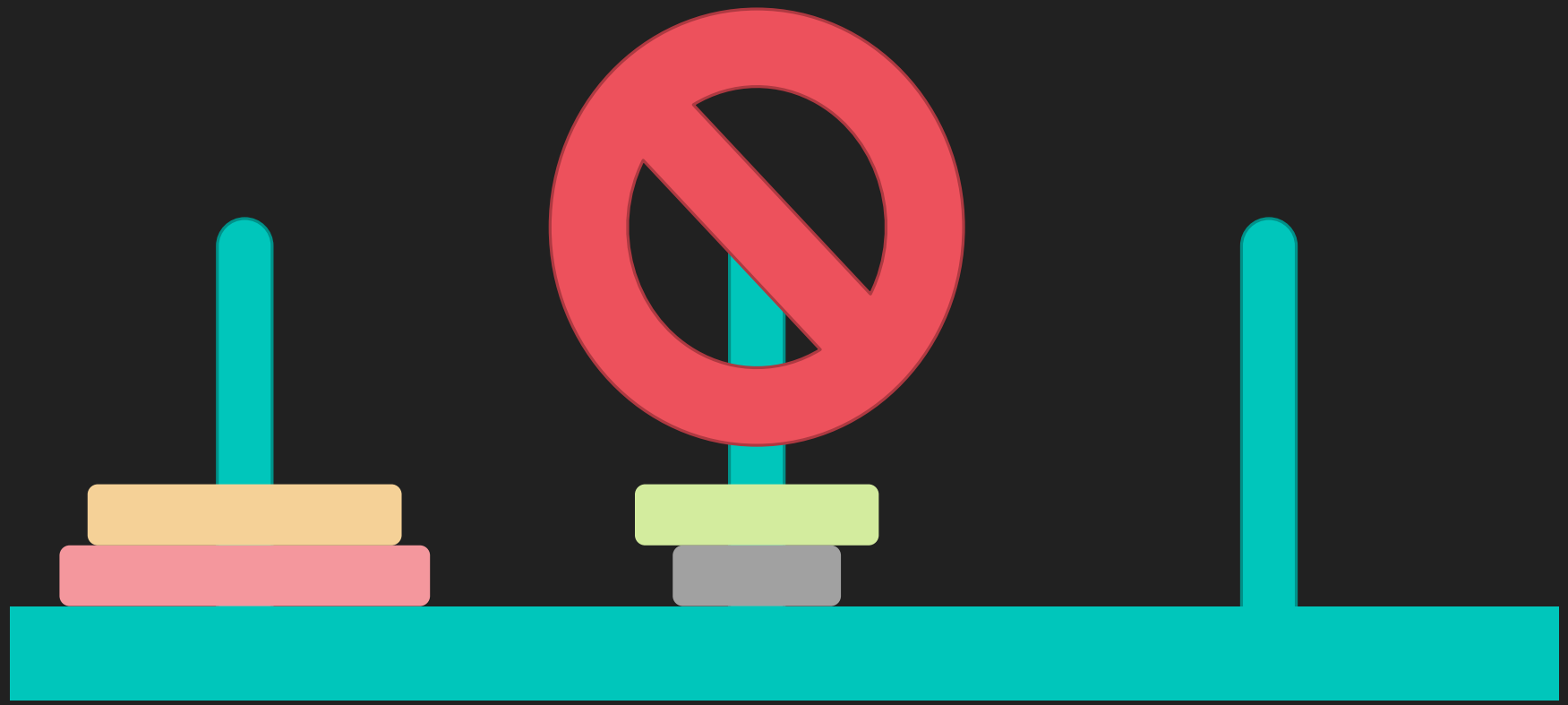
Problem solving example



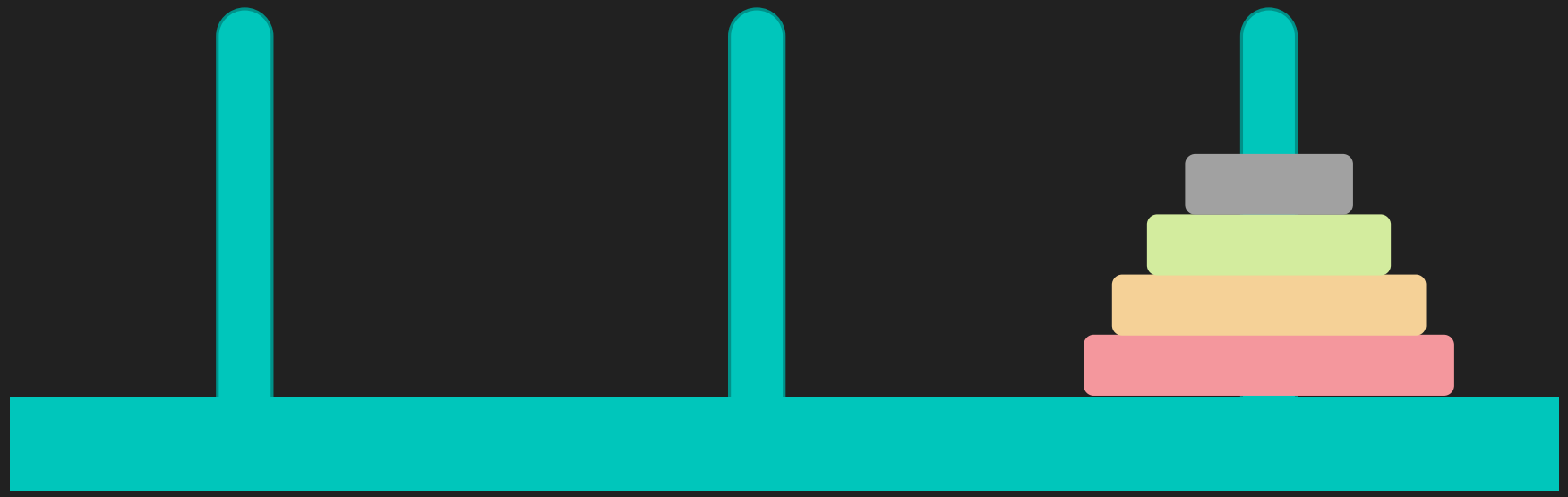
Problem solving example



Problem solving example



Problem solving example



$O(n^2)$

$o(n \log n)$

$O(1)$

time complexity

$\Theta(n!)$

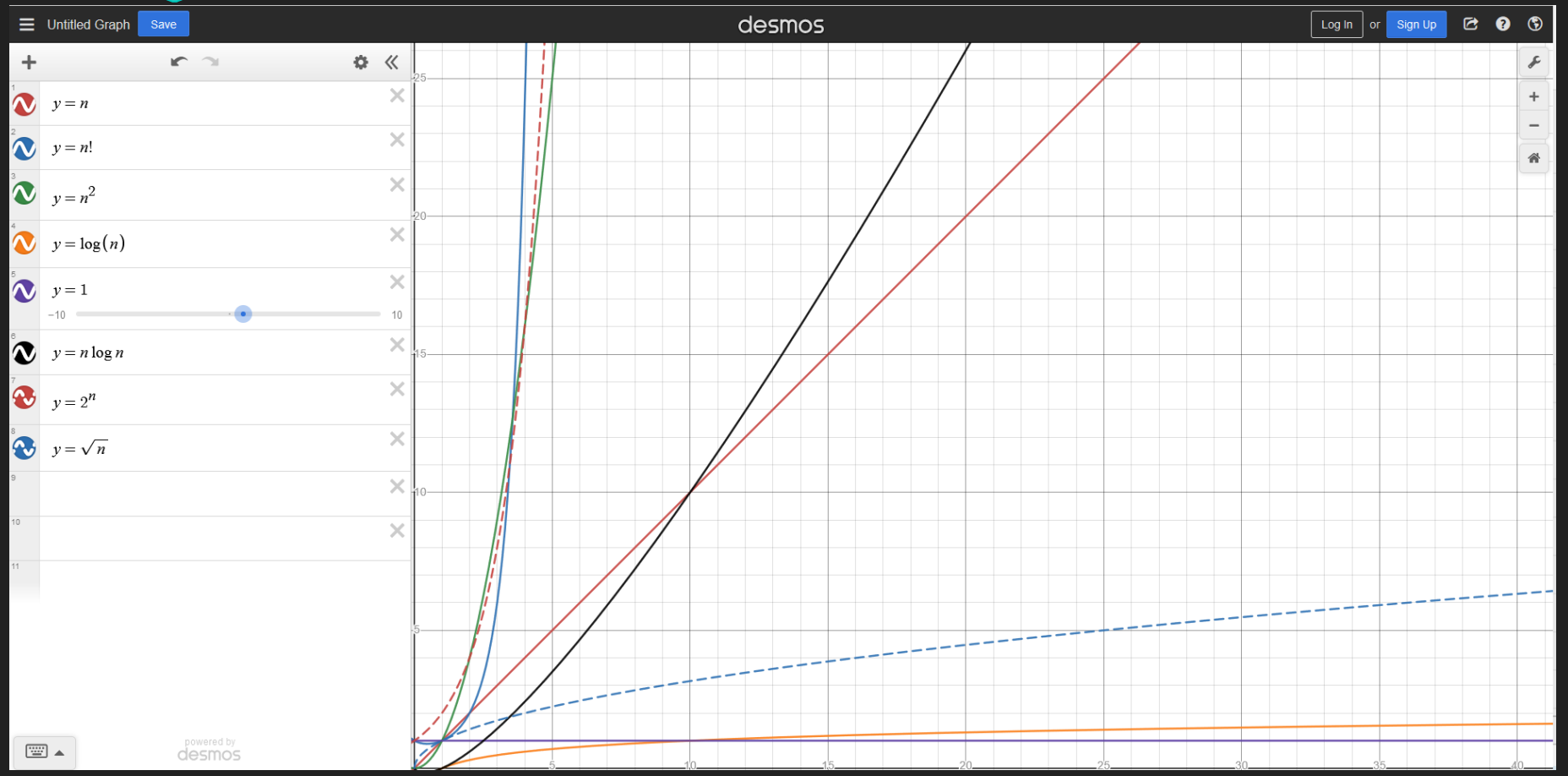
$O(2^n)$

$\theta(\log \log n)$

Complexity

- Sort these values:
  - n
  - $n!$
  - n^2
  - $\log(n)$
  - 1
  - $n \log(n)$
  - 2^n
  - $\sqrt{n}$

- 1
- $\log(n)$
- $\sqrt{n}$
- n
- $n \log(n)$
- n^2
- 2^n
- $n!$


Untitled Graph
Save
desmos
Log In
or
Sign Up
y = n
y = n!
y = n^2
y = log(n)
y = 1
−10
10
y = n log n
y = 2^n
y = √n
powered by
desmos
25
20
15
10
5
5
10
15
20
25
30
35
40

Example

- Imagine a classroom of 100 students in which you gave your pen to one person. You have to find that pen without knowing to whom you gave it.
- We can only ask yes or no question.

- Only one person knows where the pen is (this person maybe isn't the person who has the pen).
- Only the person who has the pen knows
- Everybody knows but wont tell me until I guess correctly

Example

- Imagine a classroom of 100 students in which you gave your pen to one person. You have to find that pen without knowing to whom you gave it.

- $O(n^2)$: ask everyone about everyone else.
- $O(n)$: ask each person individually.
- $O(\log n)$: divide and ask.

Example

```
a = [12, 45, 9, 4, 14, 6]

s = 0
i  = 0
while i < len(a):
        s += a[i]
        i += 1
print(s)
```

1+1+n+n+n+n+n+1 = 5n + 3 = O(n)

Notation

- Big O: Upper bound
  - Function is smaller than …
- Big Omega: Lower bound
  - Function is larger than …
- Big Theta: Tighter bound
  - Big O and Big Omega

Constant times O(1)

- a = 2
- a = b * 3 + c / 7
- a[12] = a[11] + a[10]
- if x < 2
- if x % 4 = 1

…

Analyzing loops

- How many iterations are performed?
- How many steps per iteration?
- Loop for N times:
  - O(N)

Example

```
sum = 0
for i in range(100):
    sum += a[i]
print(sum)
```

Example

```
sum = 0
for i in range(100):
    sum += a[i]
print(sum)
```