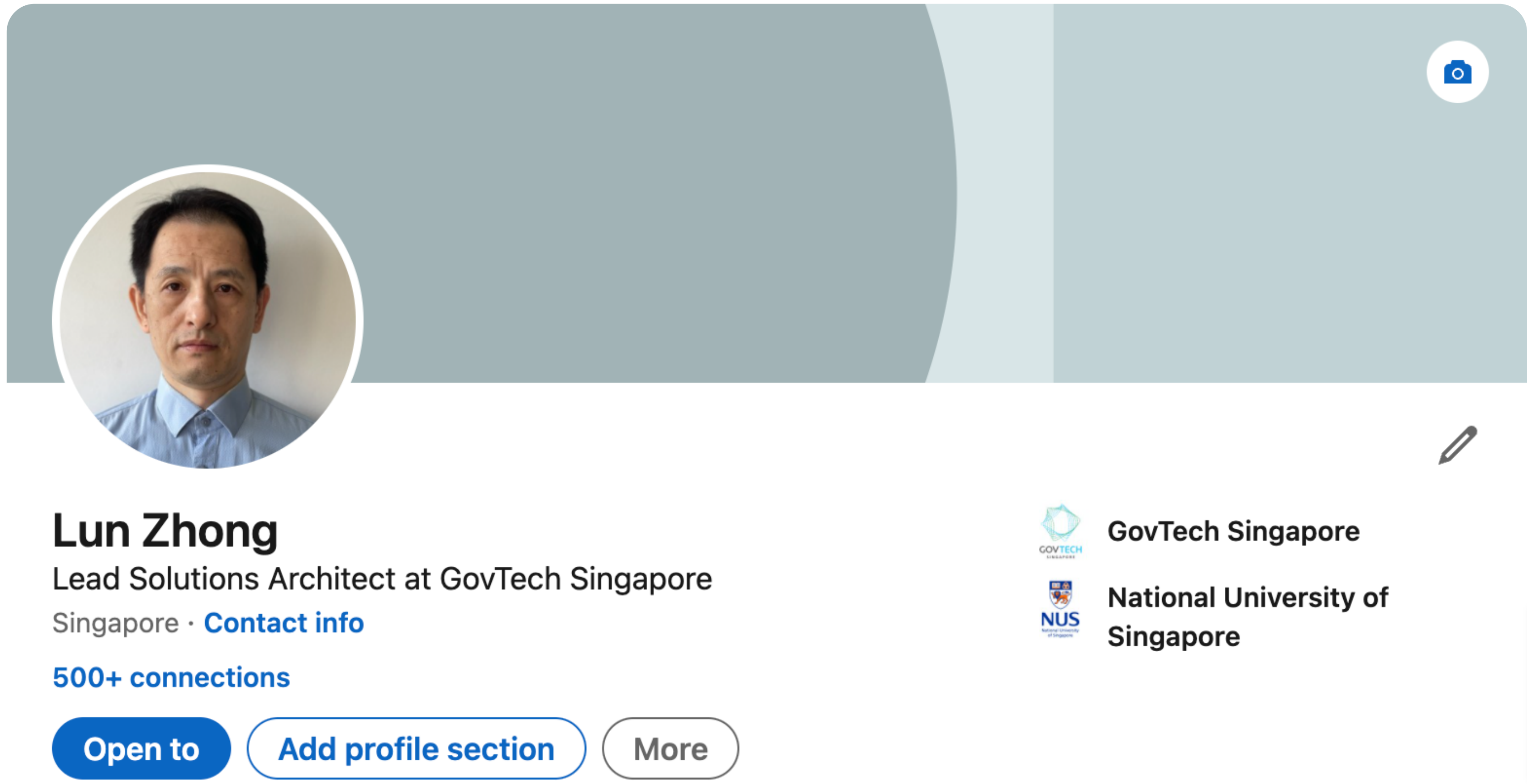
MTH251

Data Structures and Algorithms I

Zhong Lun

About Me

https://www.linkedin.com/in/zhonglun/
zhonglun@gmail.com
9647 7009



Course Structure

Learning Objectives (Jan ~ Mar 6 weeks, 6 lectures & 6 labs):

1. **Python**, **Complexity** & **Big O**
2. **Array**, **Stack**, **Queue**, and **Recursion**
3. **Linked List**
4. **Tree**
5. **Lineary Search**, **Binary Search**, and Algorithm Design & Pattern
6. Review and more

3 assignments & open book exam:

Assessment	Description	Weight Allocation
Assignment 1	Tutor-Marked Assignment 1	10%
Assignment 2	Tutor-Marked Assignment 2	10%
Assignment 3	Tutor-Marked Assignment 3	10%
Examination	Open book exam	70%
TOTAL		100%

Slides & Notebooks

slides online: https://mth251.fastzhong.com/

https://mth251.fastzhong.com/mth251.pdf

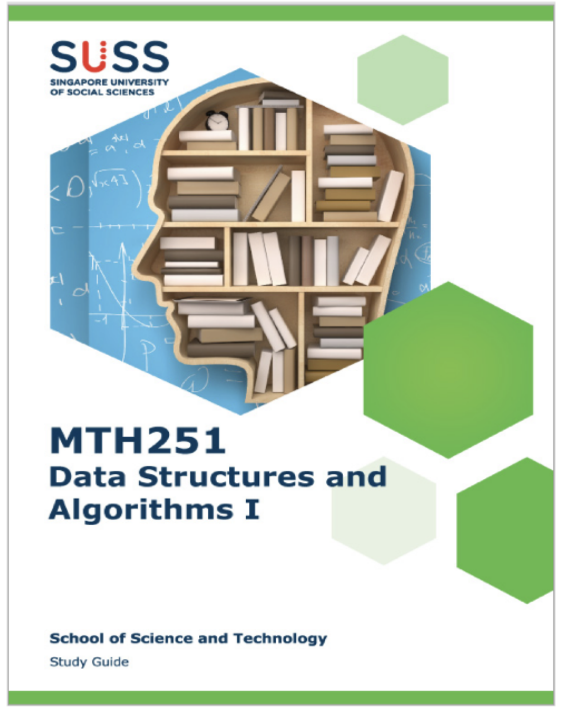
labs: https://github.com/fastzhong/mth251/tree/main/public/notebooks

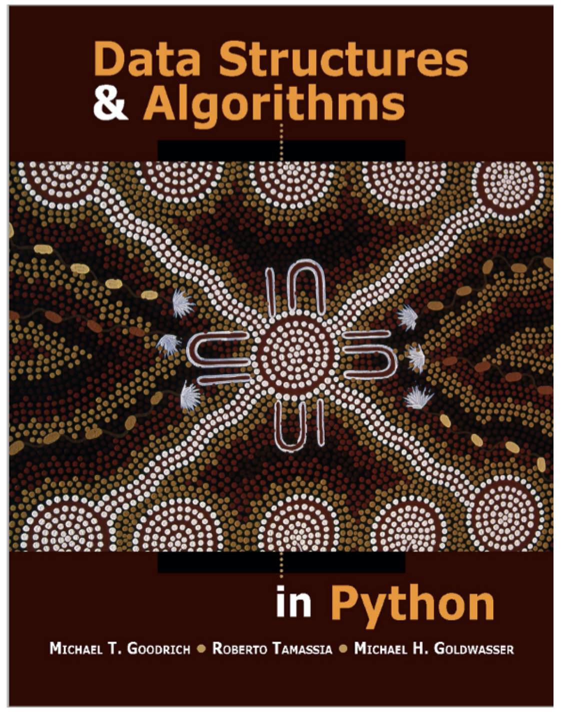
🏃 learning by doing, implementing the algo from scratch

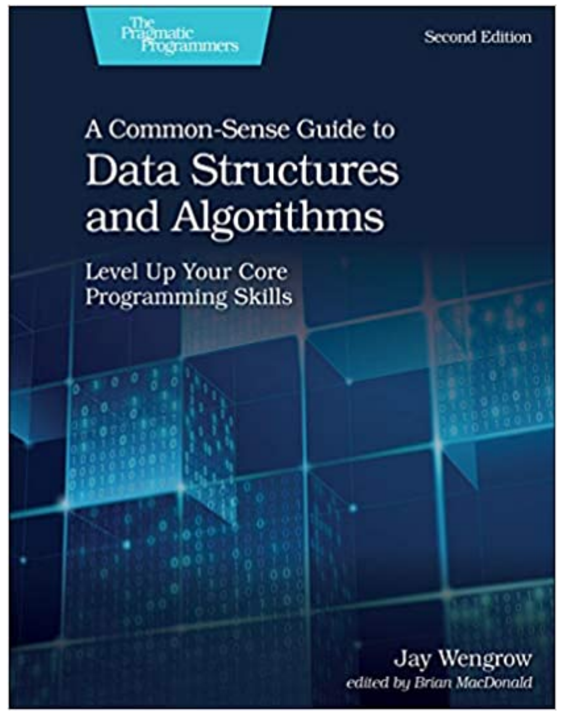
🙇 problem solving

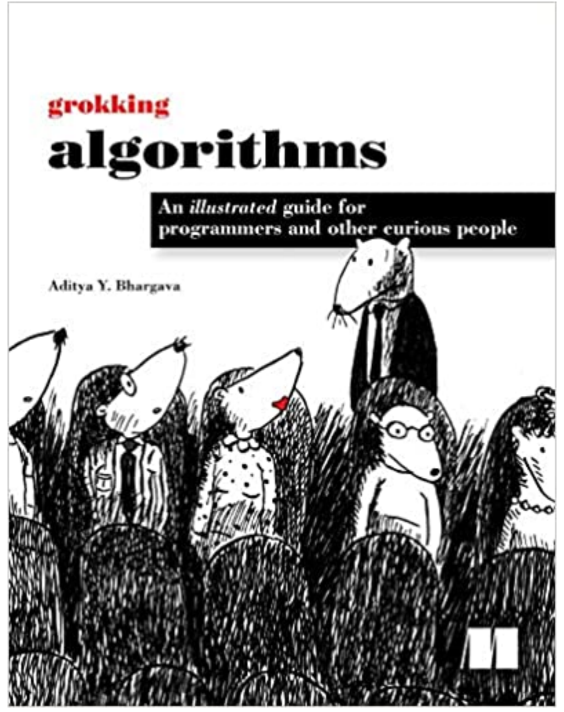
Learning Resource

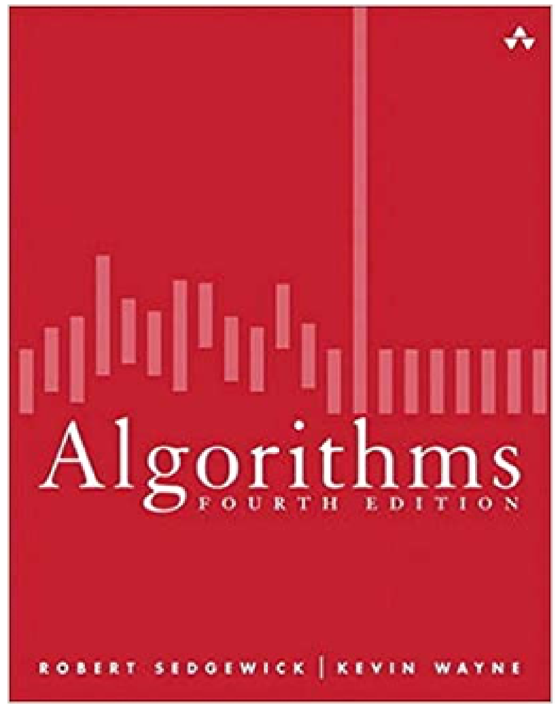
📚 books

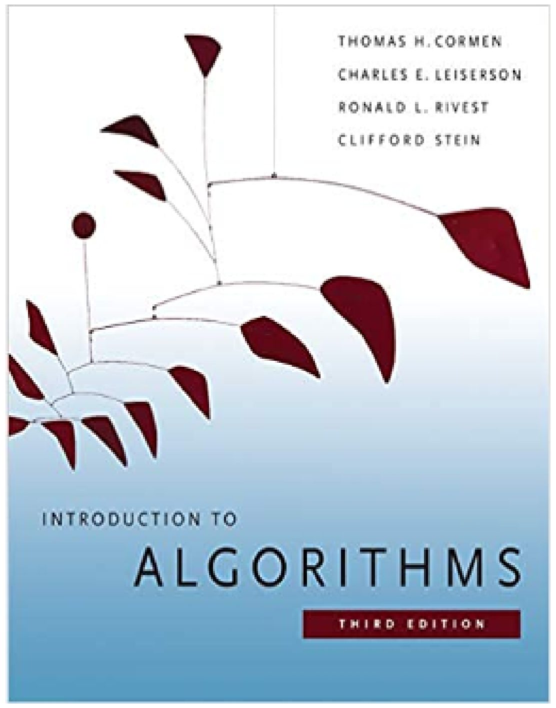












Learning Resource

if you want to dive deeper into proofs and the mathematics of computer science:

📚 Building Blocks for Theoretical Computer Science by Margaret M. Fleck

Clarification

Solution related to DSA questions:

⚠ always seek the best time and space complexity by appling DSA taught in MTH251 & MTH252

⚠ in principle, only the standard ADT operations allowed to use by default as the solution has to be language indenpendent

⚠ advanced features and built-in functions from Python not allowed if not clearly asked by the question, e.g. sort/search/find (in)/min(list)/max(list)/set/match ... , as the complexity becomes unknown and Python dependent

Python

Why Python

> *The TIOBE Programming Community index is an indicator of the popularity of programming languages.*

Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	^	Python	11.27%	-0.00%
2	1	v	C	11.16%	-5.79%
3	2	v	Java	10.46%	-2.11%
4	4		C++	7.50%	+0.57%
5	5		C#	5.26%	+1.10%
6	6		Visual Basic	5.24%	+1.27%
7	7		JavaScript	2.19%	+0.05%
8	10	^	SQL	2.17%	+0.61%

Why Python

✔ Easy To Learn

✔ Human Readable

✔ Productivity

✔ Cross Platform

> **The Zen of Python, by Tim Peters**
>
> Beautiful is better than ugly.
>
> Explicit is better than implicit.
>
> Simple is better than complex.
>
> Readability counts.
>
> Special cases aren’t special enough to break the rules.
>
> There should be one-- and preferably only one --obvious way to do it.
>
> If the implementation is hard to explain, it’s a bad idea.

Python Jobs

✔ backend: **Python** vs. Java, C++, Go, Php

✔ devops: **Python** vs. Go, Ruby, Shell

✔ test automation: **Python** vs. Groovy, shell

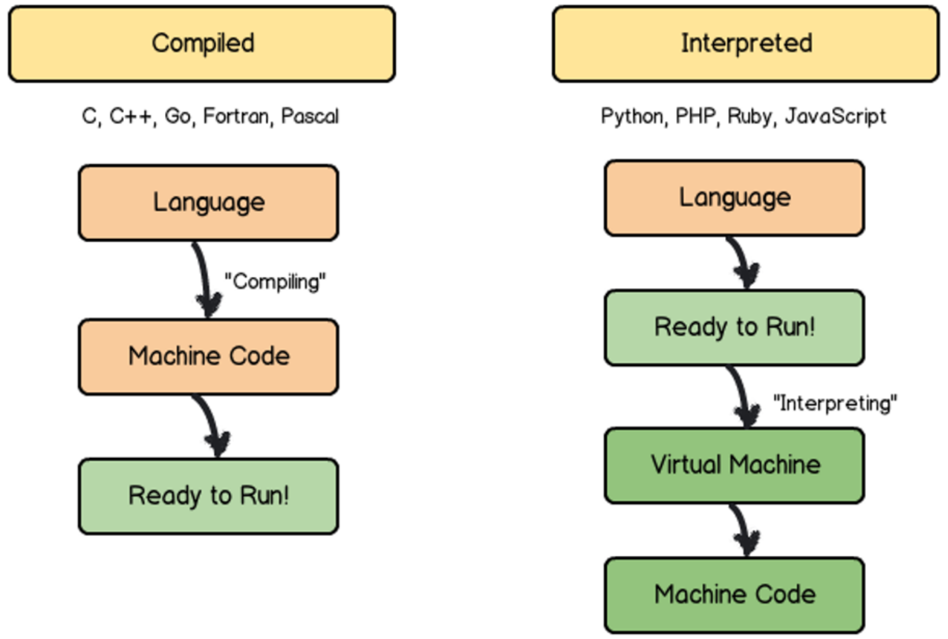
✔ data engineering: **Python** vs. Java, C++

✔ data analytics & visualization: **Python** vs. R, Java, C++

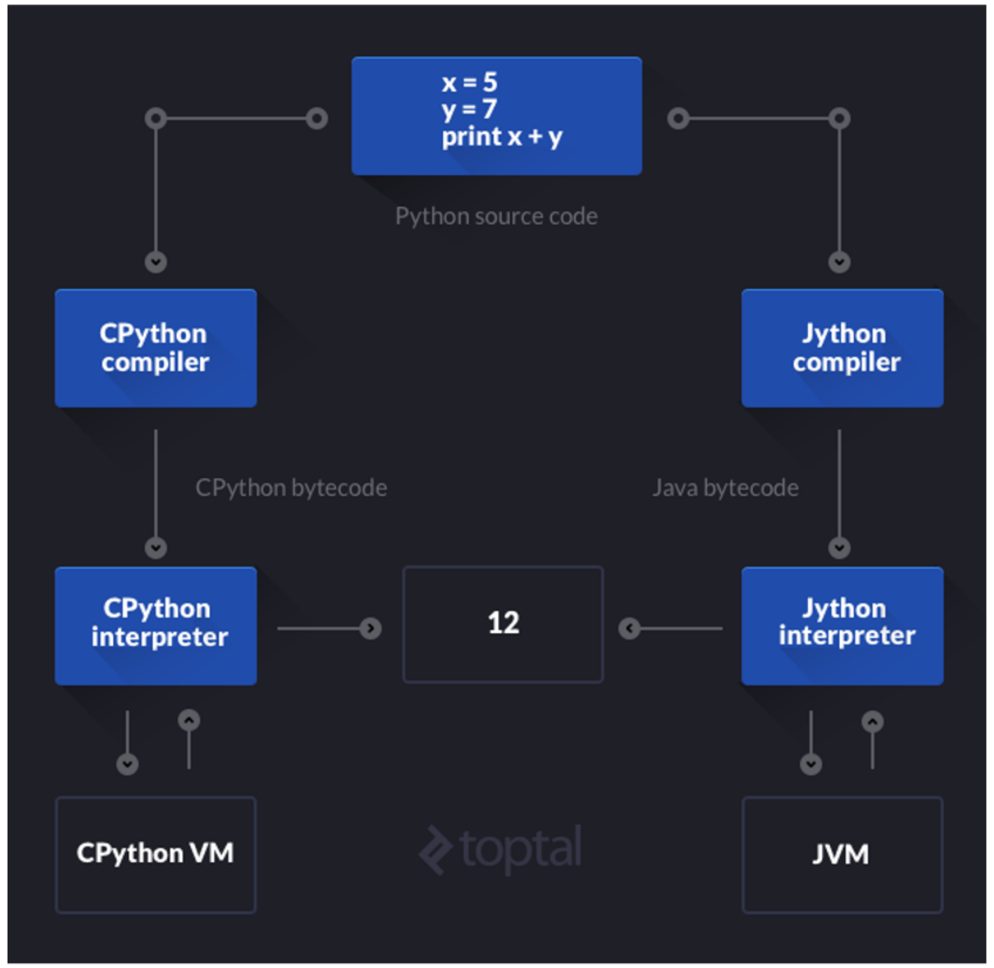
✔ data science & machine learning: **Python** vs. R, Julia, C++

Python 101

- Compiled vs. Interpreted



- CPython bytecode



- Python implementations

Implementation	Virtual Machine	Ex) Compatible Language
CPython	CPython VM	C
Jython	JVM	Java
IronPython	CLR	C#
Brython	Javascript engine (e.g., V8)	JavaScript
RubyPython	Ruby VM	Ruby

Python Data Type & Operators

- numbers: `int` `float` `complex`
  - arithmetic operator: `+` `-` `*` `/` `//` `%` `**`
  - bitwise operator: `&` `|` `^` `>>` `<<` `~`
  - `range()`: a list of integers
- strings: `''` `""` `'` `"` `\t` `\n` `\r` `\\` etc.
  - `join()` `split()` `ljust()` `rjust()` `lower()` `upper()` `lstrip()` `rstrip()` `strip()` etc.
- boolean: `True` `False`
  - True: non-zero number, non-empty string, non-empty list
  - False: 0, 0.0, "", [], None

Python Data Type & Operators

- boolean: `True` `False`
    - logic operator: `and` `or` `not`
    - comparison operator: `>` `<` `>=` `<=` `==` `!=`
    - identity operator: `is` `is not`
- None
- type conversion/casting: `int()` `float()` `str()` `bool()` `hex()` `ord()`

Python Collections

- collections: `list` `tuple` `set` `dictionary`
  - membership operator: `in` `not in`
- list []: a collection of items, usually the items all have the same type
  - sequence type, sortable, grow and shrink as needed, most widely used
- tuple (): a collection which is ordered and unchangeable
- set {}: a collection which is unordered and unindexed
- dictionary: a set of `key: value` pairs, unordered, changeable and indexed

Python Program Structure

- variable
- statement & comments
  - Python uses new lines to complete a command, as opposed to other programming languages often use `;` or `()`; relies on indentation (whitespace sensitive), to define scope, such as the scope of loops, functions and classes, as opossed to other programming languages often use `{}`
- control flow
  - `if … elif … else`
  - `while` `for` `break` `continue`

Python Program Structure

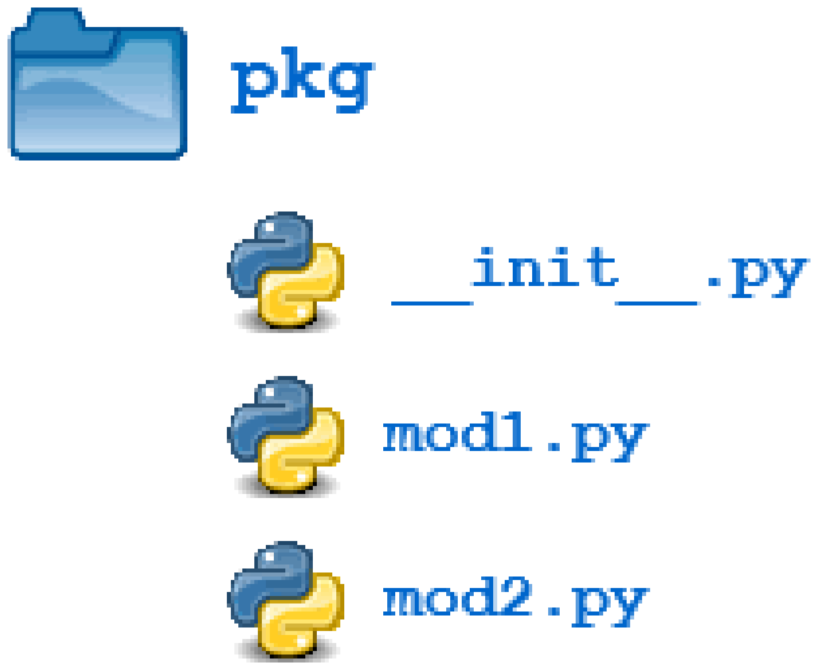
- function
  - `def` `return`
  - *`main`*
  - advanced:
    - lambda
    - decorator
    - closure
- error/exception
  - handling exception: `try … except … else … finally`
  - raise execption: `raise`

Python OO & Class

- `Procedural` vs. `OOP` vs. `FP`
- OO Principal
    - `Inherience`
    - `Encapsulation`
    - `Polymorphism`
- class, instance, attributes, properties, method
- `override` vs. `overload` vs. `overwrite`

Misc.

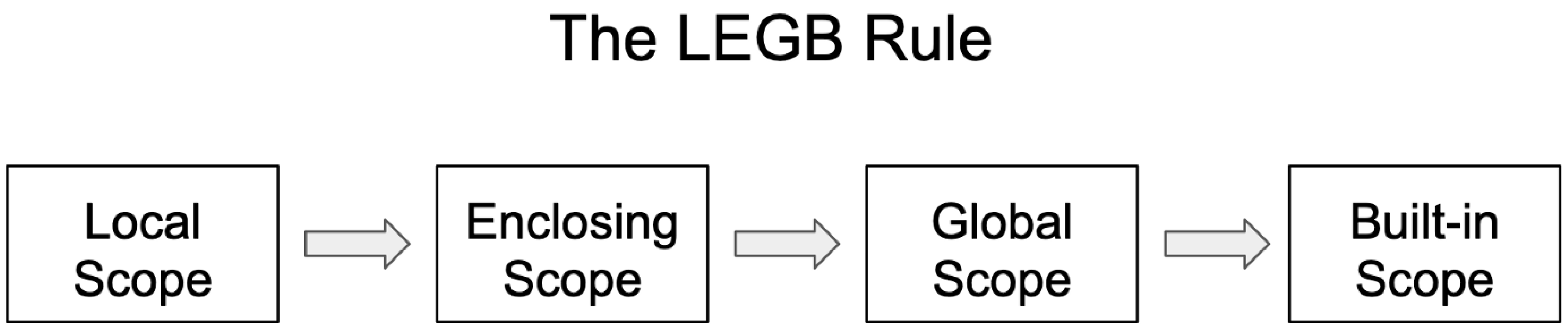
- modular programming: `function` → `class` → `module` → `package`
- Modules: Python module (default main module), C module, Build-in module
- Packages:



- Standard Lib: `math` `random` `re` `os` `itertools` `collections`

Misc.

- Namespaces & Scopes: `LEGB` rule



- memory, copy vs. deepcopy

- help

Cheatsheet:

Python Crash Course - Cheat Sheets

Comprehensive Python Cheatsheet

Python Tutorials

Programming with Mosh

- Python Tutorial - Python for Beginners 2020

freeCodeCamp

- Learn Python - Full Course for Beginners Tutorial
- Python for Everybody - Full University Python Course
- Intermediate Python Programming Course

Tech With Tim

- Learn Python - Full Course for Beginners Tutorial

The Hitchhiker’s Guide to Python!

DSA & Complexity

Data Structure & Algorithms (DSA)

A data structure is a way of organizing information so that it can be used effectively by computer

Algorithms provides computer step by step instructions to process the information and solve a problem

Program = Data Structure + Algorithm

- Input
- Output
- Definiteness
- Effectiveness
- Finiteness

Data Structure & Algorithms

Example: keyword searching

Data Structure: index

Algorithm: looking up the page no.

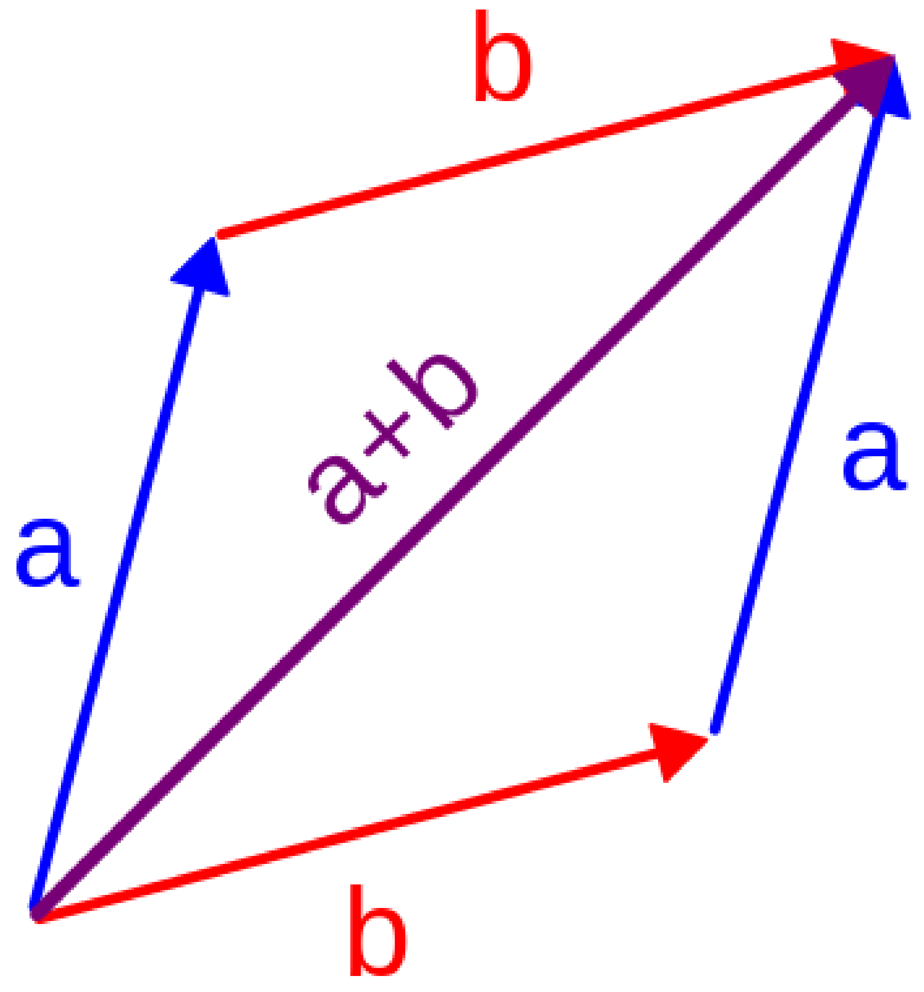
Data Structure & Algorithms

Example: adding two vectors

Data Structure: complex number

Algorithm: adding two complex numbers

$a + b = (x + yi) + (u + vi) = (x + u) + (y + v)i$



Data Structure & Algorithms

Data Structure

- Linear
  - Array, String, Linked List
  - Stack, Queue, Deque, Set, Map/Hash, etc.
- Non-Linear
  - Tree, Graph
  - Binary Search Tree, Red-Black Tree, AVL, Heap, Disjoin Set, Trie, etc.
- Others
  - Bitwise, BloomFilter, LRU Cache

Algorithms & Data Structure

Algorithms

- branching: if-else, switch
- iteration: for, while loop
- recursion: divide & conquer, backtrace
- searching: binary search, depth first, breath first, A*, etc.
- sorting: quick sort, bubble sort, merge sort, etc.
- dynamic programming
- greedy
- …

Algorithms & Data Structure

Why

✓ deeper understanding of computer system

✓ improve coding skill

✓ coding interview

✓ building powerful framework and library

How

🏃‍♀️ learning by doing, implementing from scratch

🙇 problem solving

Algorithm Complexity Analysis

How to measure Performance/Efficiency ?

- cpu, memory, io, networking, etc.
- no. of lines
- worst case vs. best case
- code slows as data grows
- ...

Algorithm Complexity Analysis

Time Complexity : by giving the size of the data set as integer N, consider the number of operations that need to be conducted by computer before the algorithm can finish

Space Complexity : by giving the size of the data set as integer N, consider the size of extra space that need to be allocated by computer before the algorithm can finish

Good code:

✔️ readability
✔️ speed
✔️ memory

👉 When: Accessing, Searching, Inserting, Deleting, ...

Big-O

Lef f(n) and g(n) be functions from positive integers to positive integers to positive reals
f = O(g) if there is a constant c > 0 such that f(n) ≦ c·g(n) for large n

💬 f(n) grows no faster than g(n)

e.g.

$f(n) = O(4n^2 + 8n + 16) \to O(n^2) \to g(n) = n^2$

$O(4n^2 + 8n + 16) = O(n^2)$

> *Big-O describes the trend of algorithm performance when the data size increases*

Big-O

$O(1)$:	constant
$O(\log_* n)$:	logarithmic
$O(n)$:	linear
$O(nlog_*n)$:	linearithmic
$O(n^2)$:	polynomial
$O(2^n)$:	exponential
$O(n!)$:	factorial

Big-O

👉 Master theorem (analysis of algorithms)

$O(f) = f$

$O(c \cdot f) = O(f)$

$O(f + g) = O(max(f, g))$

$O(f) \cdot O(g) = O(f \cdot g)$

$O(f \cdot g) \leq O(f \cdot h)$ if & only if $O(g) \leq O(h)$

$O(x^a) \leq O(x^b)$ if & only if $a \leq b$

$O(a^x) < O(b^x)$ if & only if $a < b$

$O(x^c) < O(d^x)$ if & only if $d > 1$ (assuming $c \geq 1$ and $d \geq 1$)

$O(log_* x) < O(x^c)$ if & only if $c > 0$

Big-O

Analysis of Algorithms:

- worst case
- ignore constant factors, lower-order terms
- asymptotic analysis (large input size)

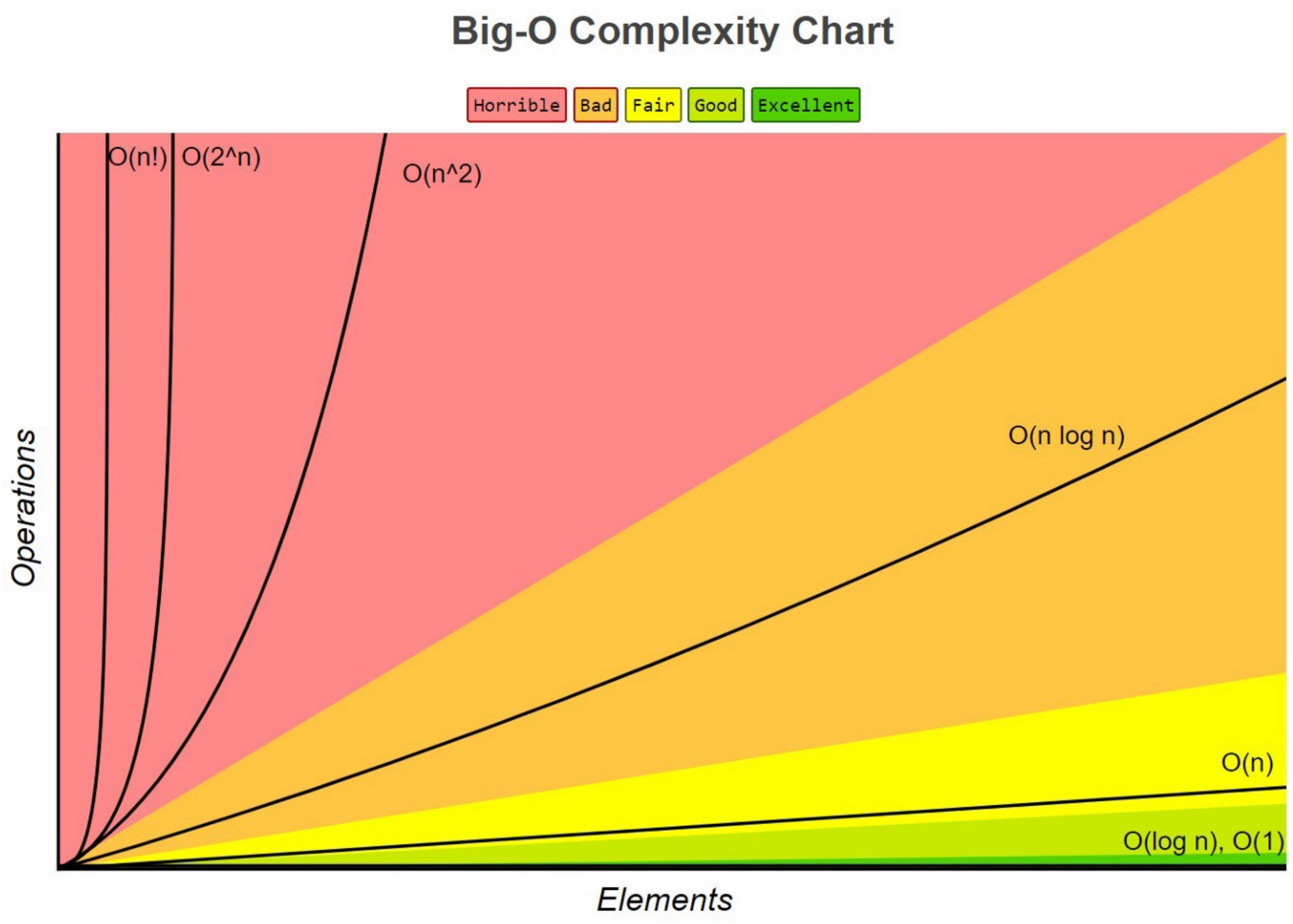
Formally, given functions f(x) and g(x), we define a binary relation: $f(x) \sim g(x)$ (as $x \to \infty$)

if and only if: $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$

asymptotic analysis is used to build numerical methods to approximate equation solutions: asymptotic expansion, asymptotic distribution, ...

Big-O

😊 $O(1) < O(\log_* n) < O(n) < O(n \log_* n) < O(n^2) < O(2^n) < O(n!)$ 🥵



https://www.bigocheatsheet.com/

Array

Array

To store a list of similar things, example:

- A list of names: [“Alex”, “Bob”, “Charles”, “David”]
- A list of numbers: [1, 2, 3, 4]

Each item in the array referred as “**element**”

Array

- Element Type: same type (array is structured data)

- Element Size: fixed

```
# java
String[] cars = {“BMW”, “Toyota”, “Tesla”} // declare & init

Integer[] scores = new Integer[10] // declare
// init
scores[0] = 90
scores[1] = 80
```

- Element Index: 0, 1, ..., length - 1

Array 2-D

```
students = [
    [“Alex”,  “M”, “S1111111A”],
    [“Bob”,   “M”, “S2222222B”],
    [“James”, “M”, “S3333333C”],
]

students[2]     → [“James”, “M”, “S3333333C”]
Students[1][2] → “S2222222B”
```

Index	0	1	2
0	Alex	M	S1111111A
1	Bob	M	S2222222B
2	James	M	S3333333C

Array Address

str = "HELLO" = ['H', 'E', 'L', 'L', 'O']

Memory Address	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	...
Computer's memory	H		E		L		L		O		...
Index Number	0		1		2		3		4		

data type: char

data type size: 2 byte (1 byte = 8 bits, 0000 0000 ~ 1111 1111)

total_size = array_size * data_type_size

array[i].address = base_address + i * data_type_size

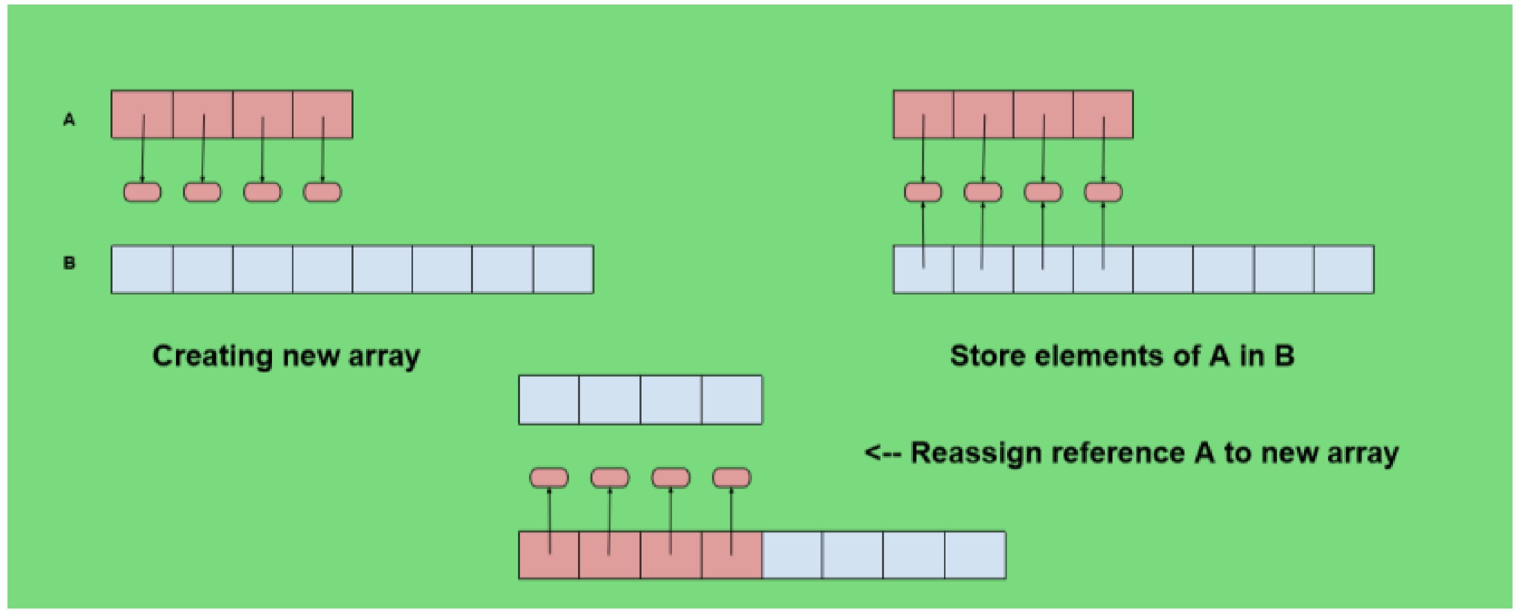
👉 O(1)

Dynamic Array

jupyter

- what is the good space/slot size for an array?

- when is the good time to expand/shrink the array?



Growth Pattern:

- Python: $NewAllocated = (size_t)newsize + (newsize >> 3) + (newsize < 9?3 : 6);$ // 0, 4, 8, 16, 25, 35, 46, 58, 72, 88, ...
- Java: $((size * 3)/2) + 1$
- C#: $size * 2$

👉 memory management is one of the biggest programming challenges.

Array Complexity

Operation	Array	Dynamic Array
Accessing	O(1)	O(1)
Searching	O(n)	O(n)
Inserting	-	O(n)
Deleting	-	O(n)

Python Collection Complexity

List [a, b, c, ...]		Dicts {k:v, ...}		Set {a, b, c,}	
mylist.append(val)	O(1)	mydict[key] = val	O(1)	myset.add(val)	O(1)
mylist[i]	O(1)	mydict[key]	O(1)		
val in mylist	O(N)	key in mydict	O(1)	val in myset	O(1)
for val in mylist:	O(N)	for key in mydict:	O(N)	for val in myset:	O(N)
mylist.sort()	O(NlogN)				

Trade-offs

```
# .. make a list ..
if thing in my_list: # O(N)

# ✅ Good
# .. make a set ..
if thing in my_set:  # O(1)

# ❌ Bad
# .. make a list ..
my_set = set(my_list) # O(N)
if thing in my_set:   # O(1)

# ✅ Good
# .. make a list ..
my_set = set(my_list) # O(N)
for many_times:
  if thing in my_set:   # O(1)
```

ADT vs. Data Structure

An **abstract data type** (ADT) is an abstraction of a **data structure** which provides only the interface to which a data structure must adhere to. The interface does not give any specific details about how something should be implemented - ADT provides implementation-independent view of a data structure.

Programming language provides different **data types** to implement/represent a specific data structure.

- Array - a linear abstract data type
- Array - a java data type
- Dynamic Array - array with changable size
- List - a python data type, more flexible than a dynamic array
- ArrayList/Vector - java data type, implementation of List

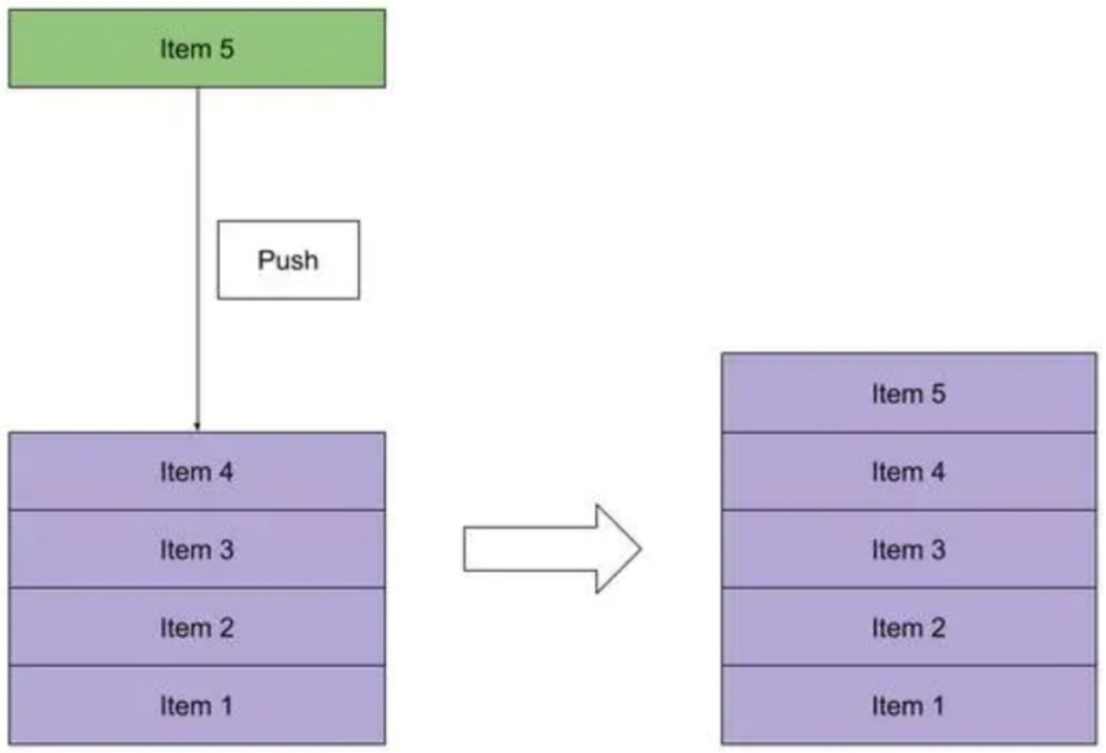
Stack & Queue

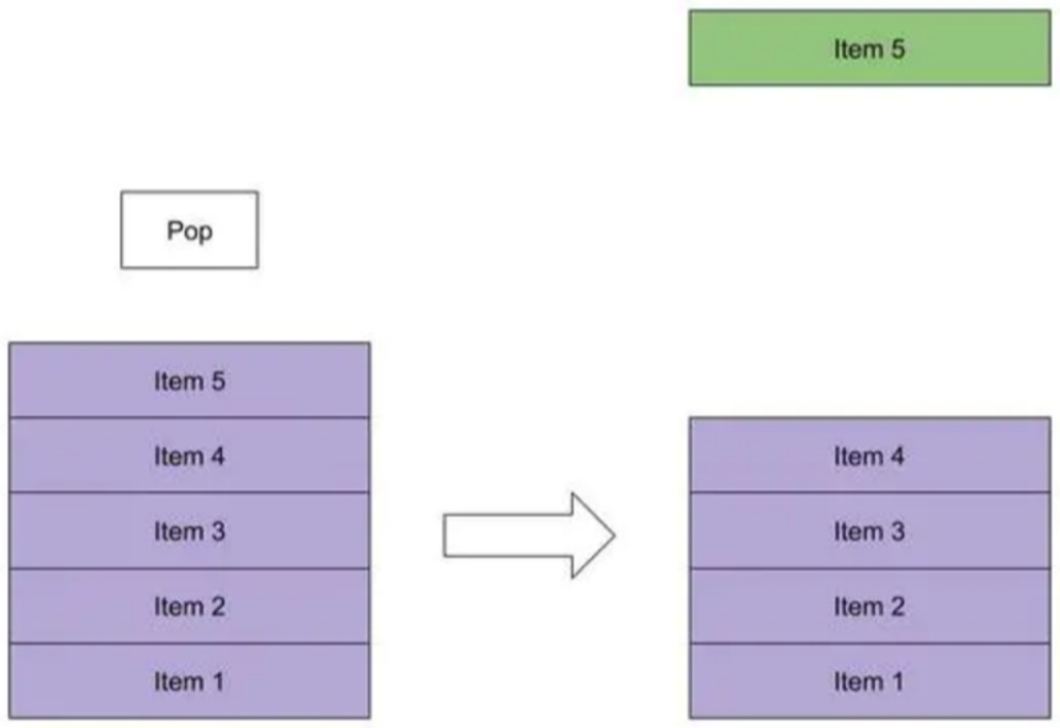
Stack

- Sequential Access vs Random Access (such as Array)
- **LIFO** (Last In First Out) sequential collection

Stack: Operations

- **push**() – pushing (storing) an element on the stack
- **pop**() – removing (accessing) an element from the stack
- top()/peek() – get the top data element of the stack, without removing it
- size(), isEmpty(), isFull()





Stack Complexity

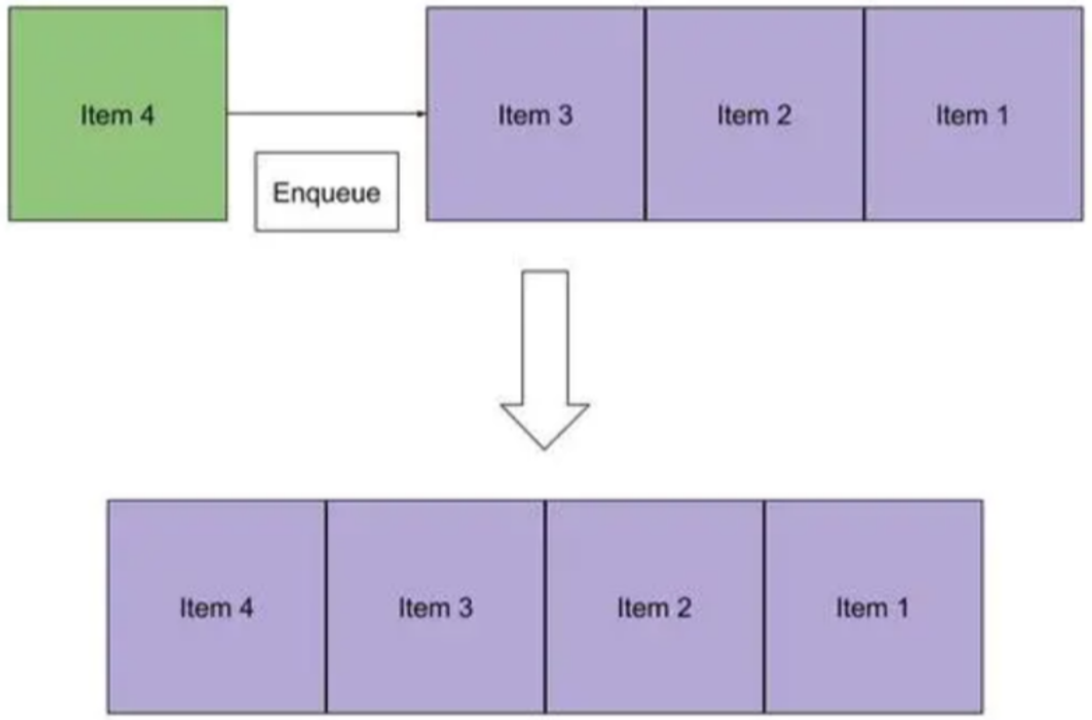
Operation	Stack
Accessing	O(n)
Searching	O(n)
Inserting	O(1) (push)
Deleting	O(1) (pop)

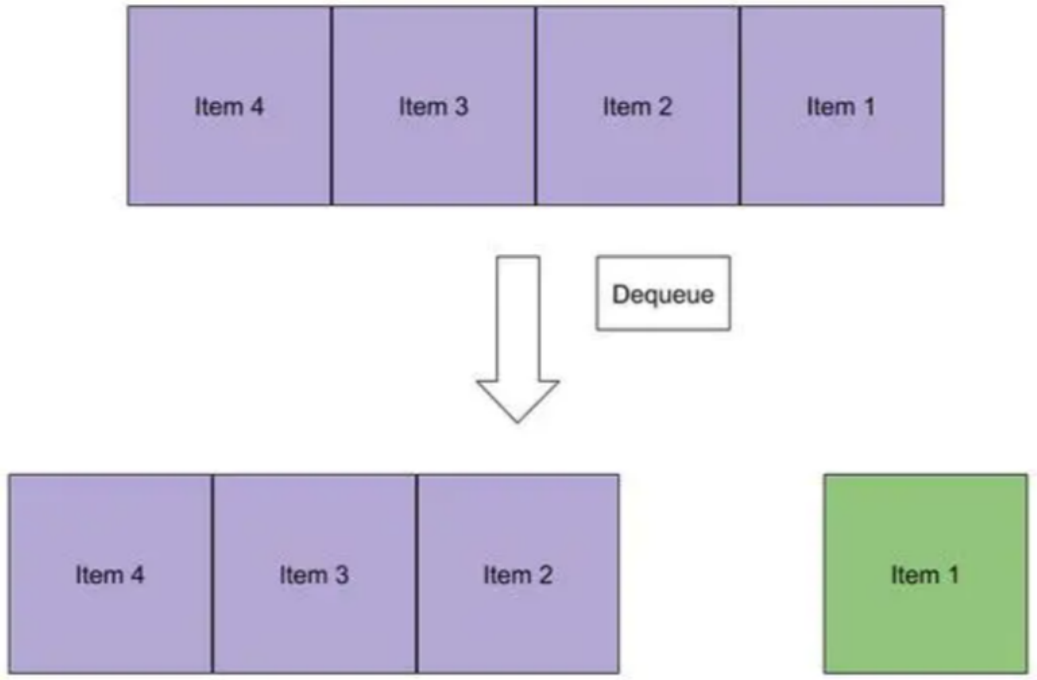
Queue

- **FIFO** (First In First Out) sequential collection

Queue: Operations

- **enqueue**() – adding (storing) an element to the queue
- **dequeue**() – removing (accessing) an element from the queue
- fist()/peek() – get the first element of the queue, without removing it
- size(), isEmpty(), isFull()





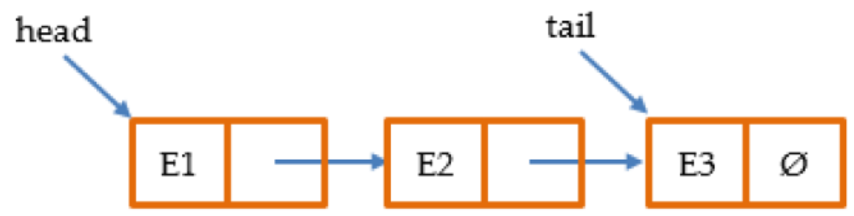
Queue Complexity

Operation	Queue
Accessing	O(n)
Searching	O(n)
Inserting	O(1) (enqueue)
Deleting	O(1) (dequeue)

Linked List

Linked List

- dynamic linear data structure
- each item contains data & pointer
- data stored in a “Node” class
- each item holds a relative position relative to the other items: 1st, 2nd, …, last item



List: Operations (potential)

- add(element)
- append(element)
- insert(position, element)
- delete(element)
- is_empty()
- __len__()
- set(position, element)
- get(position)
- search(element)
- index(element)
- pop()
- pop(position)

Linked List Complexity

Operation	Linked List	Dynamic Array
Accessing	O(n)	O(1)
Searching	O(n)	O(n)
Inserting	O(1) 🤔	O(n)
Deleting	O(1) 🤔	O(n)

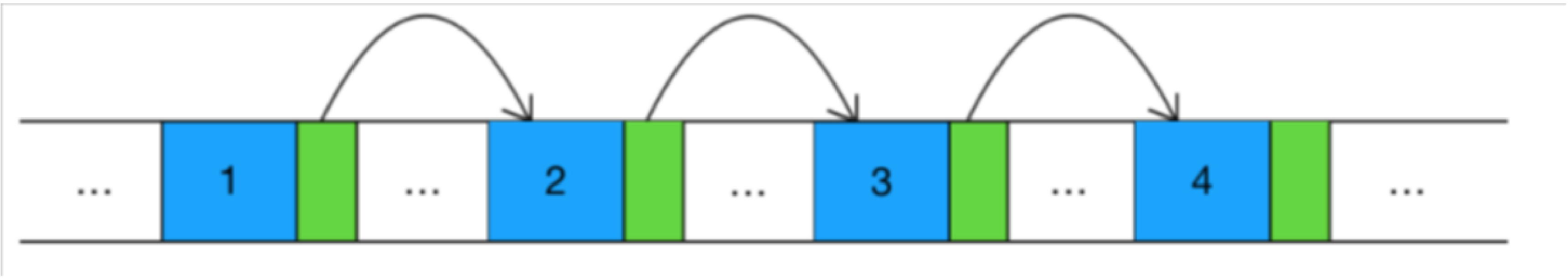
Linked List vs. Array

✔ dynamic, no need to deal with fixed memory size
✖ accessing speed

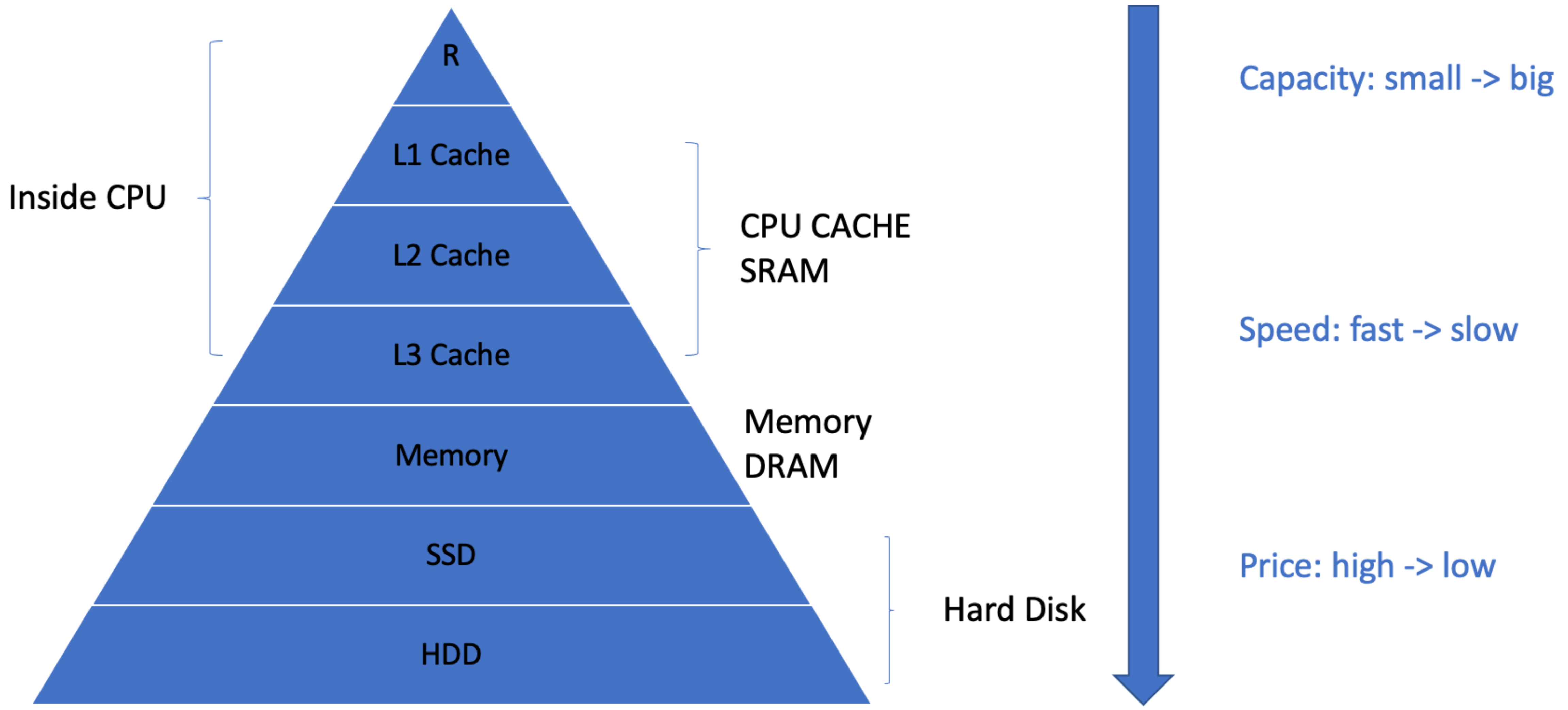
Array:

Memory Address	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	...
Computer's memory	H		E		L		L		O		...
Index Number	0		1		2		3		4		

Linked List:

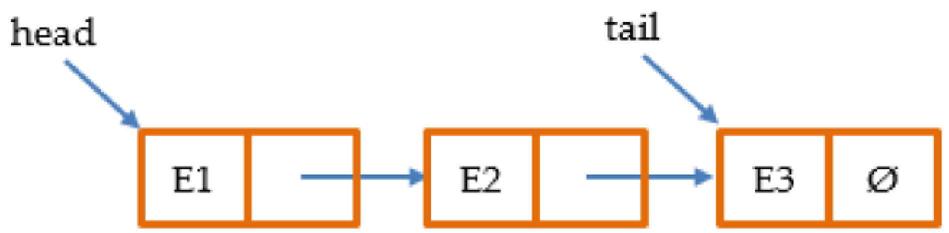


Linked List vs. Array



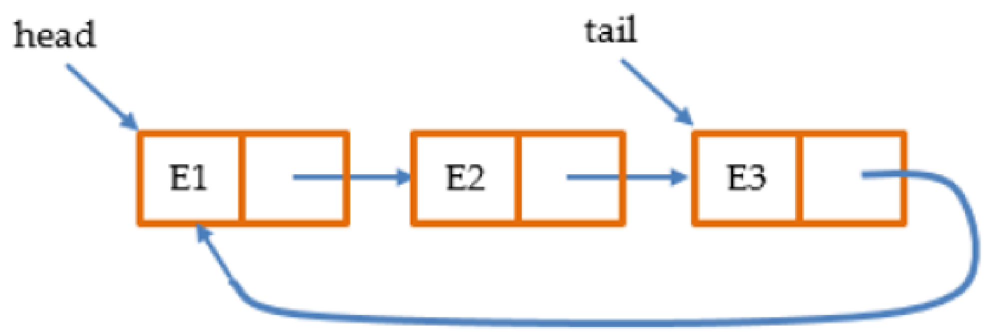
Linked List

Linked List

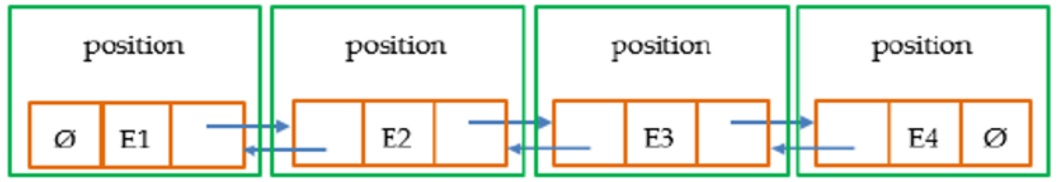


Doubly Linked List

Circular Linked List



Positional Linked List



Circular Linked List

a linked list where all nodes are connected to form a circle

- no null at the end
- can iterate from any node
- e.g. for cpu job list - OS putting running applications in a list and then to cycle through them by giviing each of them a slice of time to execute, and then making them wait, when it reaches the end of the list, it can cycle around to the front of the list

circular singly linked list, circular doubly linked list, sorted circular linked list

Circular Linked List: Operations (potential)

- append(element)
- delete(element)
- search(element)
- is_empty()
- __len__()

Positional Linked List

- use Position class instead of index
- remove iteration

Positional Linked List: Operations (potential)

- first()
- last()
- before(position)
- after(position)
- set(position, element)
- search(element)
- is_empty()
- __len__()
- add_first(element)
- add_last(element)
- add_before(position, element)
- add_after(position, element)

Recursion

Recursion

Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself

👉 Recursion by definition is a function that calls itself.

Recursion

- base case
- recursive case

Example:

Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, ...

- when n = 1, fib(1) = 0
- when n = 2, fib(2) = 1
- when n > 2, fib(n) = fib(n-1) + fib(n-2)

```
def fib_recursive(n):
    if (n == 1):
        return 0
    if (n == 2):
        return 1
    return fib_recursive(n-1) + fib_recursive(n-2)
```

base case

recursive case

Recursion vs. Iterative

> *Anything with a recursion can be done iteratively (loop)*

🤗 Intuitive/DRY, code readability

```
def fib_recursive(n):
    if (n == 1):
        return 0
    if (n == 2):
        return 1
    return fib_recursive(n-1) + fib_recursive(n-2)
```

base case

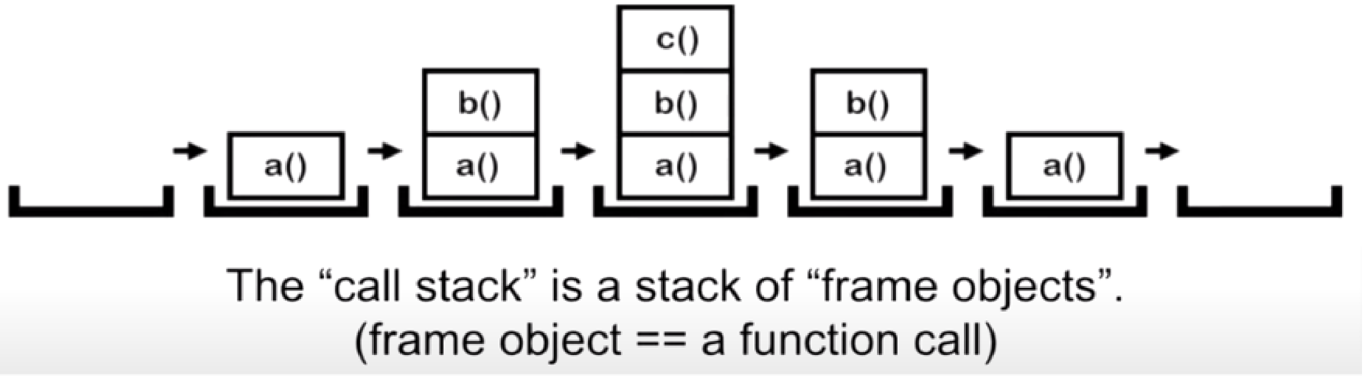
recursive case

🤔 Optimization, call stack

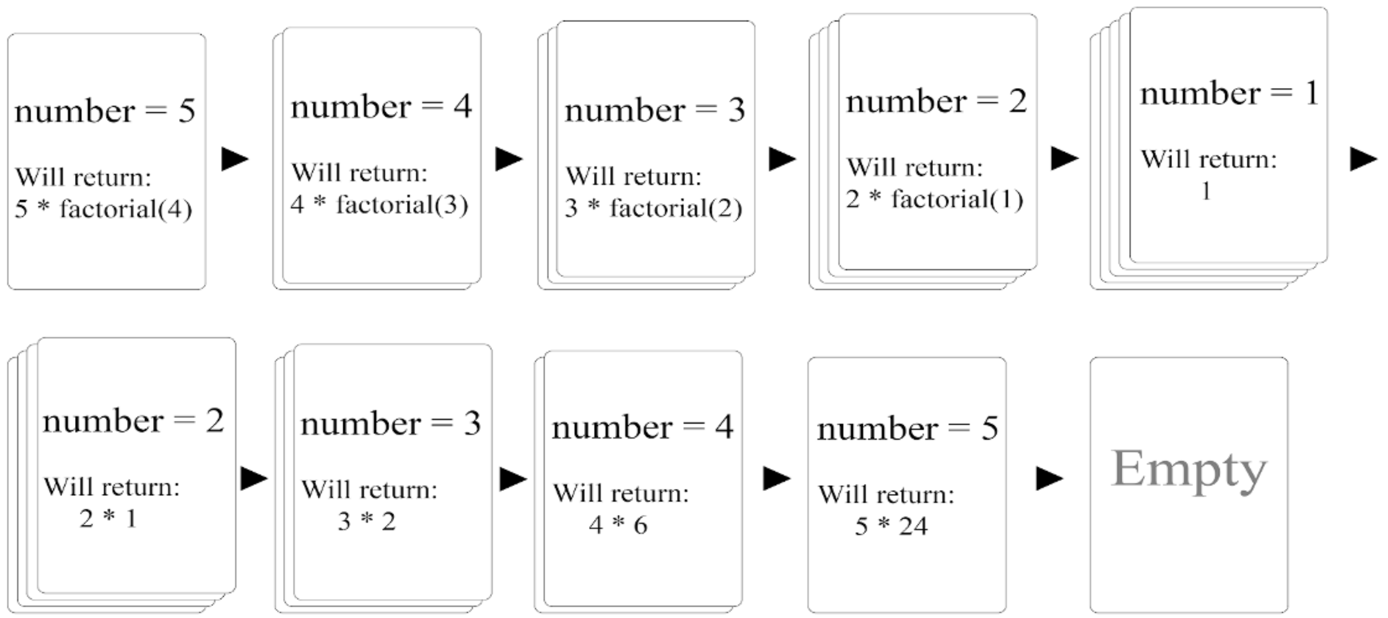
```
def fib_iterative(n):
    fib = [0,1]
    i = 2
    while (i < n): # index: 2, 3, ..., n-1
        fib.append(fib[i-1] + fib[i-2]) # fib[i]
        i += 1
    return fib[n-1] # the nth number
```

Recursive Call

Call Stack:



fib_recursive(5):



Recursive Call

- Max call stack size (stack overflow error)
- Tail Call Optimization
- Memorization

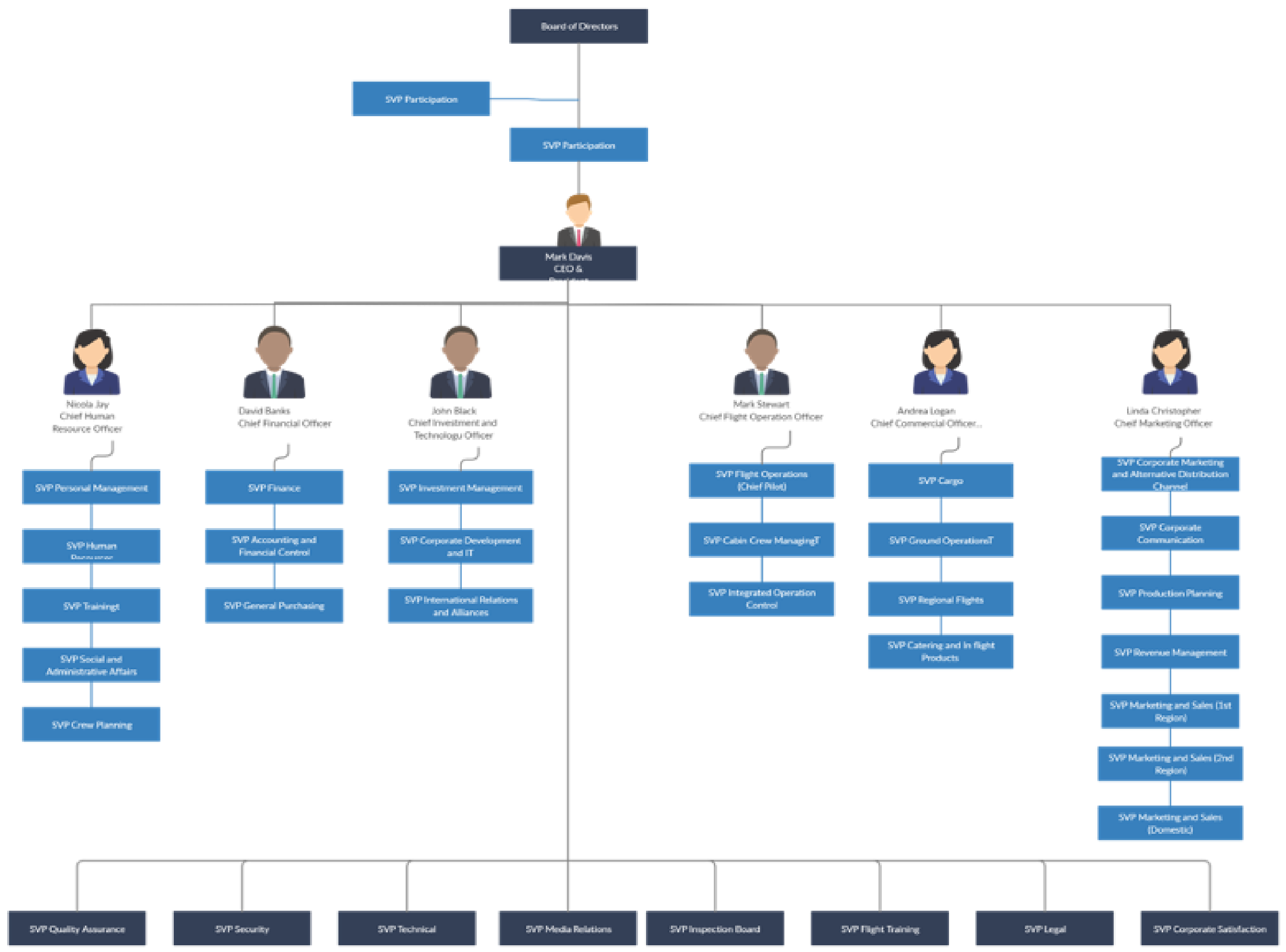
Recursive Call

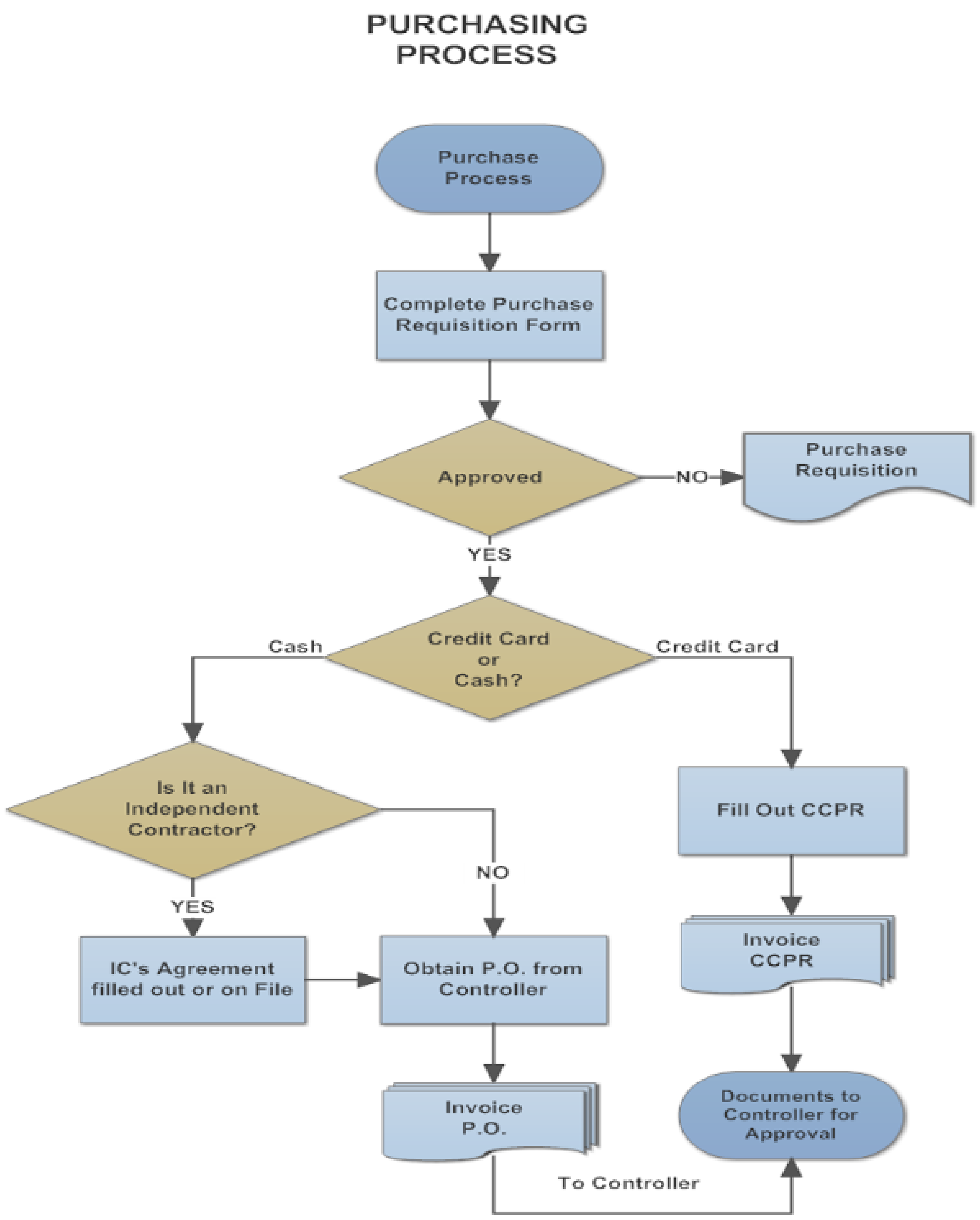
Fundamental technique to solve problem:

- Identifying the base case
- Identifying the recursion formula/equation to transform the problem to smaller version
    - Problem requires back-tracking
    - Problem has tree structure

Tree

Tree

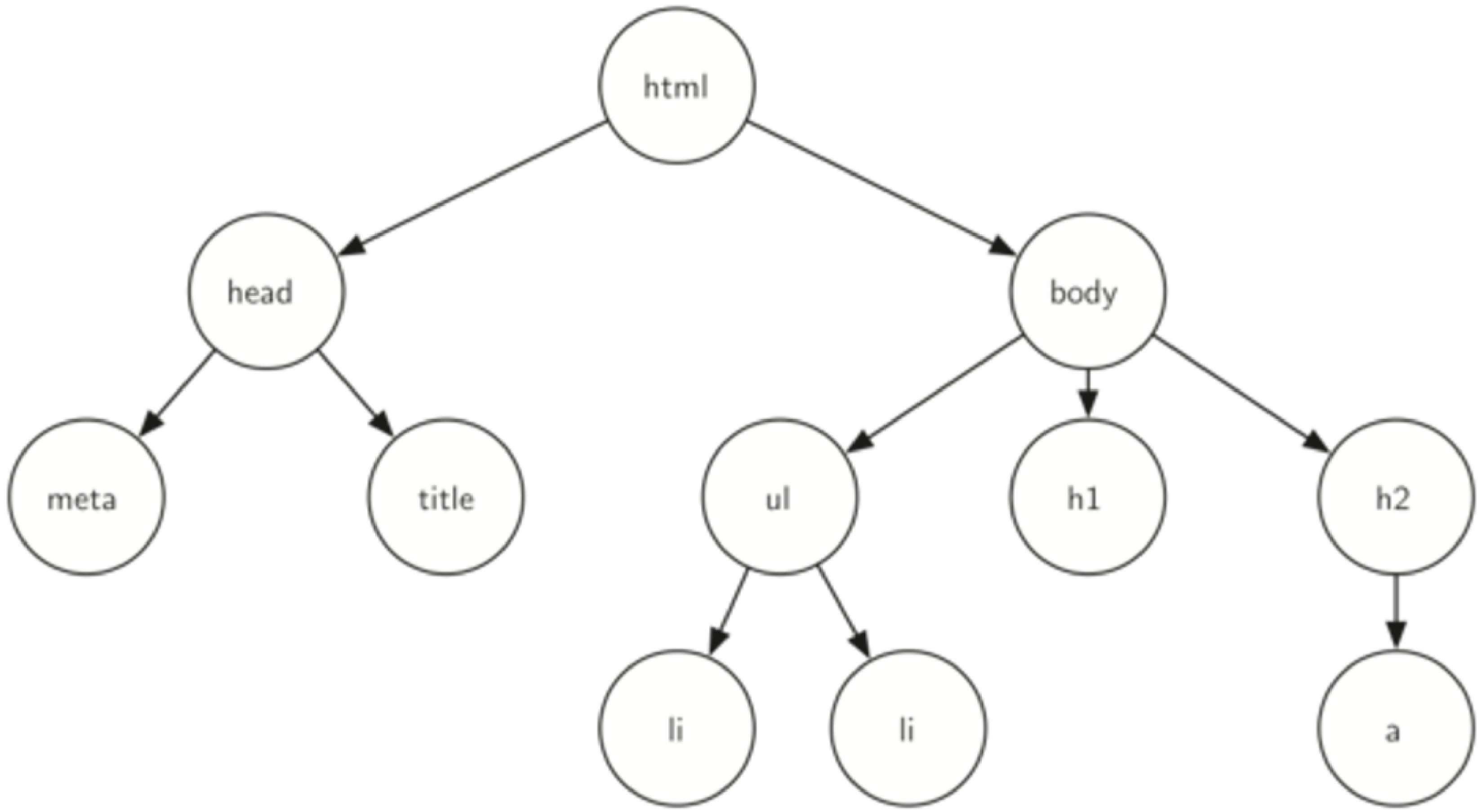

Board of Directors
SVP Participation
SVP Participation
Mark Davis
CEO &
Nicola Jay
Chief Human
Resource Officer
SVP Personal Management
SVP Human
SVP Trainingt
SVP Social and
Administrative Affairs
SVP Crew Planning
David Banks
Chief Financial Officer
SVP Finance
SVP Accounting and
Financial Control
SVP General Purchasing
John Black
Chief Investment and
Technologu Officer
SVP Investment Management
SVP Corporate Development
and IT
SVP International Relations
and Alliances
Mark Stewart
Chief Flight Operation Officer
SVP Flight Operations
(Chief Pilot)
SVP Cabin Crew ManagingT
SVP Integrated Operation
Control
Andrea Logan
Chief Commercial Officer...
SVP Cargo
SVP Ground OperationsT
SVP Regional Flights
SVP Catering and In flight
Products
Linda Christopher
Cheif Marketing Officer
SVP Corporate Marketing
and Alternative Distribution
Channel
SVP Corporate
Communication
SVP Production Planning
SVP Revenue Management
SVP Marketing and Sales (1st
Region)
SVP Marketing and Sales (2nd
Region)
SVP Marketing and Sales
(Domestic)
SVP Quality Assurance
SVP Security
SVP Technical
SVP Media Relations
SVP Inspection Board
SVP Flight Training
SVP Legal
SVP Corporate Satisfaction


PURCHASING
PROCESS
Purchase
Process
Complete Purchase
Requisition Form
Approved
NO
Purchase
Requisition
YES
Credit Card
or
Cash?
Cash
Credit Card
Is It an
Independent
Contractor?
NO
YES
IC's Agreement
filled out or on File
Obtain P.O. from
Controller
Invoice
P.O.
Fill Out CCPR
Invoice
CCPR
Documents to
Controller for
Approval
To Controller

Tree

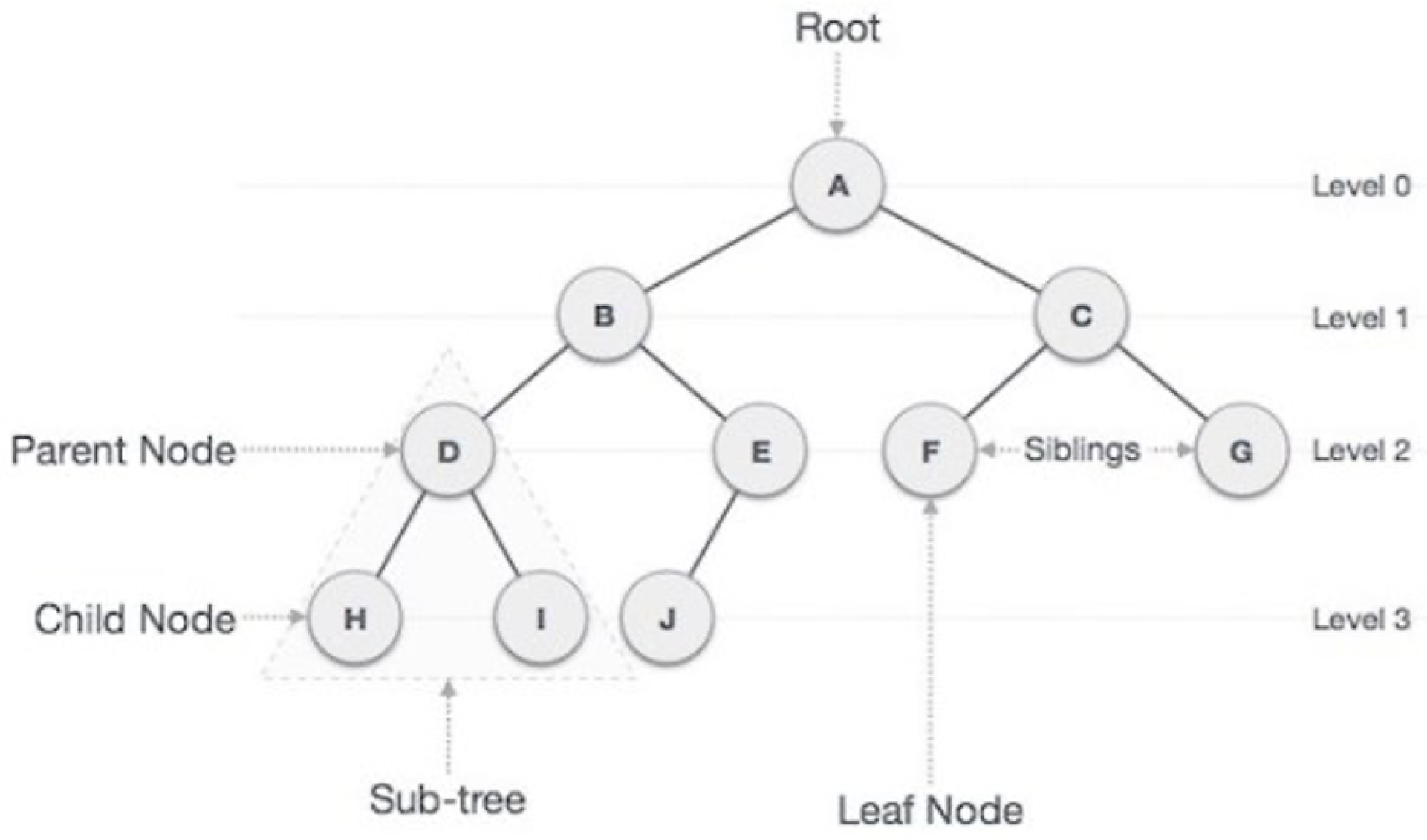
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
    <li>List item one</li>
    <li>List item two</li>
</ul>
<h2><a href="http://www.suss.edu.sg">SUSS</a><h2>
</body>
</html>
```

HTML DOM tree



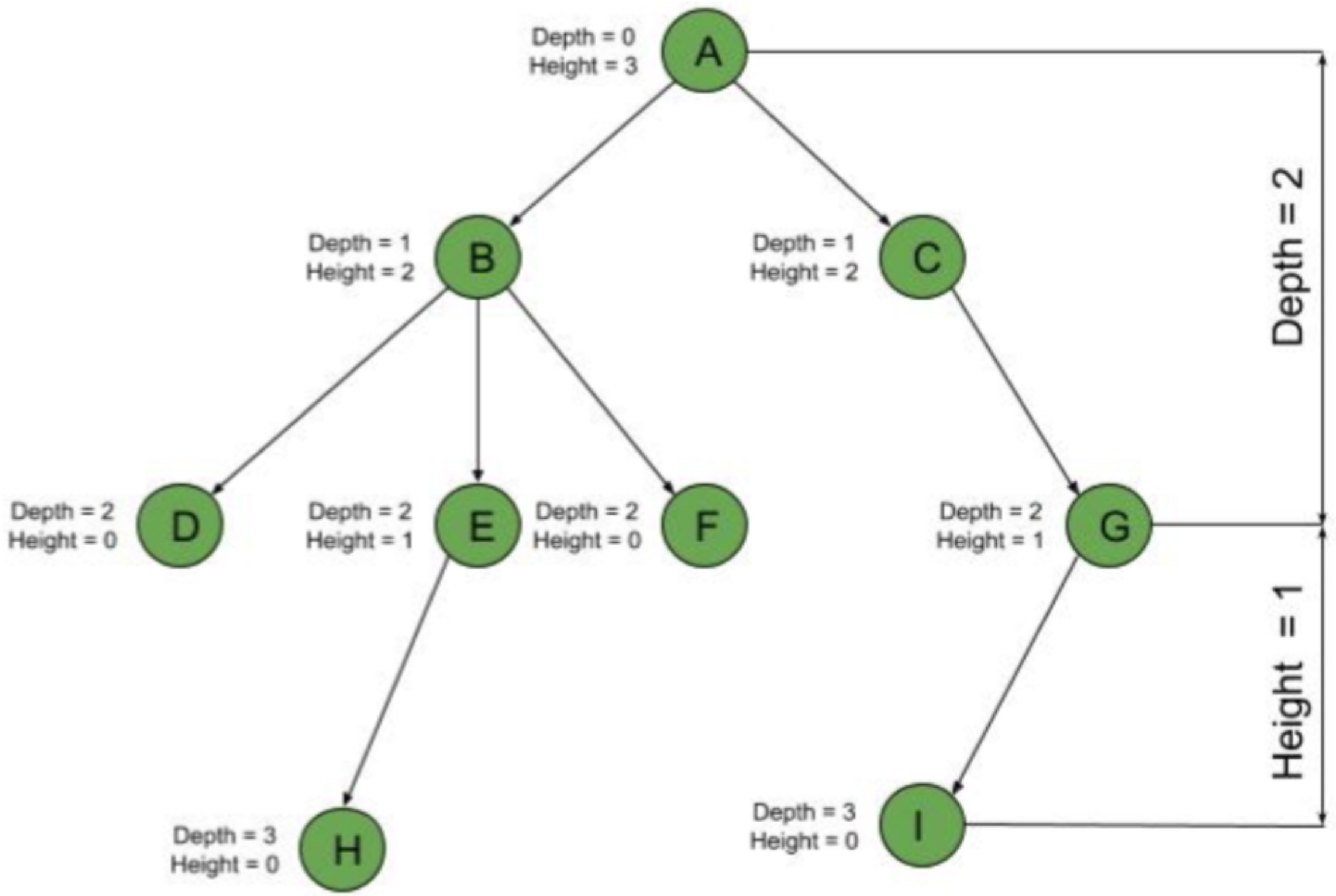
Tree Terminology

- Node: Root, Leaf, Internal Node
- Node: Parent, Children, Sibling
- Edge
- Degree: no. of outgoing edges or Children
- SubTree
- Path: A → B → D → H
- Level: no. edges in path from root to the node
- Depth: no. edges in path from the node to the root
- Height: no. edges in longest path from the node to the leaf



Tree Terminology

- **Level**
- **Depth**
- **Height**

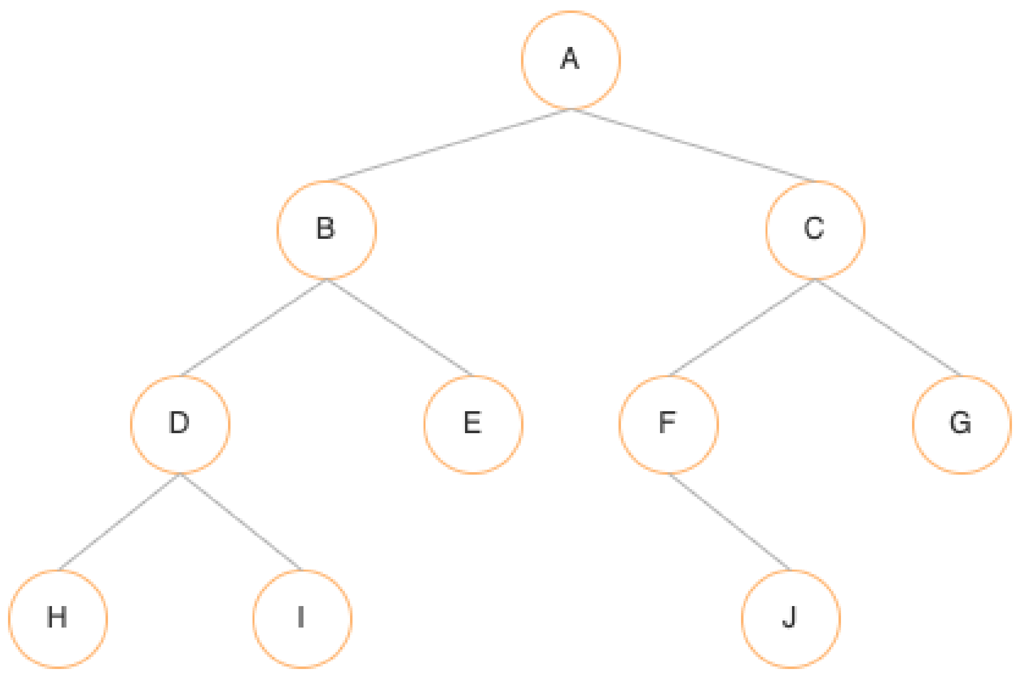


Binary Tree

- One root
- Max 2 child nodes
- One and only one path from root to each node (path)
- Max nodes on level: 2^l
- Max nodes total: $2^{h+1} - 1$

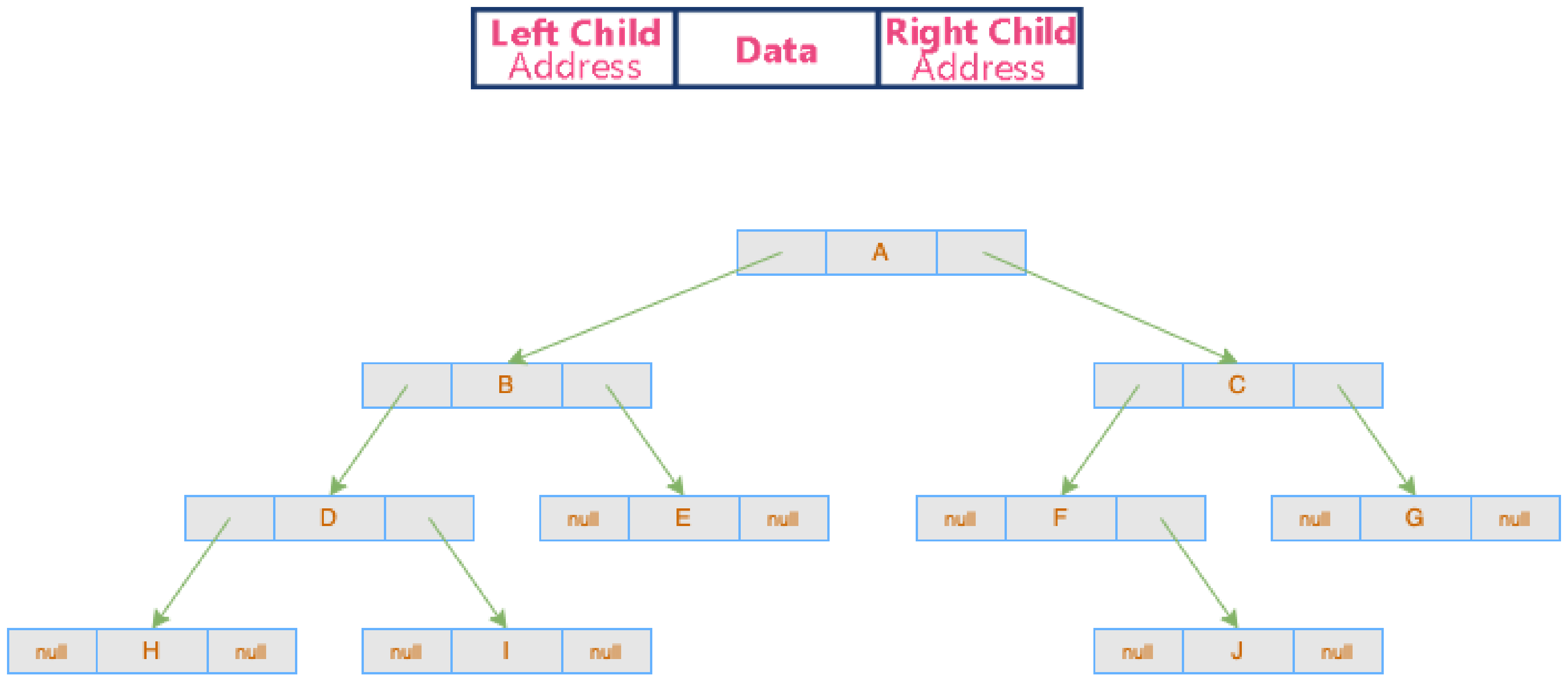
Binary Tree

Array



['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', '#', '#', '#', '#', '#', 'J']

Left/Right Linked List

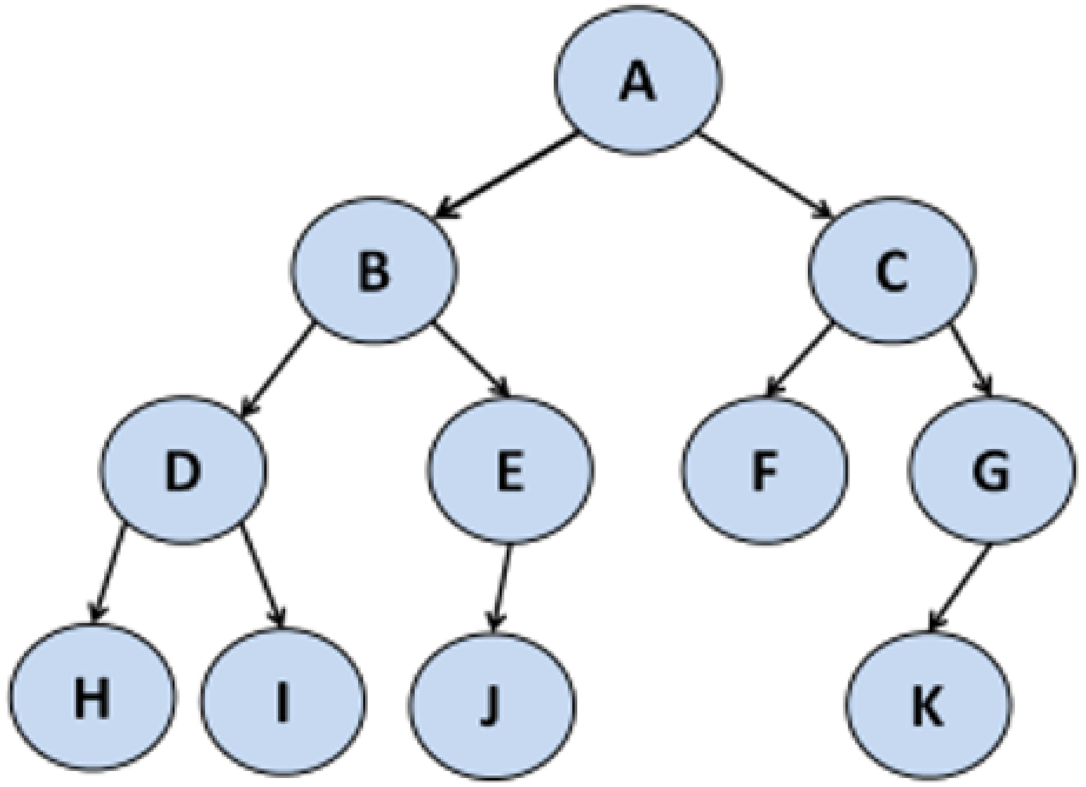


Binary Tree Traverse (DFS): pre-order

ROOT → Left → Right

1. Visit the root
2. Traverse the left subtree
3. Traverse the right subtree

A B D H I E J C F G K

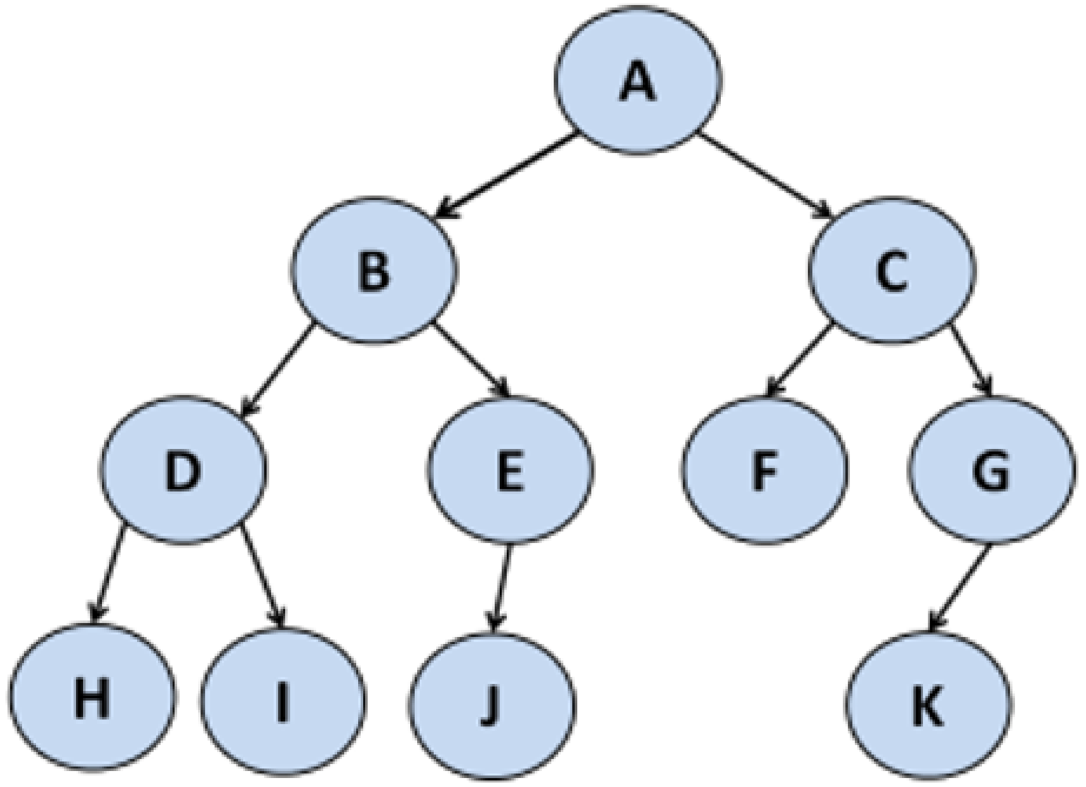


Binary Tree Traverse (DFS): in-order

Left → Root → Right

1. Traverse the left subtree
2. Visit the root
3. Traverse the right subtree

H D I B J E A F C K G

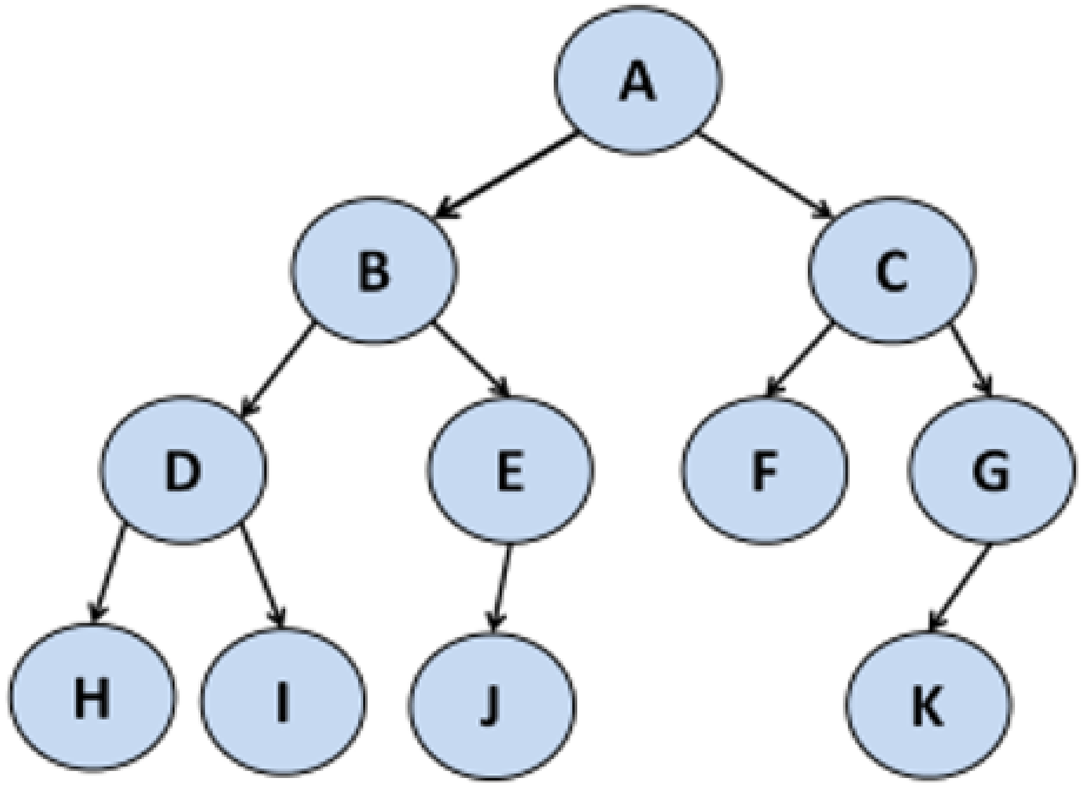


Binary Tree Traverse (DFS): post-order

Left → Right → Root

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root

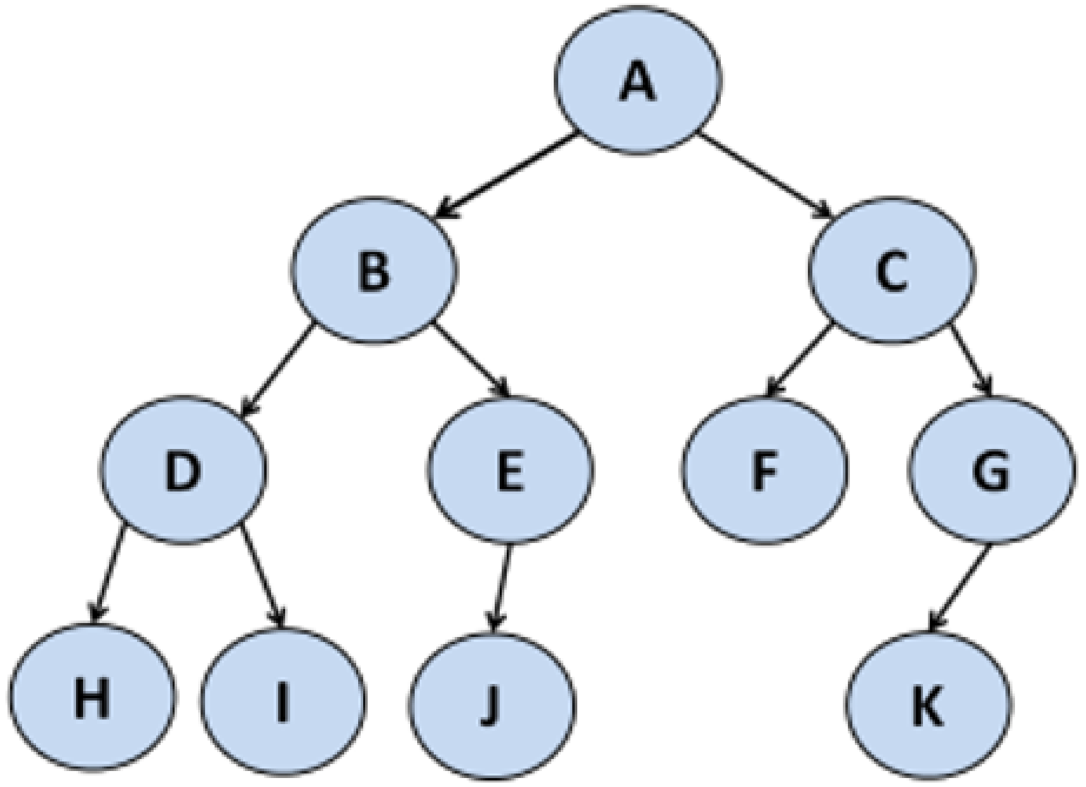
H I D J E B F K G C A



Binary Tree Traverse (BFS): level order

1. Visit the root
2. Visit the left node
3. Visit the right node
4. Go to next level

A B C D E F G H I J K

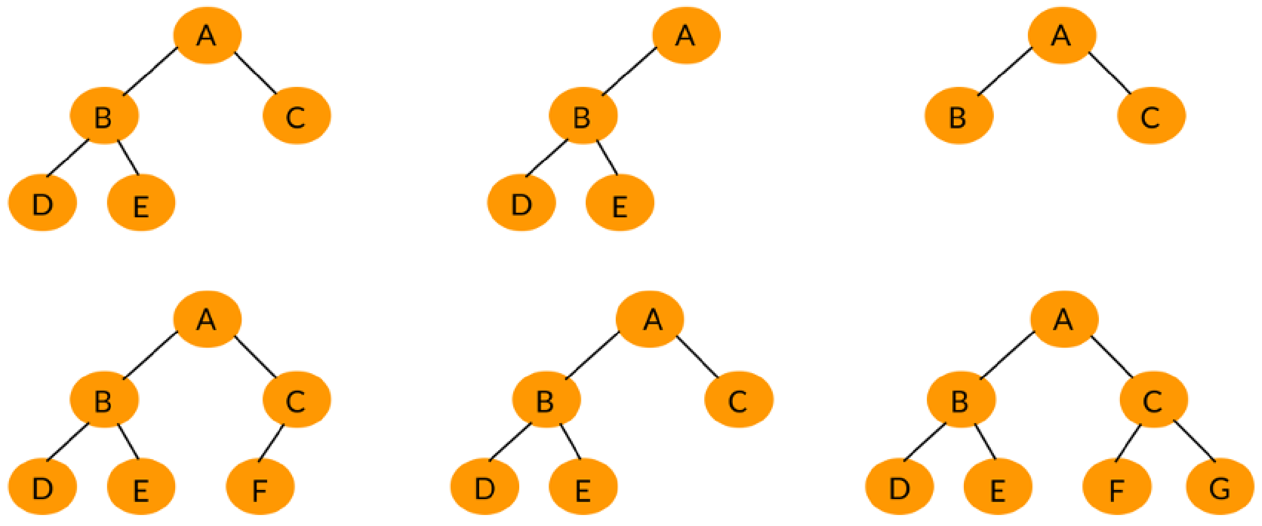


Binary Tree

- **Complete Binary Tree** : every level is completely filled except the last (leaf) and all nodes are as far left as possible

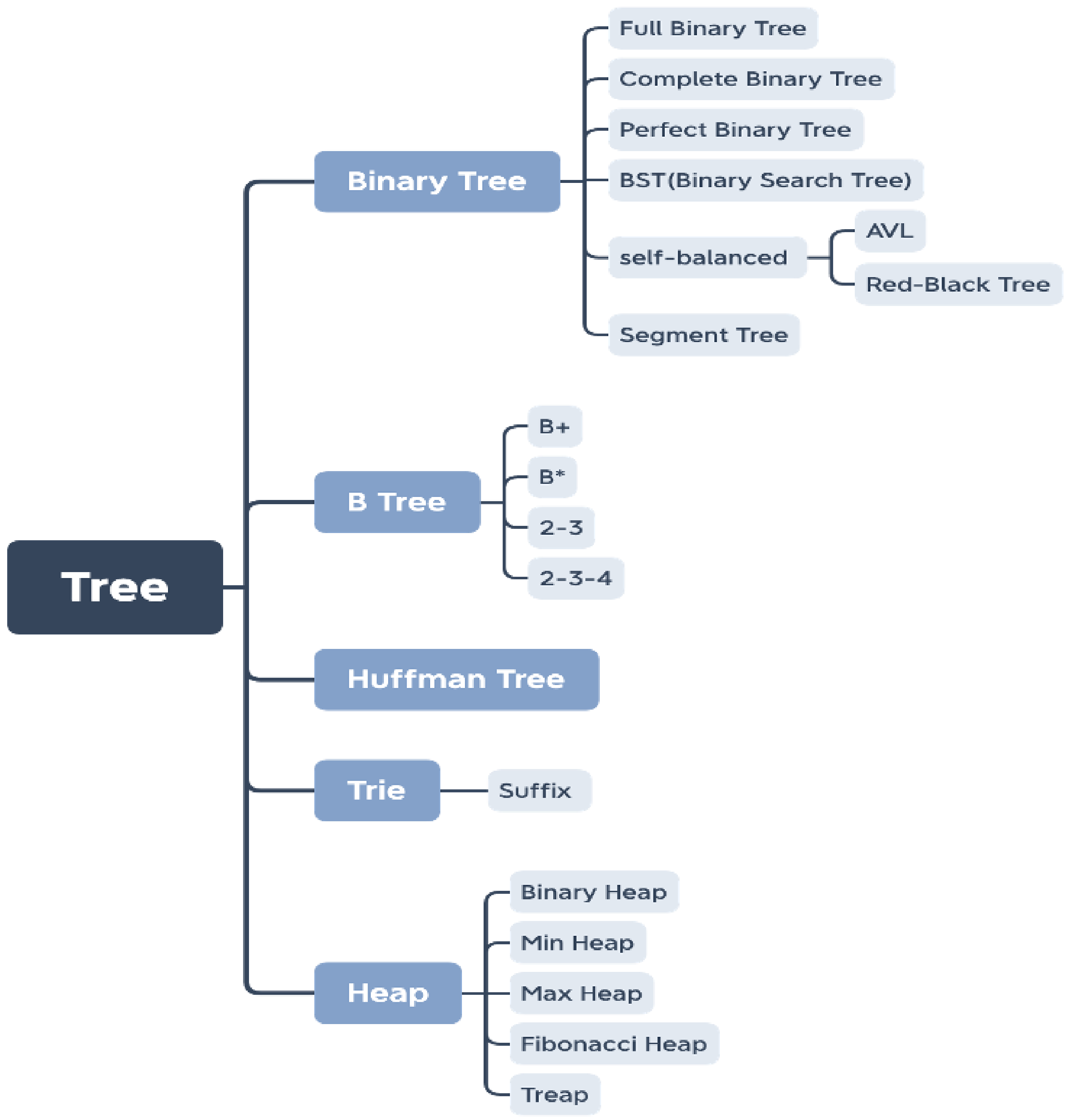
- **Full Binary Tree** : every node has two child nodes except leaf

- **Perfect Binary Tree** : every node has two child nodes except leaf and all leaves on same level



Complete Binary Tree
Full Binary Tree
Perfect Binary Tree

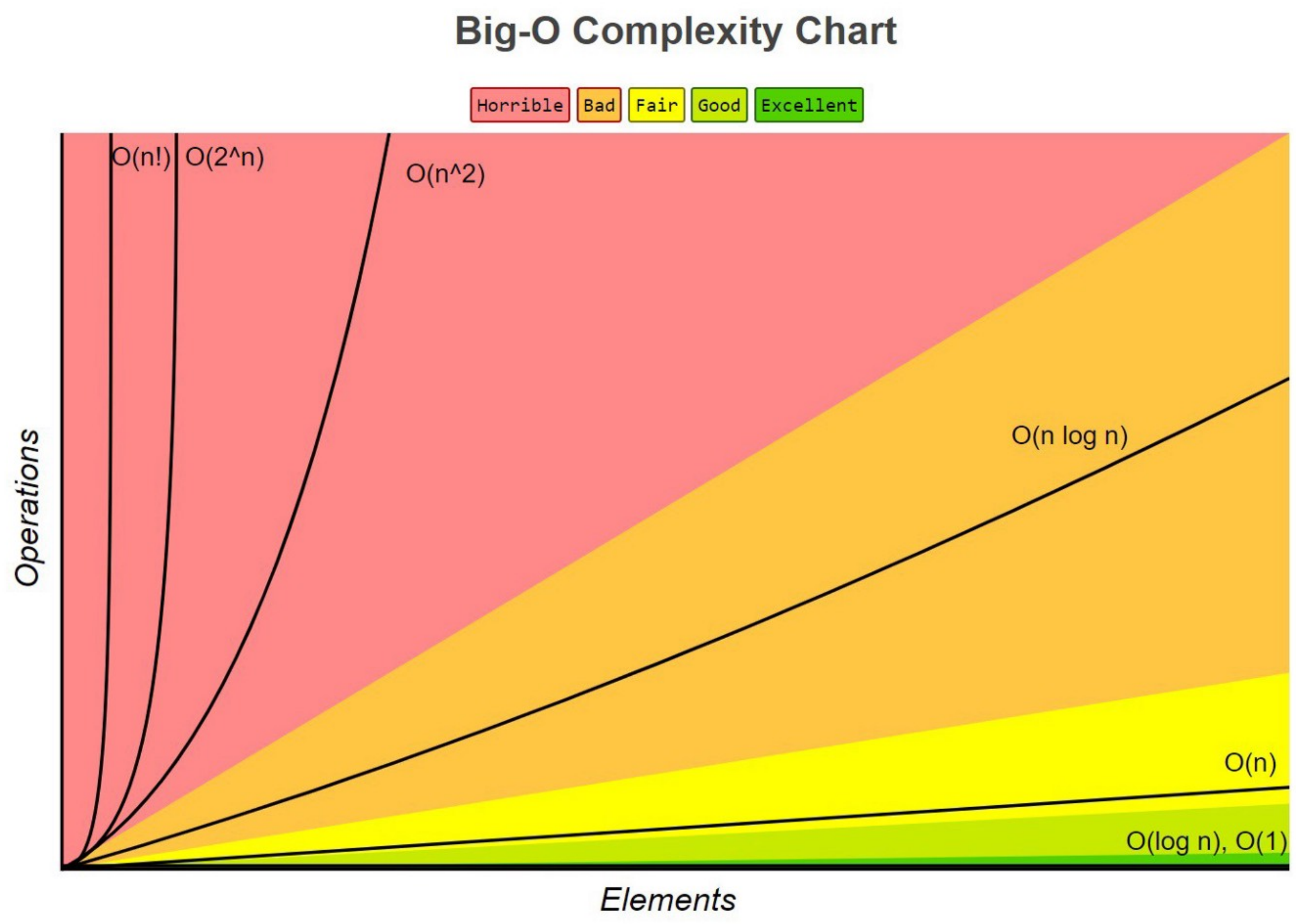
Trees


Tree
Binary Tree
Full Binary Tree
Complete Binary Tree
Perfect Binary Tree
BST(Binary Search Tree)
self-balanced
AVL
Red-Black Tree
Segment Tree
B Tree
B+
B*
2-3
2-3-4
Huffman Tree
Trie
Suffix
Heap
Binary Heap
Min Heap
Max Heap
Fibonacci Heap
Treap

Summary

Big-O

😊 $O(1) < O(\log_* n) < O(n) < O(n \log_* n) < O(n^2) < O(2^n) < O(n!)$ 🥵



https://www.bigocheatsheet.com/

Array

Concept	▪ consecutive memory space: arr[i].address = base_address + i * data_type_size ▪ same data type → same size for each element ▪ fixed length
Complexity	**Accessing O(1)**, Searching O(n)
Notes	▪ not memory friendly ▪ cpu cacheable ▪ index from 0 ▪ fundamental data structure to implement others such as stack, queue, heap ▪ data type (programming language) vs. data structure
Hands-on	dynamic array, stack/queue, binary search, etc.

Stack

Concept	**LIFO/FILO**
Complexity	Accessing O(n), Searching O(n), Inserting/push O(1), Deleting/pop O(1)
Notes	Stack implementation by dynamic array or linked list
Hands-on	function call stack, expression matching, etc.

Queue

Concept	**FIFO/LILO**
Complexity	Accessing O(n), Searching O(n), Inserting/enqueue O(1), Deleting/dequeue O(1)
Notes	Queue implementation by dynamic array or linked list
Hands-on	priority queue, circular queue, job queue, resource pool, etc.

Linked List

Concept	▪ nonconsecutive memory space ▪ node: data + pointer ▪ Single Linked List, Doubly Linked List, Circular Linked List, Positional Linked List
Complexity	Accessing O(n), Searching O(n), **Inserting O(1)**, **Deleting O(1)**
Notes	▪ accessing slower than array ▪ with/without head/tail node (which don’t store any data) ▪ fundamental data structure to implement others such as skip list, hash table, etc.
Hands-on	stack, queue, traverse/reverse/update/merge, etc.

Binary Tree

Concept	▪ one root ▪ max 2 child nodes ▪ height & depth ▪ 4 traversal (DFS/BFS): in-order(left-root-right), pre-order(root-left-right), post-order(left-right-root), level-order ▪ proper, perfect, full, complete binary tree
Complexity	▪ DFS: time O(n), space O(h) ▪ BFS: time O(n), space O(n)
Notes	stored in array or linked nodes
Hands-on	4 traversal

Algorithm Design & Pattens

Linear Search

Linear & Binary Search

Complexity: $O(n)$

Binary Search

- Input: array, target element
- Output: position (-1 if not existing)

As we know, whenever we are given a sorted Array or LinkedList or Matrix, and we are asked to find a certain element, the best algorithm we can use is the

- Shrink the search space every iteration (recursion)

- Cannot exclude potential answers during each shrinking

> Although the basic idea of binary search is comparatively straightforward, the details can be surprisingly tricky
> — Donald Knuth

Binary Search

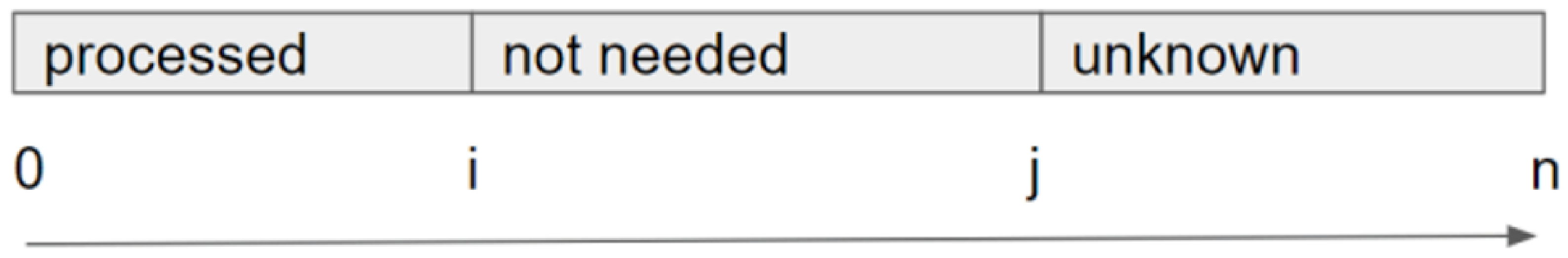
- Contains
- First occurrence of a key
- Last occurrence of a key
- Least element greater than
- Greatest element less than
- Closest element

Further reading: Generalized Binary Search with predicates and main theorem:
https://www.topcoder.com/community/competitive-programming/tutorials/binary-search

Two Pointers

In problems where we deal with sorted arrays (or LinkedLists) and need to find a set of elements that fulfill certain constraints, the **Two Pointers** approach becomes quite useful. The set of elements could be a pair, a triplet or even a subarray.

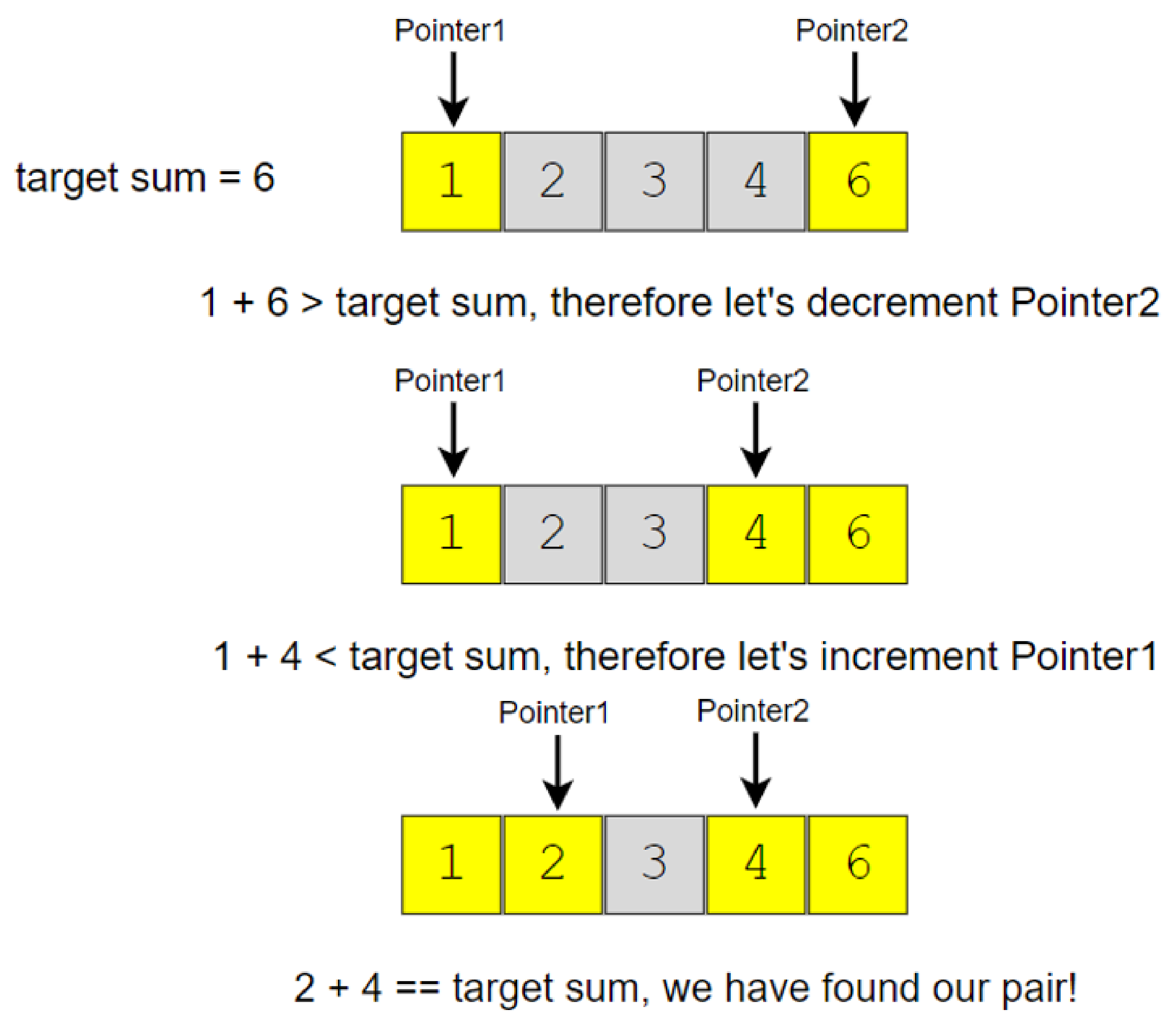
- same direction



- reverse direction

two sum

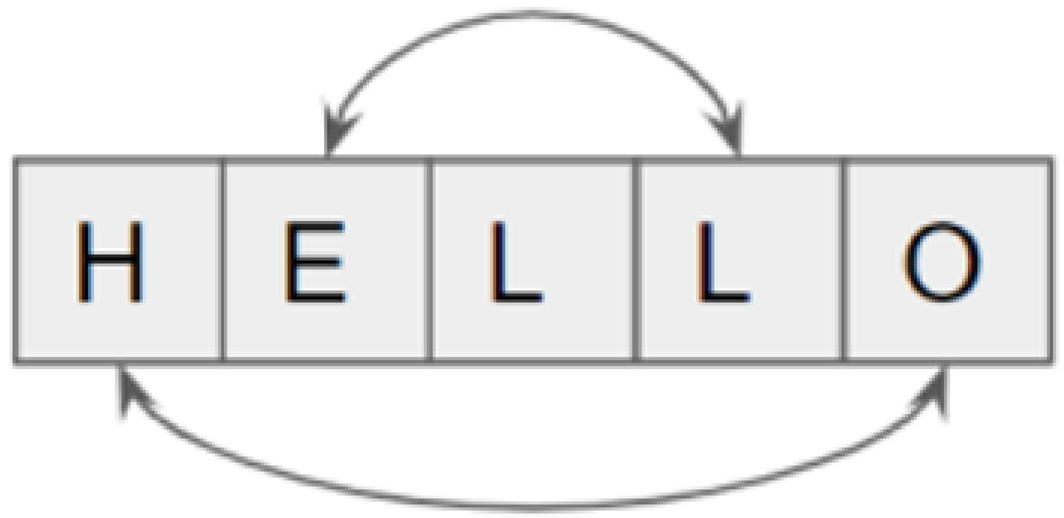
Given an array of sorted numbers and a target sum, find a pair in the array whose sum is equal to the given target.



reverse a string

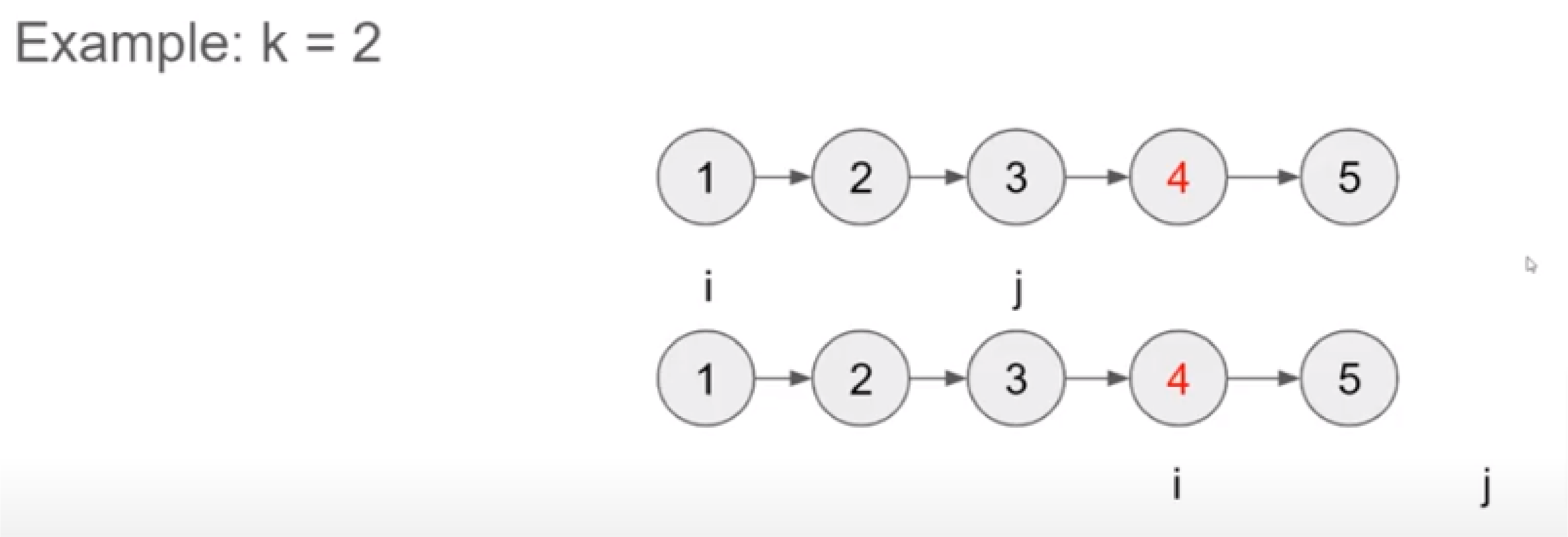
Write a function that reverses a string. The input string is given as an array of characters char[]. Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.



k’th node from the end of linked list

Given the head of a Singly LinkedList, Find k'th node from the end of a linked list.



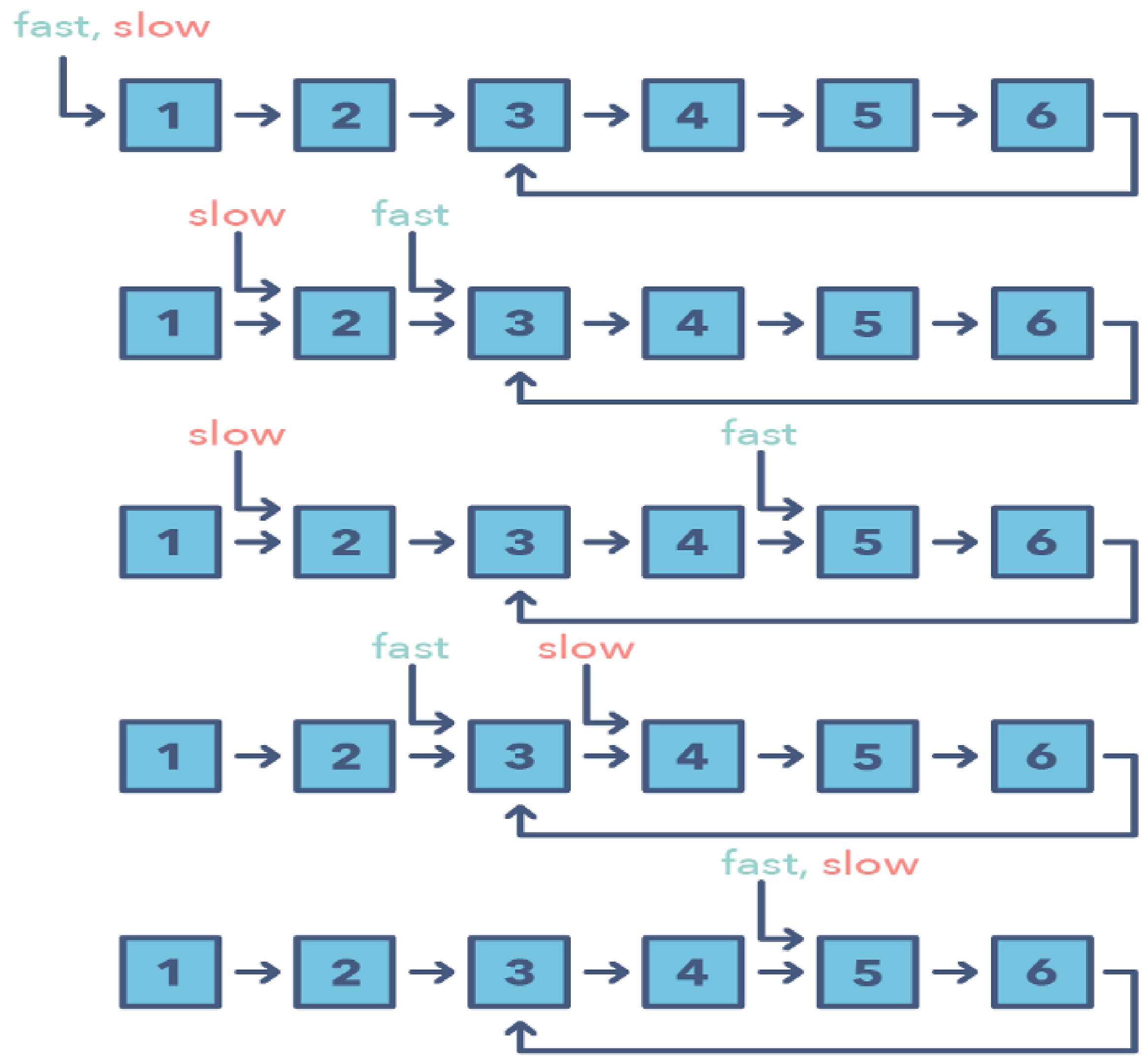
Fast & Slow Pointers

The **Fast & Slow pointer** approach, also known as the Hare & Tortoise algorithm, is a pointer algorithm that uses two pointers which move through the array (or sequence/LinkedList) at different speeds.

- Fast & Slow Pointers, distance btw them
- Speed

LinkedList has a cycle

Given the head of a Singly LinkedList, write a function to determine if the LinkedList has a cycle in it or not.

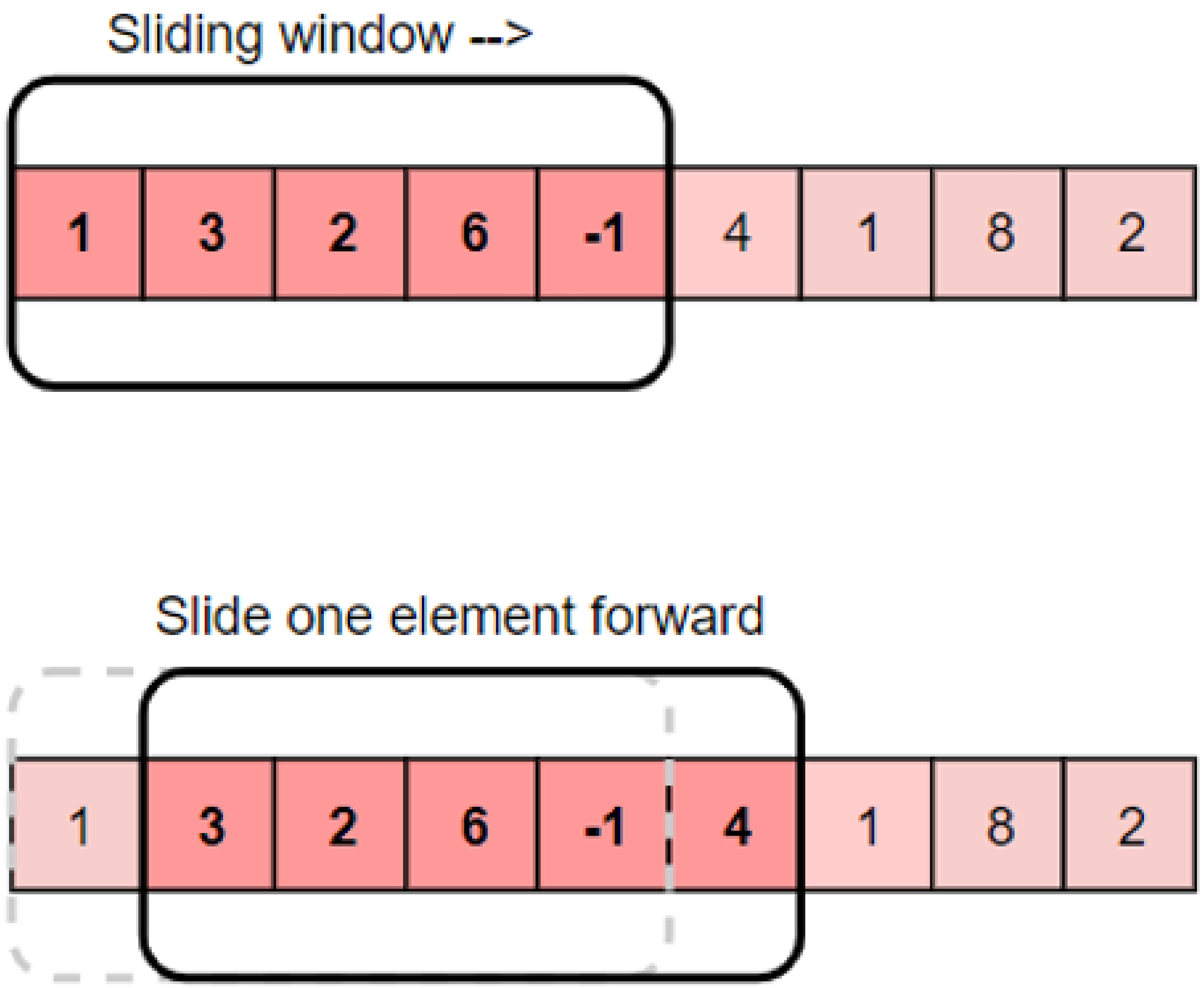


Sliding Window

In many problems dealing with an array (or a LinkedList), we are asked to find or calculate something among all the contiguous subarrays (or sublists) of a given size.

the average

Given an array, find the average of all contiguous subarrays of size ‘K’ in it.



longest substring

Given a string, find the length of the longest substring which has no repeating characters.

Example1 Input: String="aabccbb“ Output: 3

Explanation: The longest substring without any repeating characters is "abc".

Example2 Input: String="abbbb“ Output: 2

Explanation: The longest substring without any repeating characters is "ab".

Example3 Input: String="abccde“ Output: 3

Explanation: Longest substrings without any repeating characters are "abc" & "cde".

SINGAPORE UNIVERSITY OF SOCIAL SCIENCES

LAB

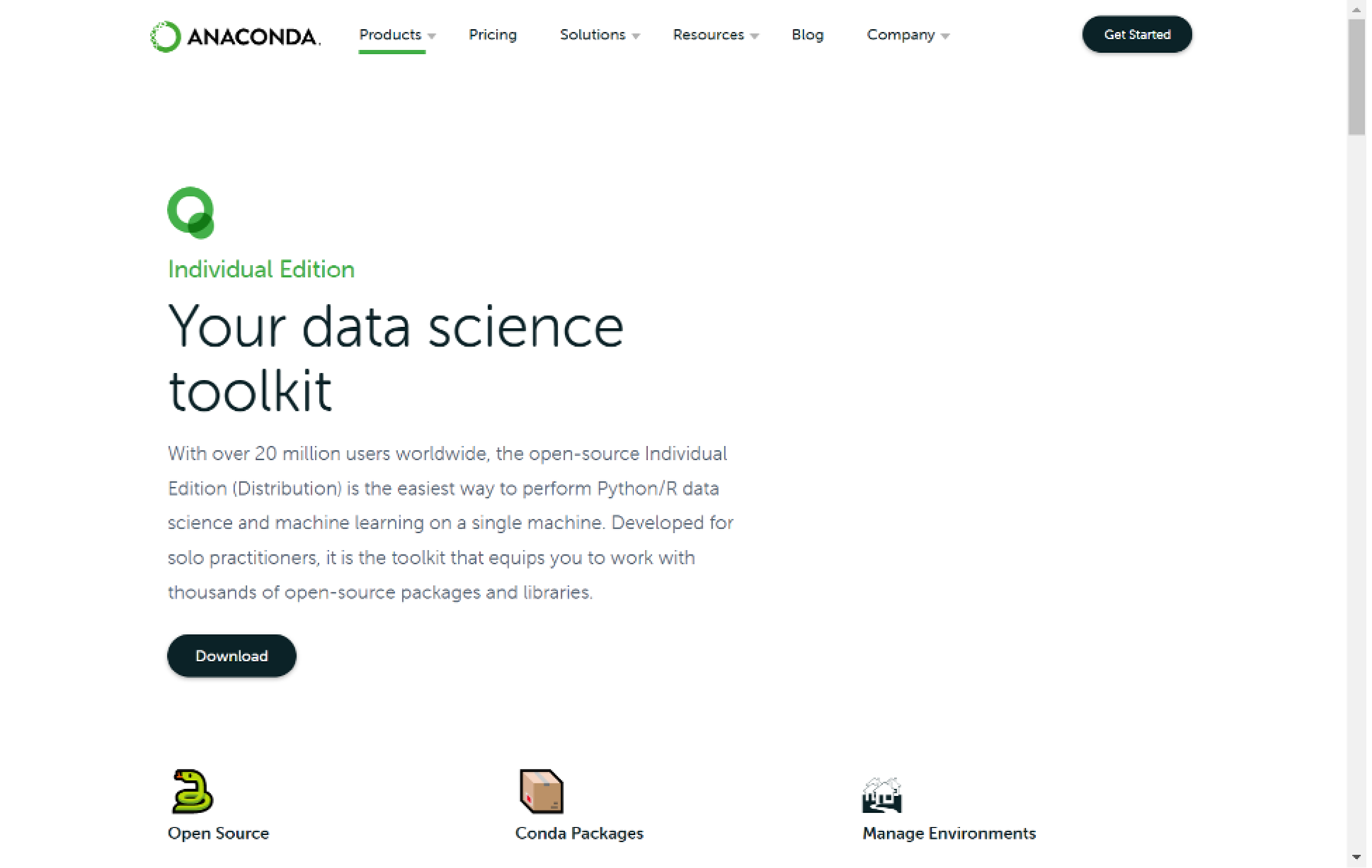
Lab 1

- download and install Anaconda
- create and Activate your Anaconda Python env
- (Optional) install and setup VS Code
- familiar yourself with Python and do exercise 📝 lab1.ipynb

Lab 1

Download and install Anaconda

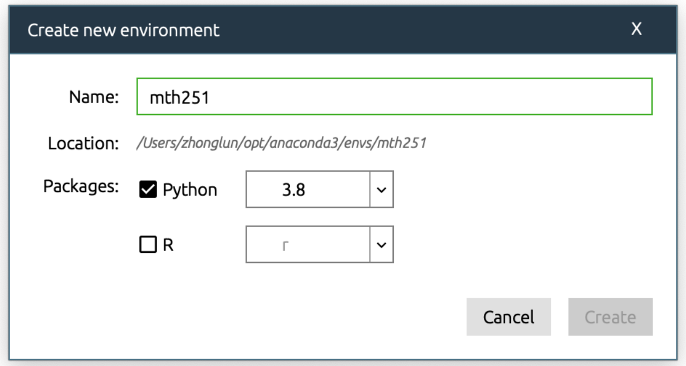
https://www.anaconda.com/products/individual

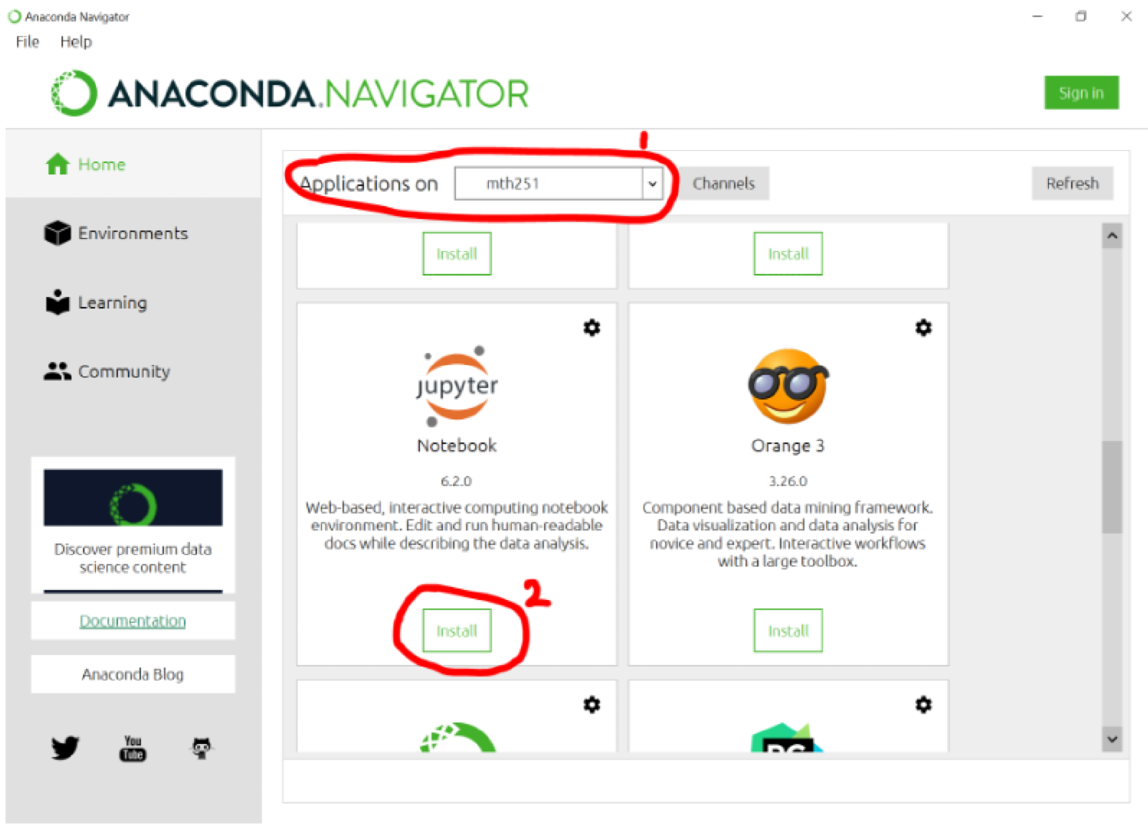


Lab 1

Create and Activate your Anaconda Python env

1. Launch Conda Navigator
2. Environments → +Create to create a new env (**mth251**)
3. Switch to your Python env **mth251** from "Application on"
4. Install Jupyter Notebook
5. After that, Launch Jupyter Notebook





Lab 1

Create and Activate your Anaconda Python env

💬 Also possible to perform via command line:

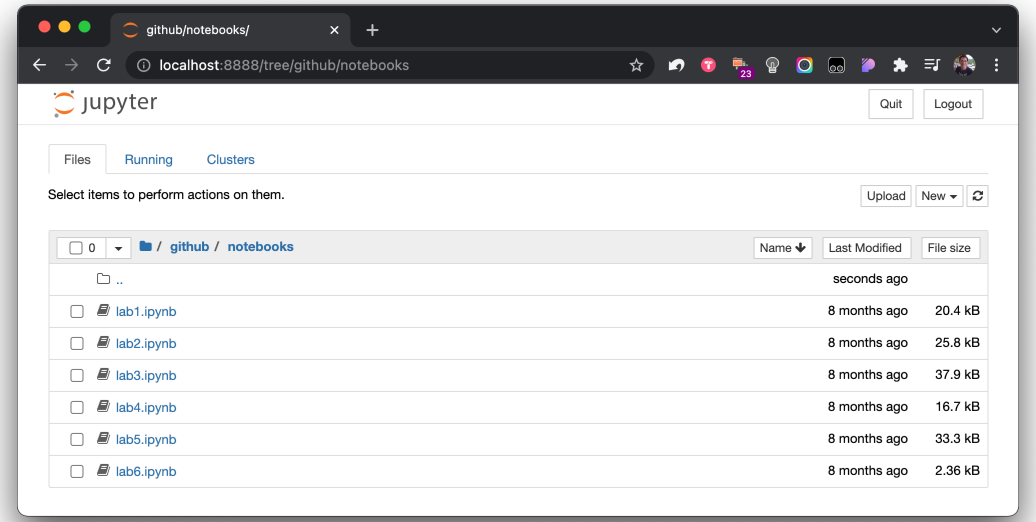
```
> # create the env
> conda create -n mth251 python=3.8
> # activate the env
> conda activate mth251
> # install jupyter
> conda install -c conda-forge notebook
> # multipledispatch for lab1
> conda install -c anaconda multipledispatch
> # start jupyter notebook
> jupyter notebook
```

- copy & paste the Jupyter link in the prompt to your browser
- save notebooks to your local
- Control-C to stop Jupyter from the command line

Lab 1

Create and Activate your Anaconda Python env

6. Now you are ready to create, edit and run Jupyter notebooks (lab1.ipynb):



Lab 1

VS Code

> VS Code now fully integrated with Jupyter notebook, refer to this link:
> Jupyter Notebooks in VS Code

Google Colab

> Google provides online Jupyter env:
> https://colab.research.google.com/

notebooks: https://github.com/fastzhong/mth251/tree/main/public/notebooks

colab: https://colab.research.google.com/github/fastzhong/mth251/blob/main/public/notebooks/lab1.ipynb

Lab 1

lab1.ipynb

- Python

  1. Data Type & Operators
  2. Collections
  3. Program Structure
  4. OO & Class

- Big O

Lab 2

- review Array, Stack, Queue, Recursion
- exercise 📝 lab2.ipynb
- priority queue
- circular queue

Lab 2

Exercise: two sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example 1

Input: *nums = [2,7,11,15], target = 9*

Output: *[0,1]*

Because nums[0] + nums[1] == 9, we return [0, 1]

Example 2

Input: *nums = [3,2,4], target = 6*

Output: *[1,2]*

Example 3

Input: *nums = [3,3], target = 6*

Output: *[0,1]*

Lab 2

Exercise: valid parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Example 1

Input: s = "()[]{}" Output: true

Example 2

Input: s = "(]" Output: false

Example 3

Input: s = "([)]" Output: false

Example 4

Input: s = "{[]}" Output: true

Lab 2

`Priority Queue` is similar to queue but the element with higher priority can be moved forward to the front. Use exiting Queue class to implement a priority queue (element with lower value has higher priority).

🗒 Priority Queue can be used in Printer Jobs or Schedule Tasks.

Lab 2

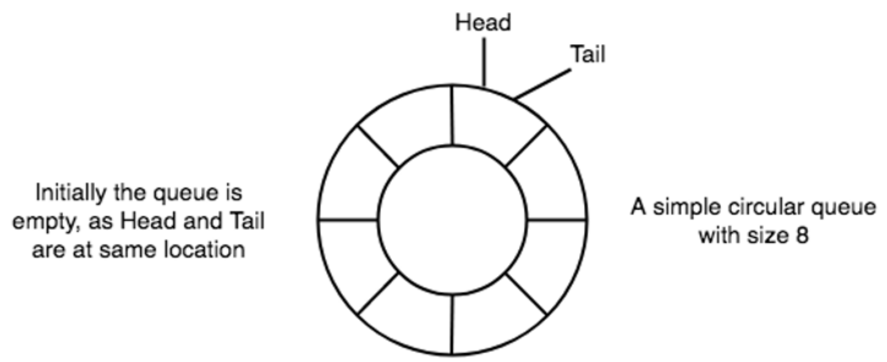
`Circular Queue` is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

Design your implementation of circular queue.

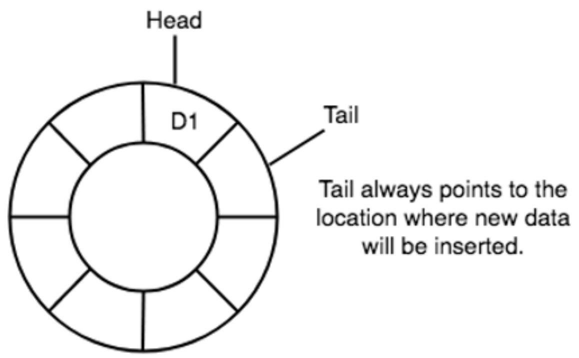
Lab 2

Circular Queue

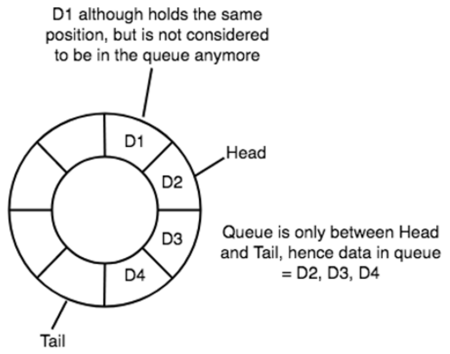
1. init



2. enqueue D1



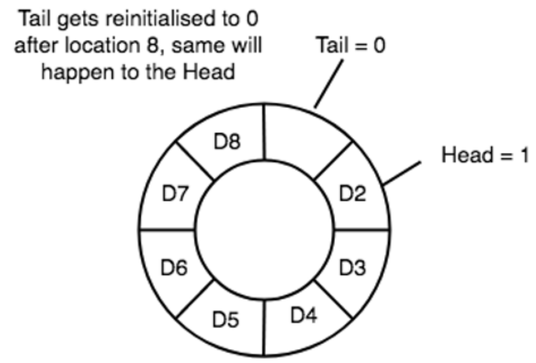
3. enqueue D2, D3, D4 and dequeue D1



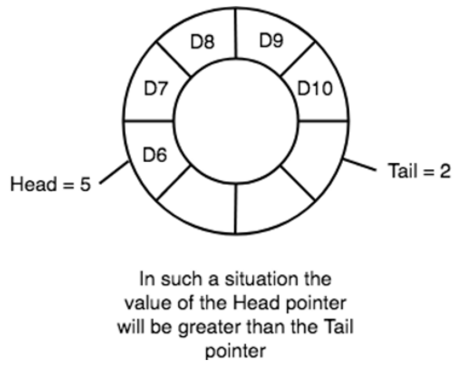
Lab 2

Circular Queue

4. enqueue D5, D6, D7, D8



5. dequeue D2, D3, D4, D5 and enqueue D9, D10



Lab 2

Circular Queue

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue (and it does not prevent the program accidentally creates a large queue or stack and use up the memory).

Implementation of CircularQueue class:

- **enqueue**(): insert the element
- **dequeue**(): delete the element
- **front**(): return the first element in the queue, if queue is empty, return None
- **rear**(): return the last element in the queue, if queue is empty, return None
- **is_empty**(): return true if queue is empty
- **is_full**(): return true if queue is full

Lab 3

- Python (lab1)

  5. misc.

  6. PEP8
- review Linked List, Doubly Linked List
- exercise 📝 lab3.ipynb

Lab 3

Exercise

- implement Stack by linked list
- implement Queue by linked list
- reverse a linked list
  - recursive implementation
  - iterative implementation

Lab 4

- review Binary Tree and 4 traverse methods
- exercise 📝 lab4.ipynb

Lab 4

Exercise

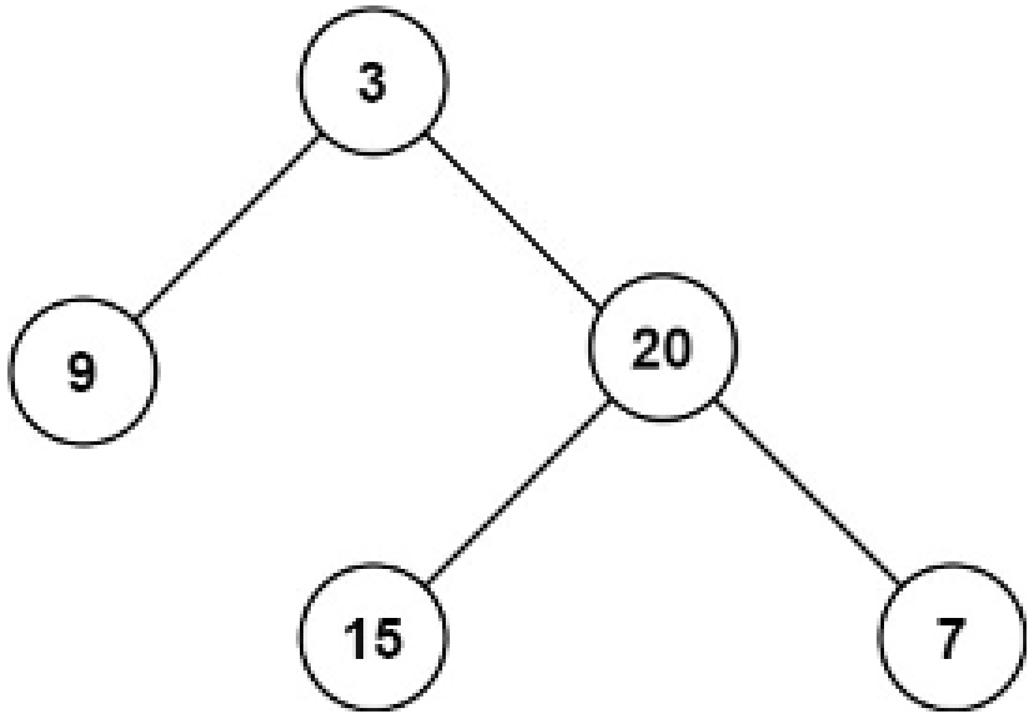
- convert binary tree from linked list to array
- convert binary tree from array to linked list
- check a balanced binary tree

Lab 4

Exercise: get maximum depth of binary tree

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

For example: the maximum depth is 3

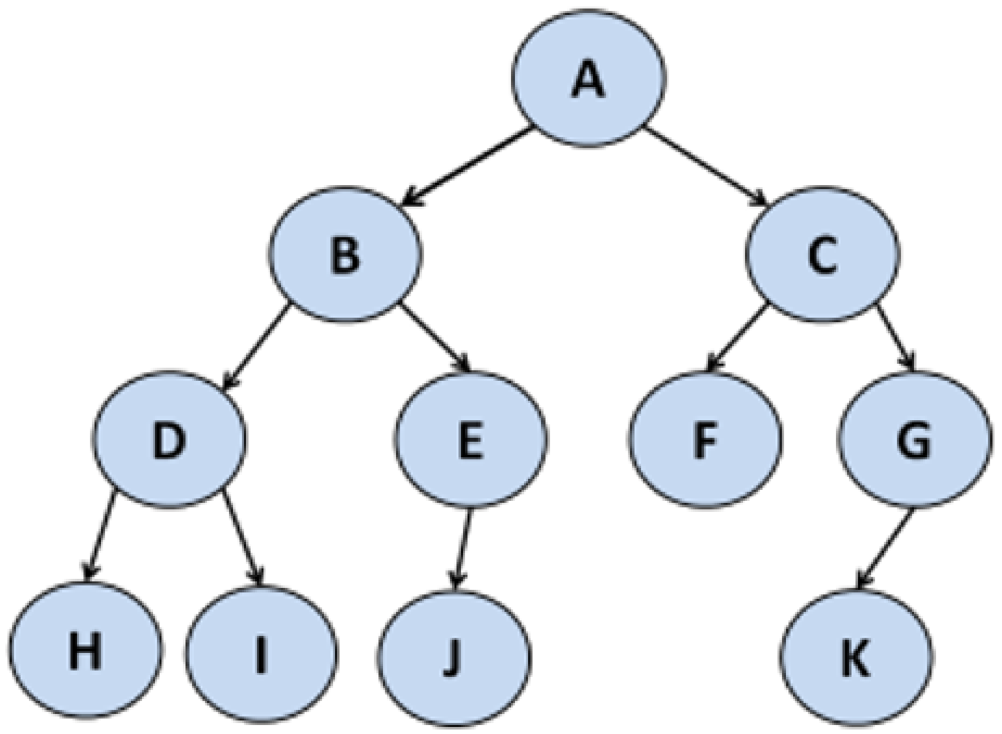


Lab 4

Exercise: get minimum depth of binary tree

A binary tree's minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

For example: the minimum depth is 3

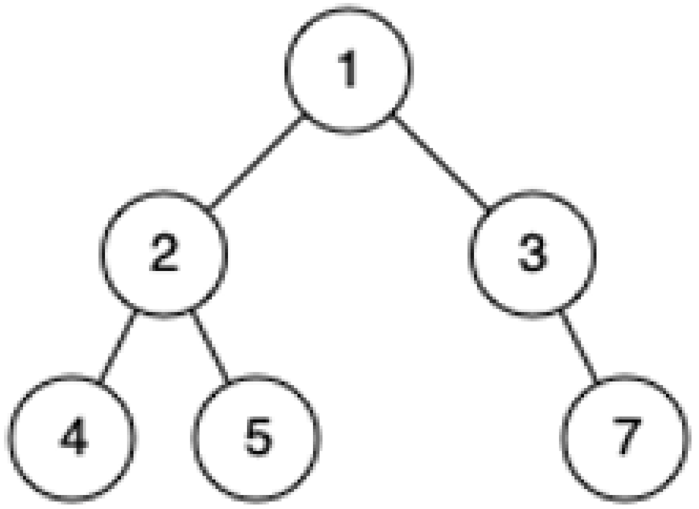


Lab 4

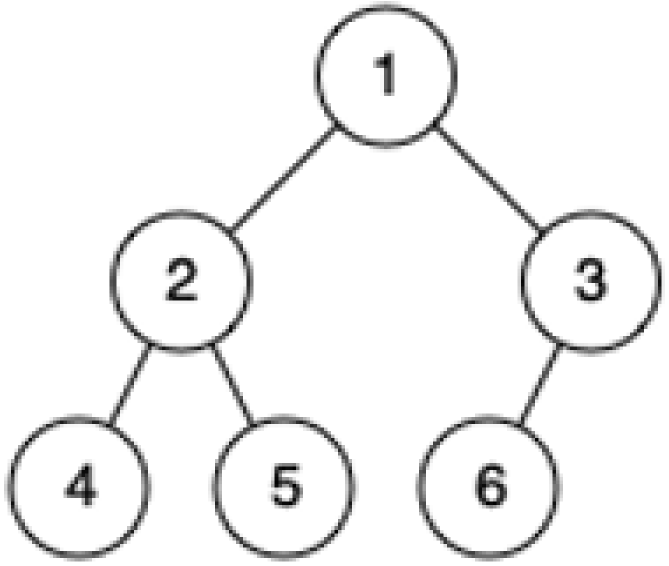
Exercise: check a complete binary tree

In a complete binary tree, every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

complete = true:



complete = false:



Lab 5

- review linear search, binary search
- let us do some exercises 📝 lab5.ipynb

Lab 5

lineary search & binary search

1. Go to https://www.cs.usfca.edu/~galles/visualization/Search.html to understand how Linear Search & Binary Search is working

2. Implement Linear Search & Binary Search in Python by yourself:

- familiar with Python coding style
- understand the input, output, steps and ending condition
- learn and compare different approaches (time & space complexity)
- test code reliability with different cases

Lab 5

Exercise: palindrome

Implement a Python function to determines if a string is a palindrome, for example, ‘racecar’ and ‘level’ are palindromes

Lab 5

Exercise: remove duplicate numbers

Given a sorted array nums, remove the duplicates in-place such that each element appears only once and returns the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

Example 1

Input: *nums = [1,1,2]*

Output: *2, nums = [1,2]*

Explanation: Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the returned length.

Example 2

Input: *nums = [0,0,1,1,1,2,2,3,3,4]*

Output: *5, nums = [0,1,2,3,4]*

Explanation: Your function should return length = 5, with the first five elements of nums being modified to 0, 1, 2, 3, and 4 respectively. It doesn't matter what values are set beyond the returned length.

Lab 6

TMA review