NHẬP MÔN LẬP TRÌNH
(Introduction to Programming)

Chapter 7 – Algorithm with Python
(Lesson 2 – Introduction to Algorithm)

Presenter: **Dr. Nguyen Dinh Long**

Email: dinhlonghcmut@gmail.com

Phone: +84 947 229 599

Google-site: https://sites.google.com/view/long-dinh-nguyen

Dec. 2022

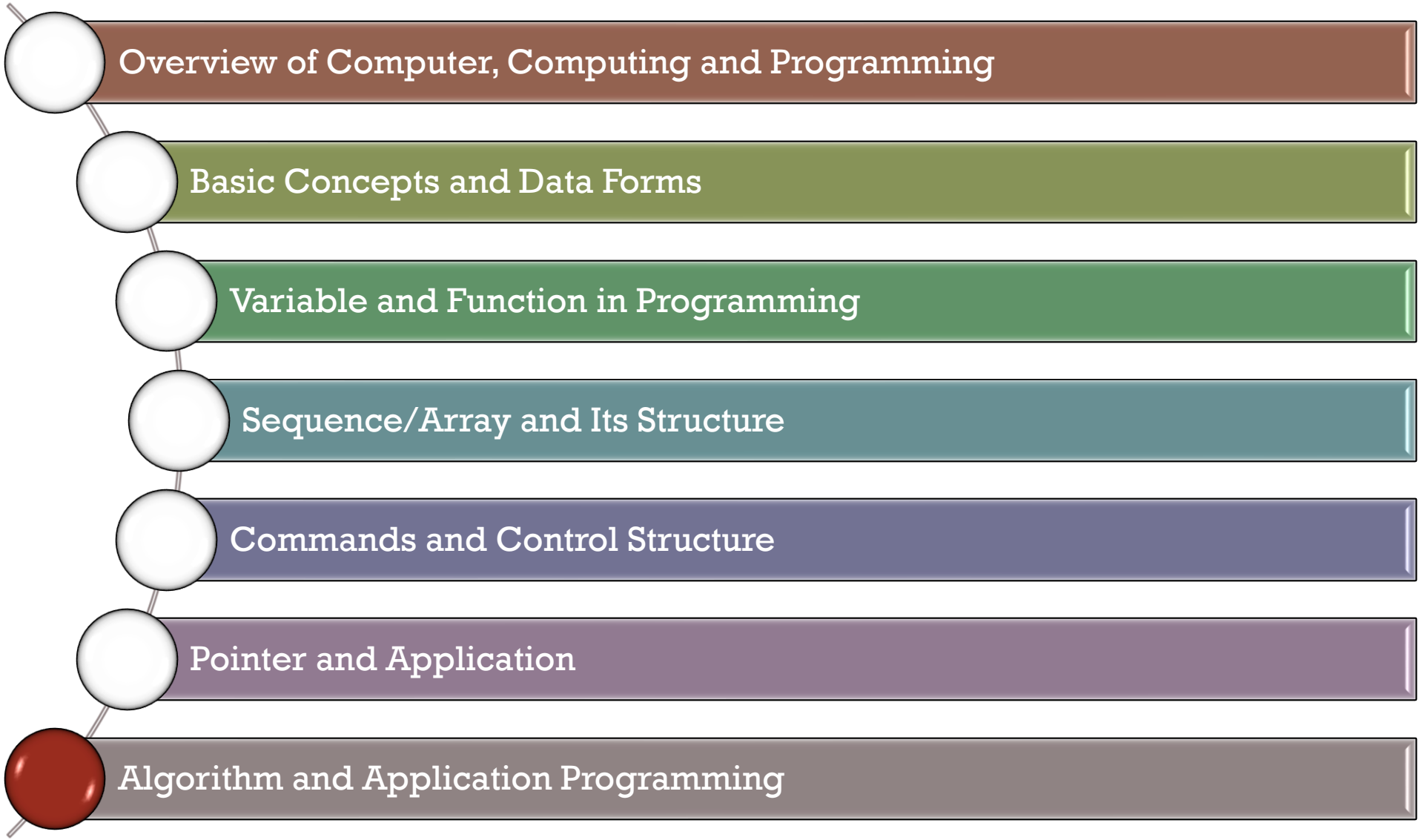
Programming

Hamilton was a self-taught programmer, working in the US in the 1960's. Owing to the success of her previous work, Hamilton was the first programmer to be hired for the Apollo project. She became the Director of Software Engineering at the MIT Instrumentation lab. Her lab developed the on-board flight software for NASA's Apollo space project, which took humankind to the moon.

The achievement was a monumental task at a time when computer technology was in its infancy: The astronauts had access to only 72 kilobytes of computer memory (a 256-gigabyte cell phone today carries almost a million times more storage space). Programmers had to use paper punch cards to feed information into room-sized computers with no screen interface.

Margaret Hamilton, NASA's lead software engineer for the Apollo, stands next to the code she wrote by hand that took humanity to the moon in 1969.

Outline


Overview of Computer, Computing and Programming
Basic Concepts and Data Forms
Variable and Function in Programming
Sequence/Array and Its Structure
Commands and Control Structure
Pointer and Application
Algorithm and Application Programming

References

Main:

- Maurizio Gabbrielli and Simone Martini, 2010. *Programming Languages: Principles and Paradigms*, Springer.
- Cao Hoàng Trụ, 2004. *Ngôn ngữ lập trình- Các nguyên lý và mô hình*, Nhà xuất bản Đại học Quốc gia Tp. Hồ Chí Minh

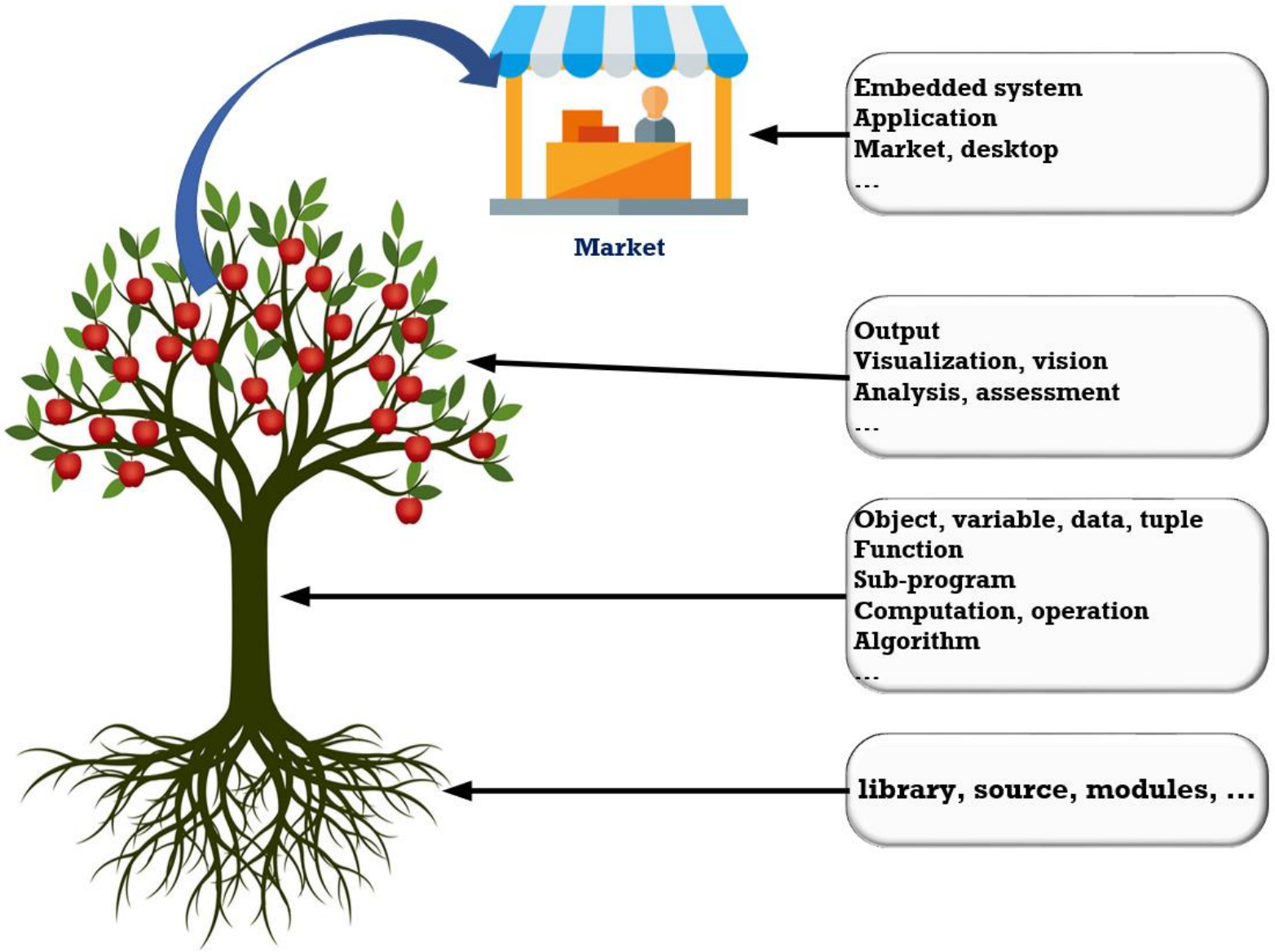
More:

- Wes McKinney, 2013. *Python for Data Analysis*, O'Reilly Media.
- Guido van Rossum, Fred L. Drake, Jr.,, 2012. *The Python Library Reference*, Release 3.2.3.

- Slides here are collected and modified from several sources in Universities and Internet.

Computer programs

- **General structure:**



Content of Chapter 7 (lesson 2)

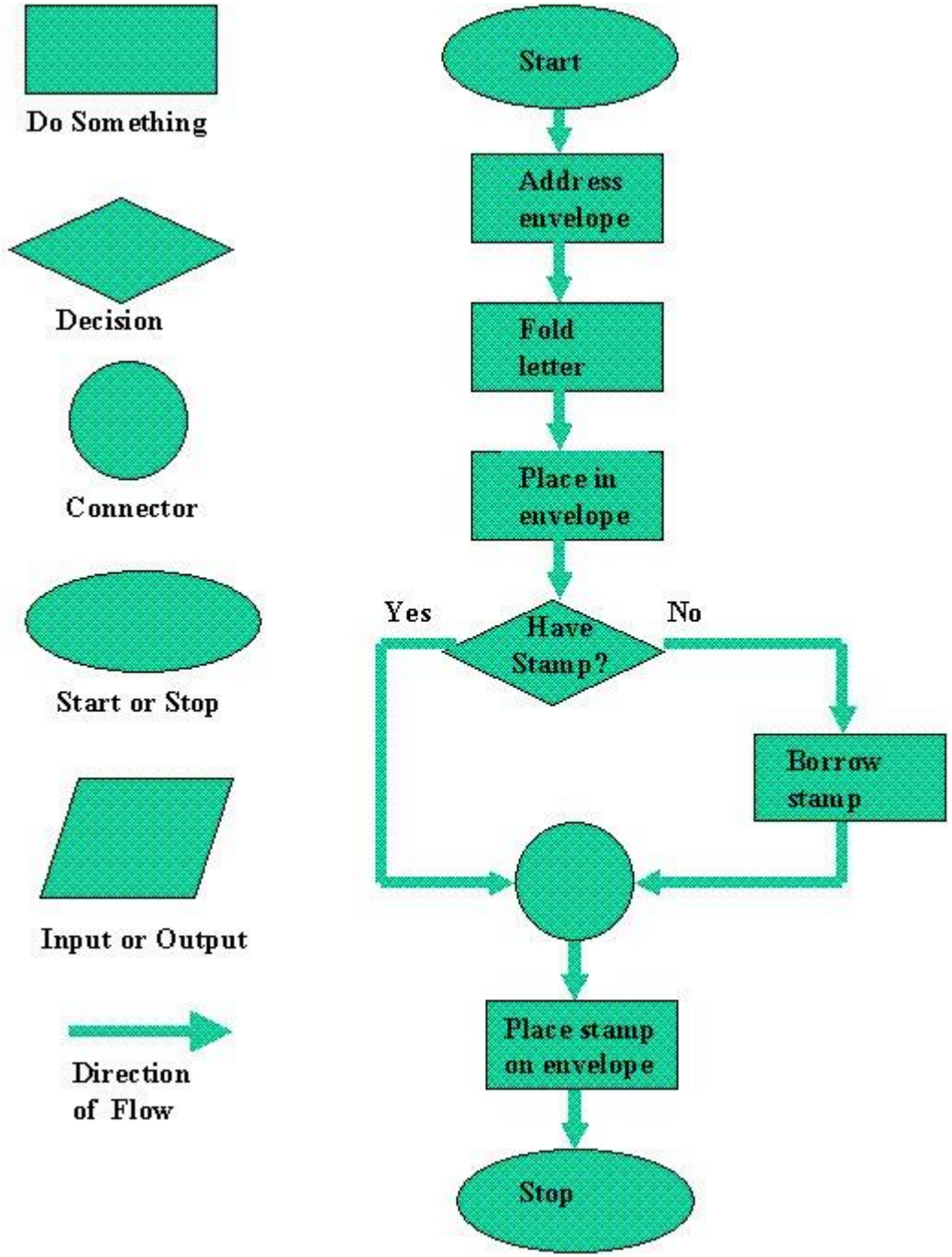
1. Introduction to Algorithm
2. Mathematics of Algorithm in Programming
3. How to Design an Algorithm
4. A Story of Algorithm
5. Application and Practice

Structure of Computer programs

- **Python Programming Analysis:**
  - Objects
  - Types: boolean, integer, float, string, complex
  - Variables: global variables, local variables
  - Methods, Calculation, Computations
  - Classes, functions
  - Sequences: list, tuples, set, dictionary
  - Arrays: 1D array (vector), 2D array (matrix), n-D array ...

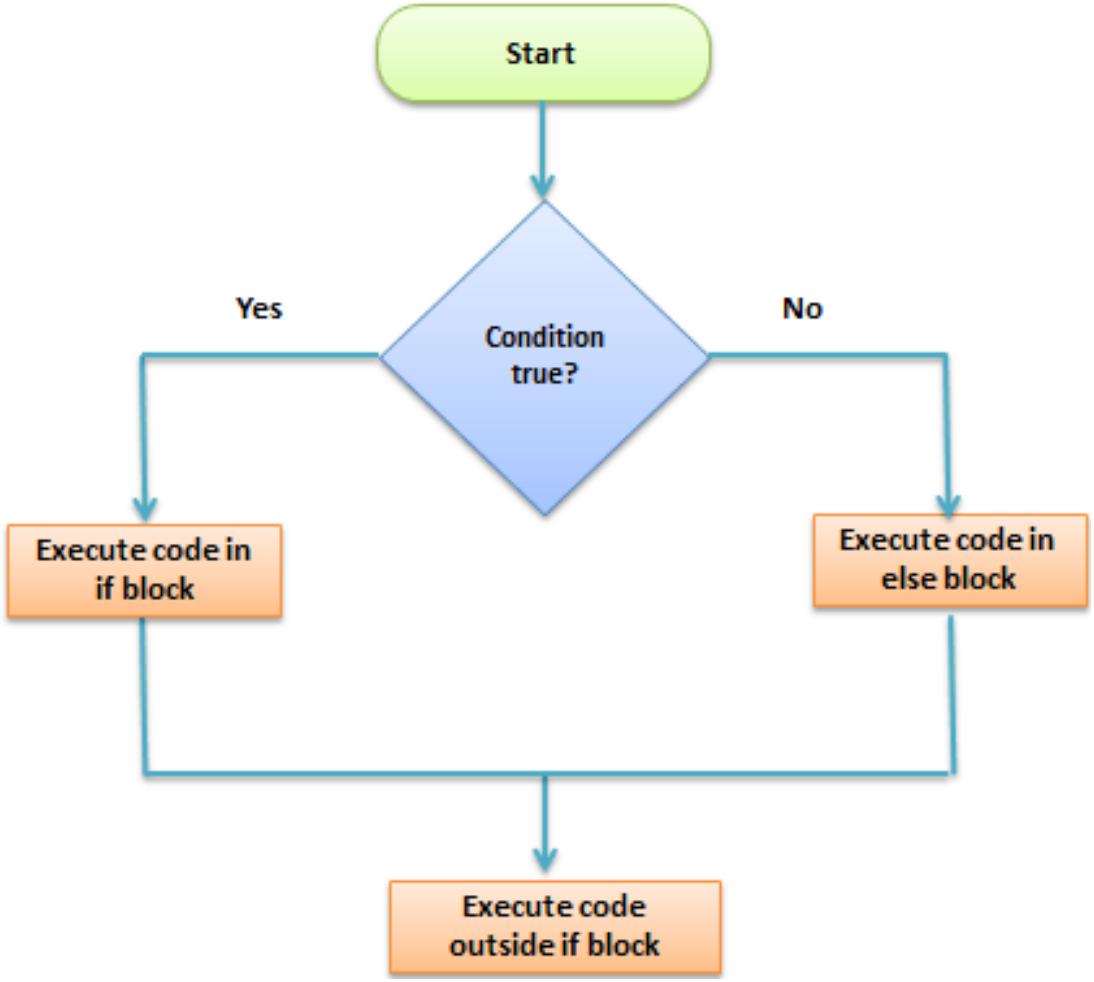
Structure of Computer programs

- **Computer programming:**
  - Modeling
  - Data Reading-Writing-Updating-Deleting
  - Flow charts, diagram, graph/figure ...

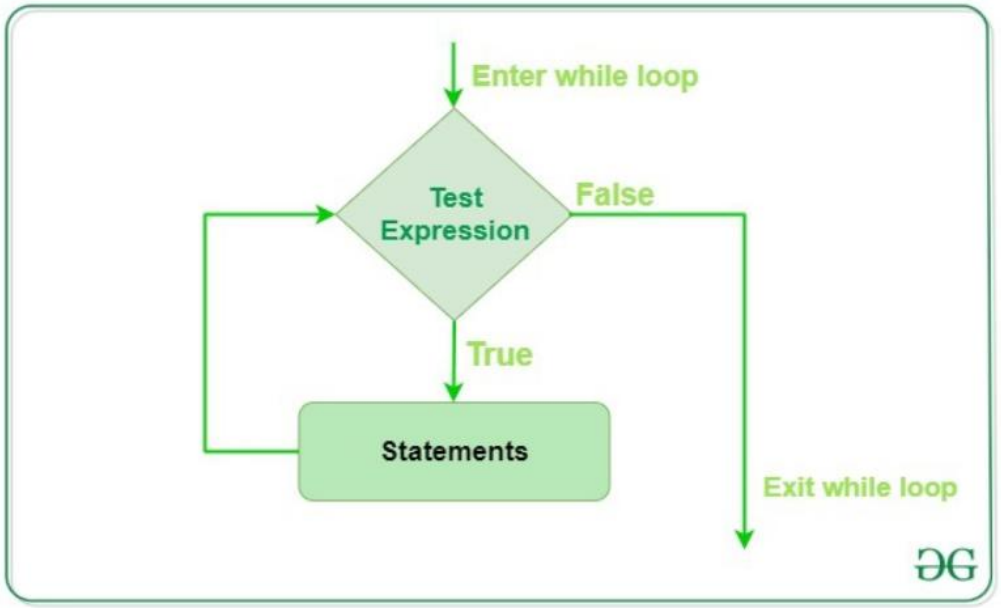


Python Command/Control/Statement

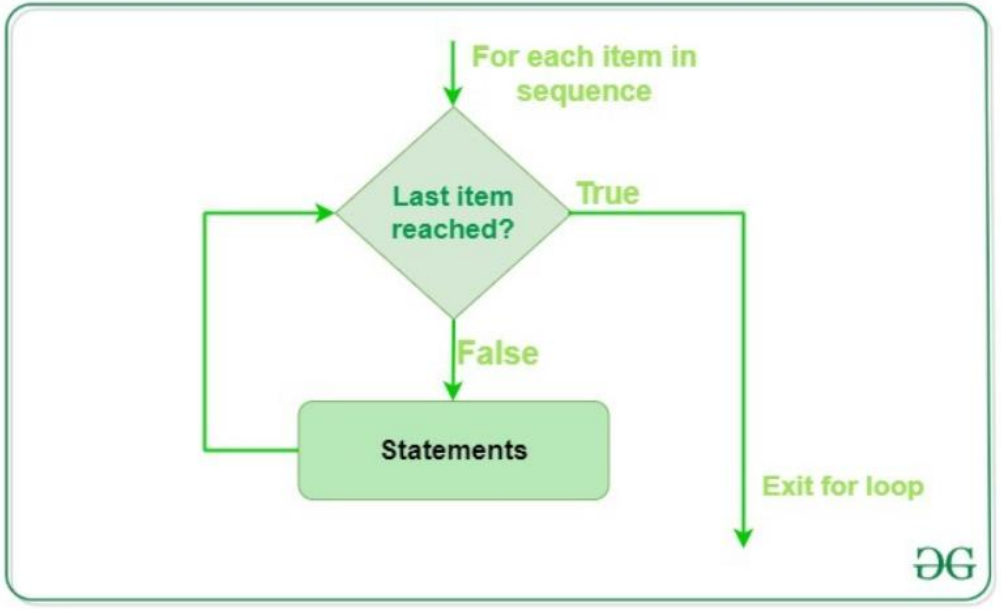
- For ... loop, While ... Loop, If/else ... statement



Flowchart of While Loop :

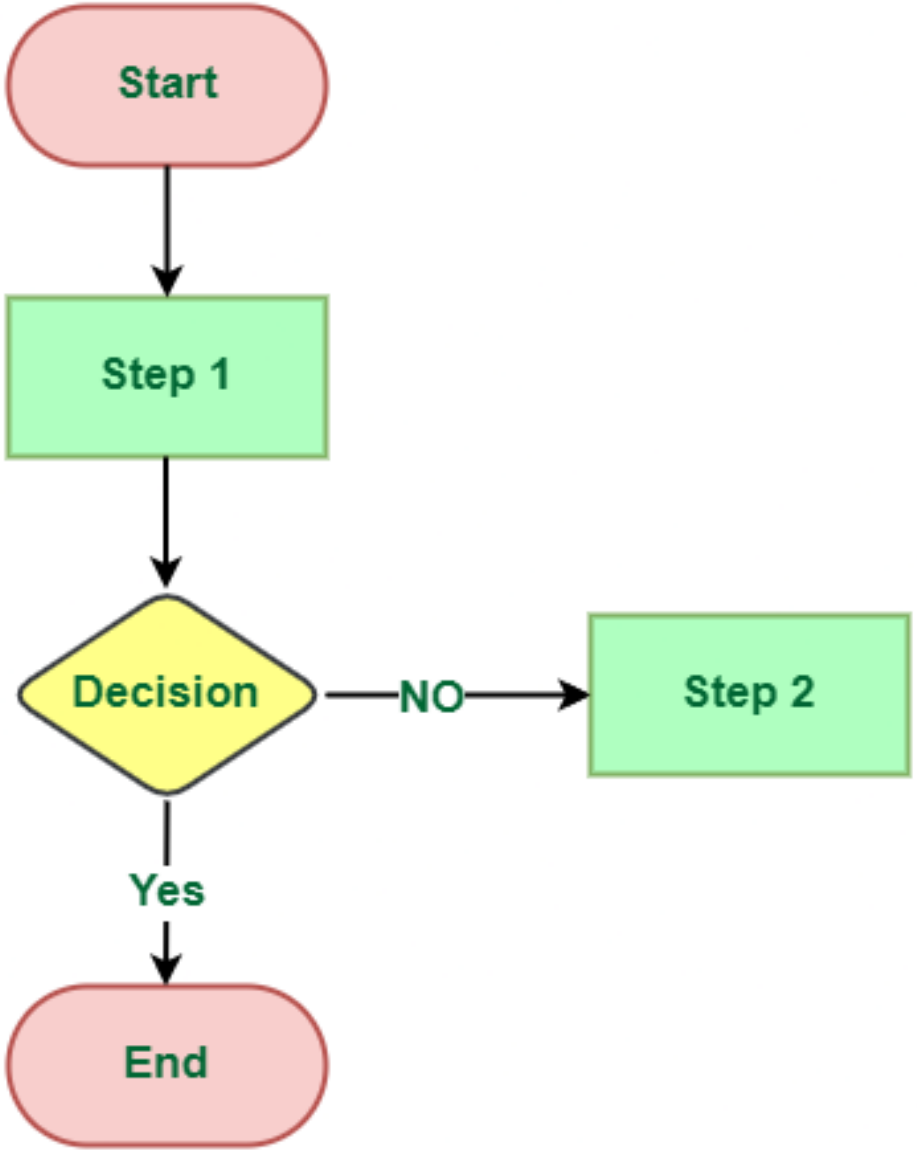


Flowchart of for loop



Python Programming

Algorithms



Python Algorithms

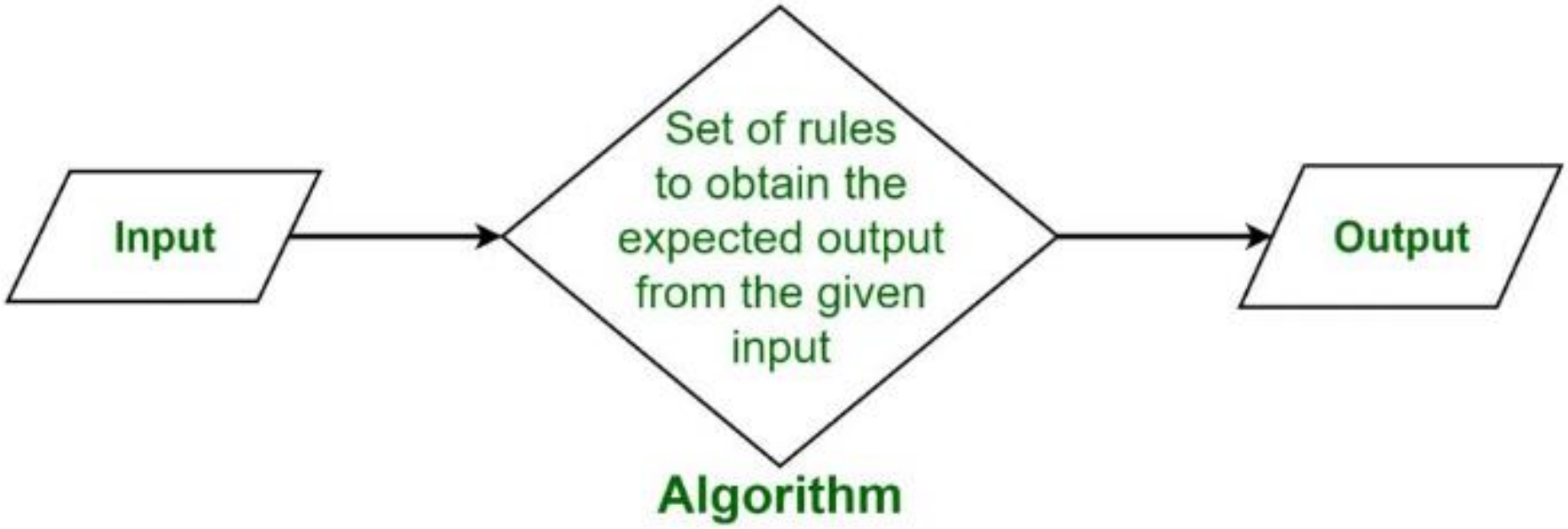
What is an algorithm?

- An algorithm is a procedure to accomplish a specific task. An algorithm is the idea behind any reasonable computer program.
- ”A set of finite rules or instructions to be followed in calculations or other problem-solving operations ”. ”A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations”.

Problem: Sorting

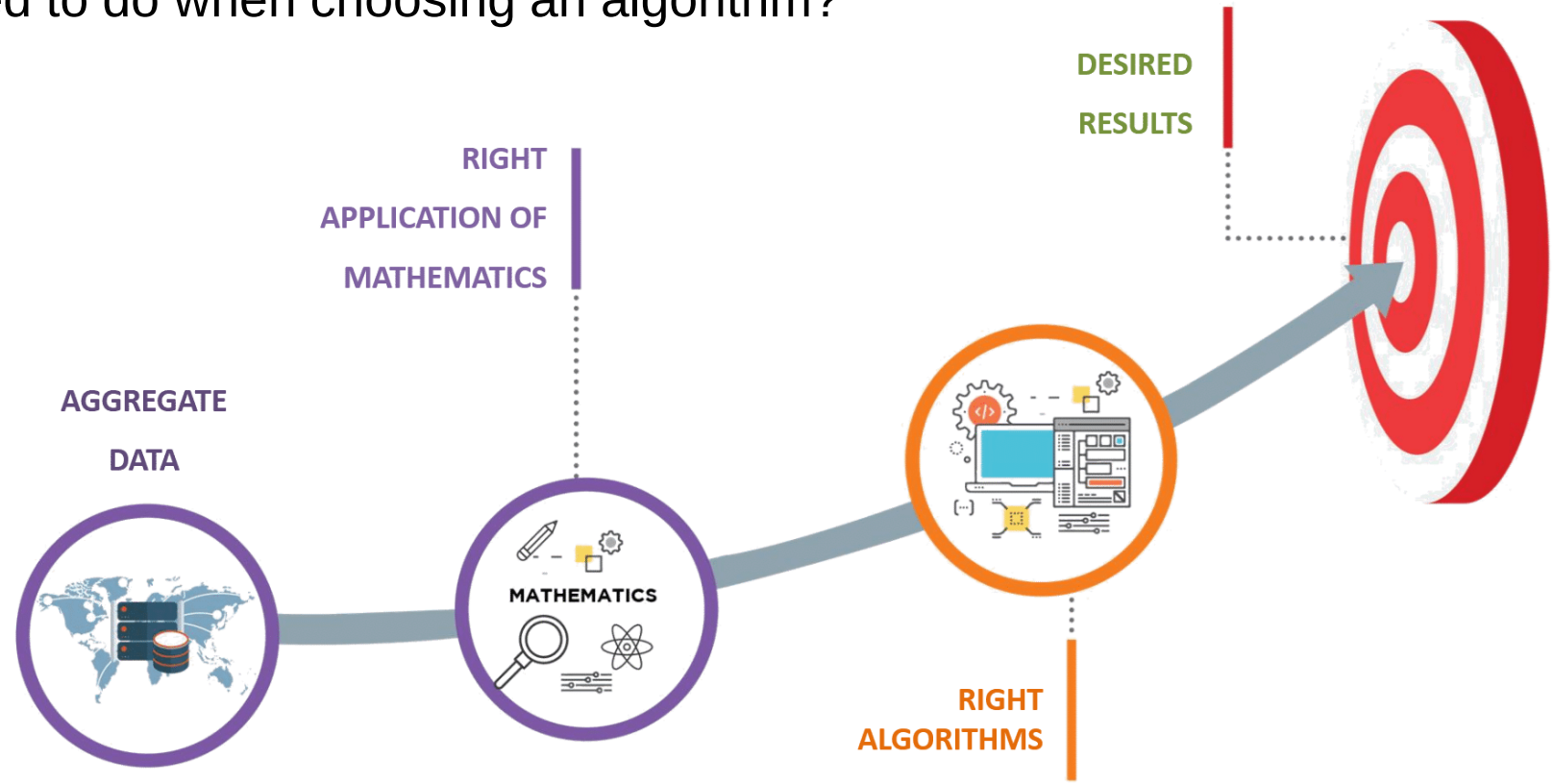
Input: A sequence of n keys $a_1, \ldots, a_n$.

Output: The permutation (reordering) of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_{n-1} \leq a'_n$.



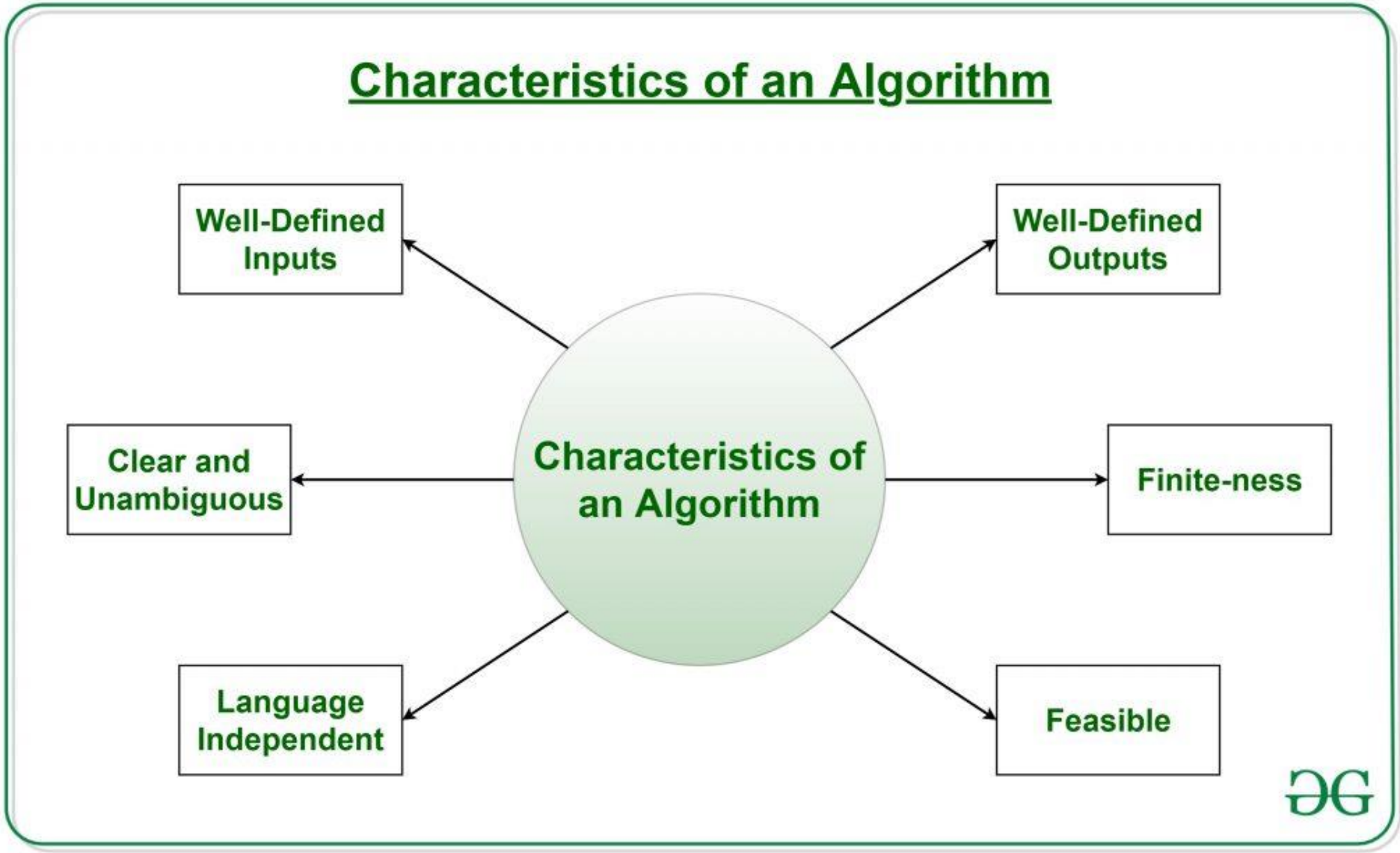
Python Algorithms

- **Thinking in Algorithms:**
- Data structures have been tightly tied to algorithms since the dawn of computing.
- A number of general approaches used by algorithms to solve problems.
- What do you need to do when choosing an algorithm?



Python Algorithms

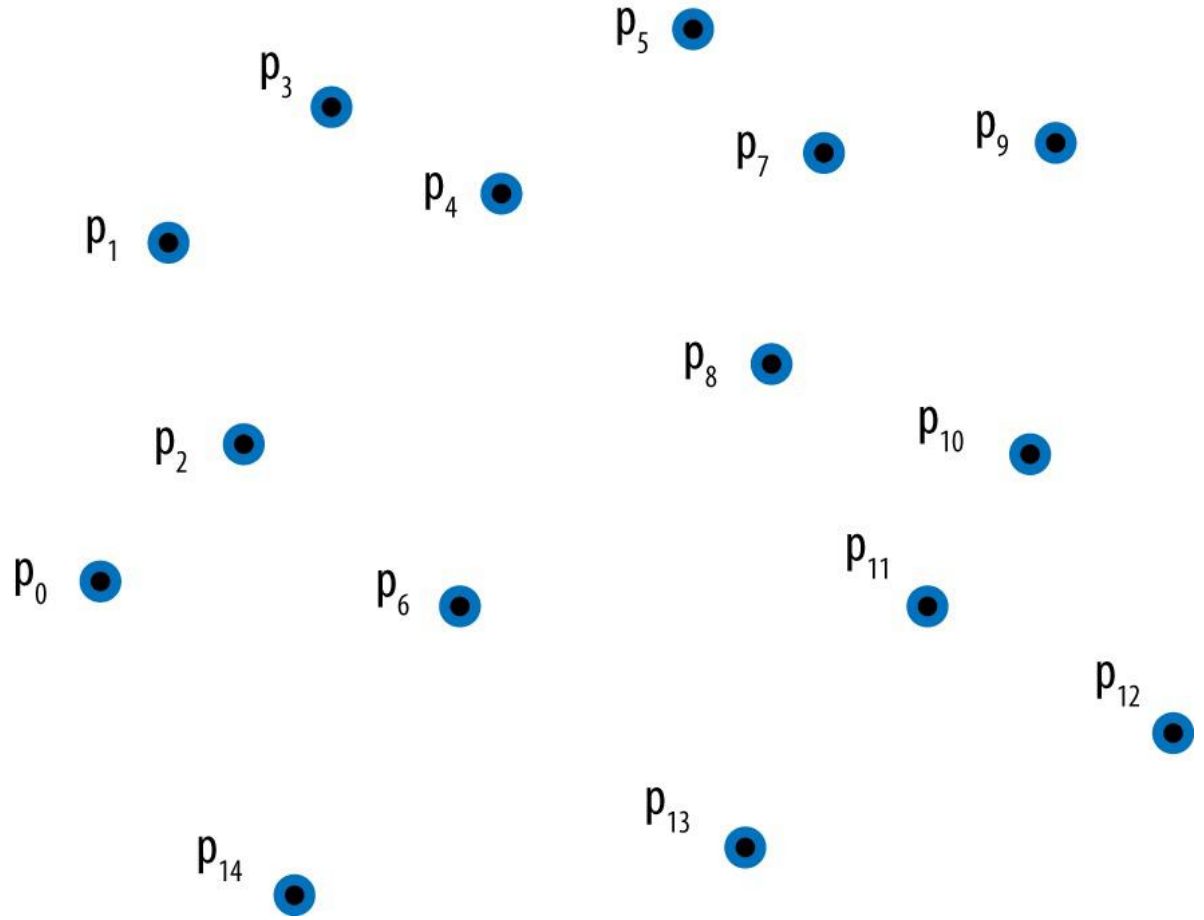
- **Thinking in Algorithms**



Python Algorithms

❑ Understand the Problem:

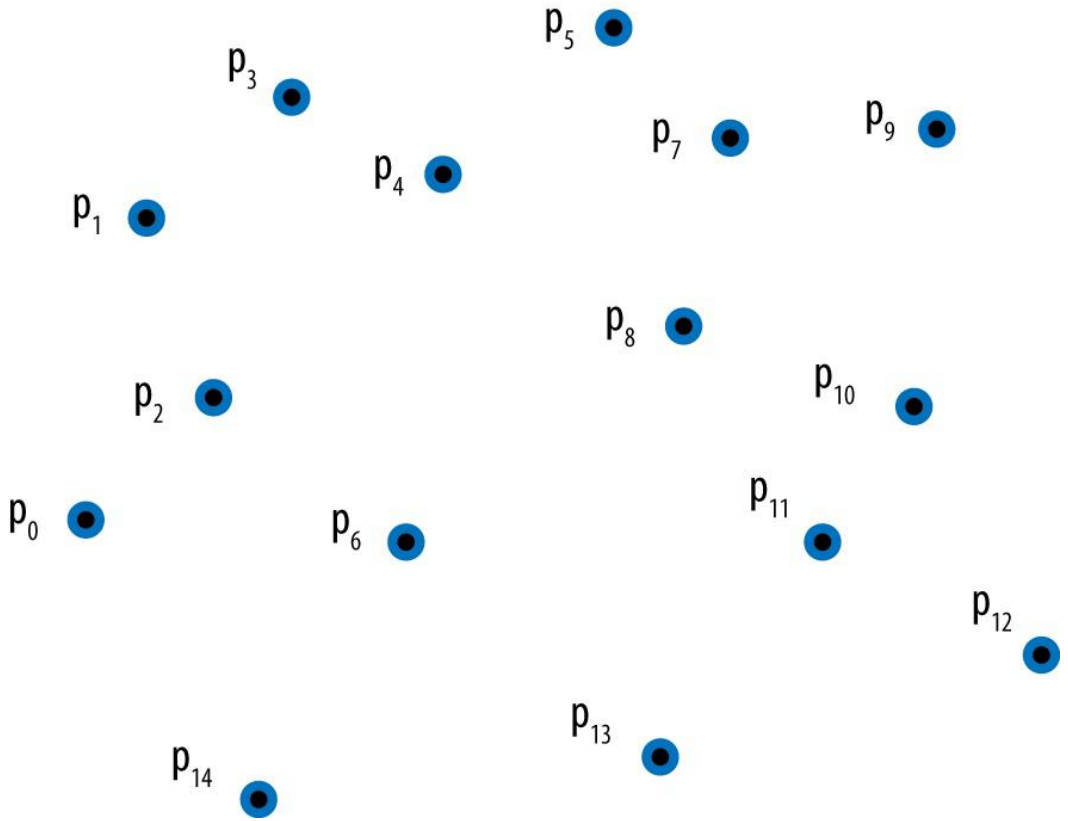
- The first step in designing an algorithm is to understand the problem you want to solve.
- Let's start with a sample problem from the field of computational geometry.

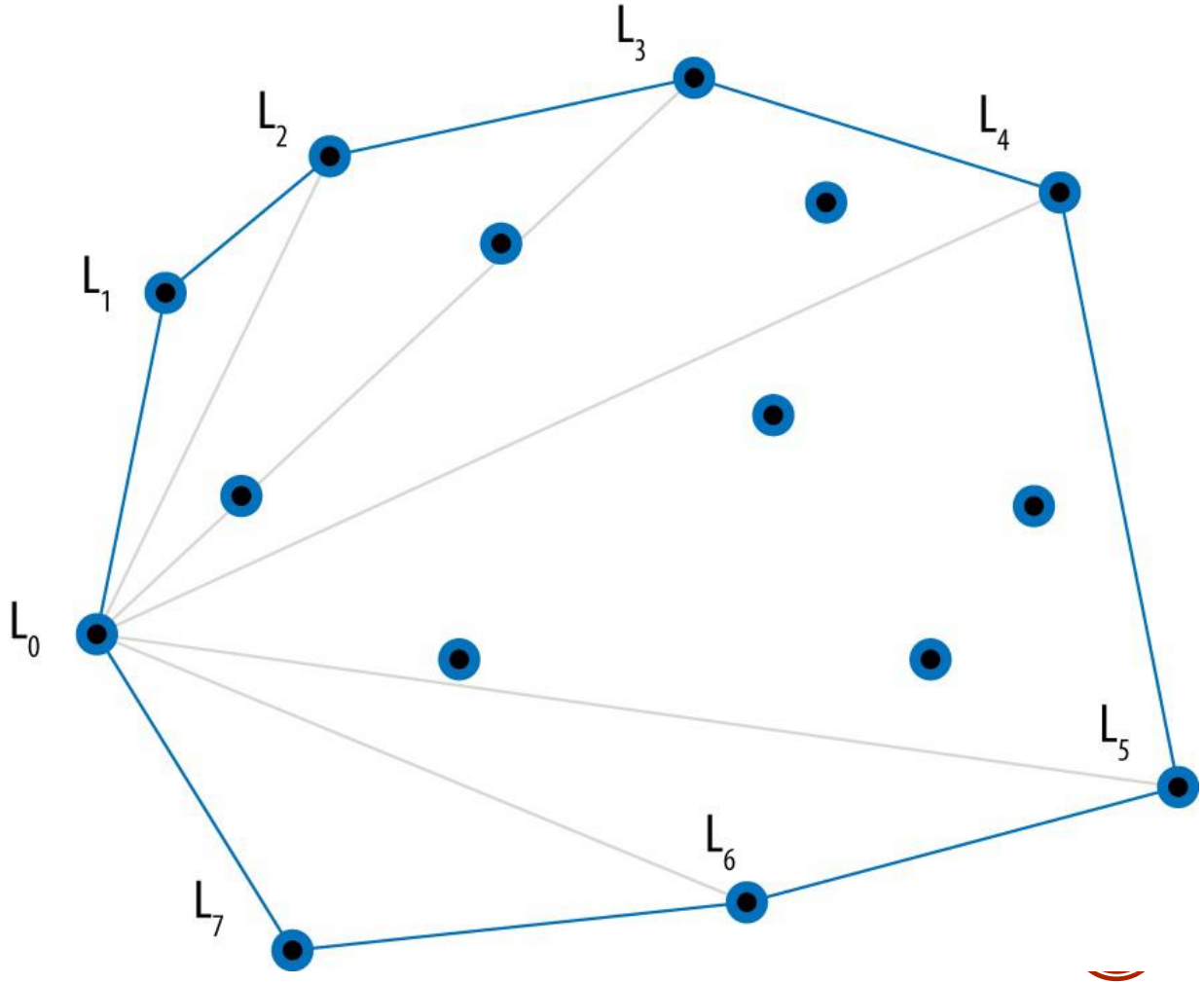


Python Algorithms

Understand the Problem:

- Let's start with a sample problem from the field of computational geometry.





Python Algorithms

❑ The Mathematics of Algorithms:

One of the most important factors for choosing an algorithm is the speed with which it is likely to complete. Characterizing the expected computation time of an algorithm is inherently a mathematical process.

- Size of a Problem Instance: An instance of a problem is a particular input data set given to a program. In most problems, the execution time of a program increases with the size of this data set.

- Rate of Growth of Functions: We describe the behavior of an algorithm by representing the rate of growth of its execution time as a function of the size of the input problem instance. Characterizing an algorithm's performance in this way is a common abstraction that ignores numerous details.

Python Algorithms

The Mathematics of Algorithms:

- Analysis in the Best, Average, and Worst Cases: One question to ask is whether the results of the algorithm will be true for all input problem instances. The data could contain large runs of elements already in sorted order. The input could contain duplicate values. Regardless of the size n of the input set, the elements could be drawn from a much smaller set and contain a significant number of duplicate values.

Worst case

Defines a class of problem instances for which an algorithm exhibits its worst runtime behavior. Instead of trying to identify the specific input, algorithm designers typically describe properties of the input that prevent an algorithm from running efficiently.

Average case

Defines the expected behavior when executing the algorithm on random problem instances. While some instances will require greater time to complete because of some special cases, the vast majority will not. This measure describes the expectation an average user of the algorithm should have.

Best case

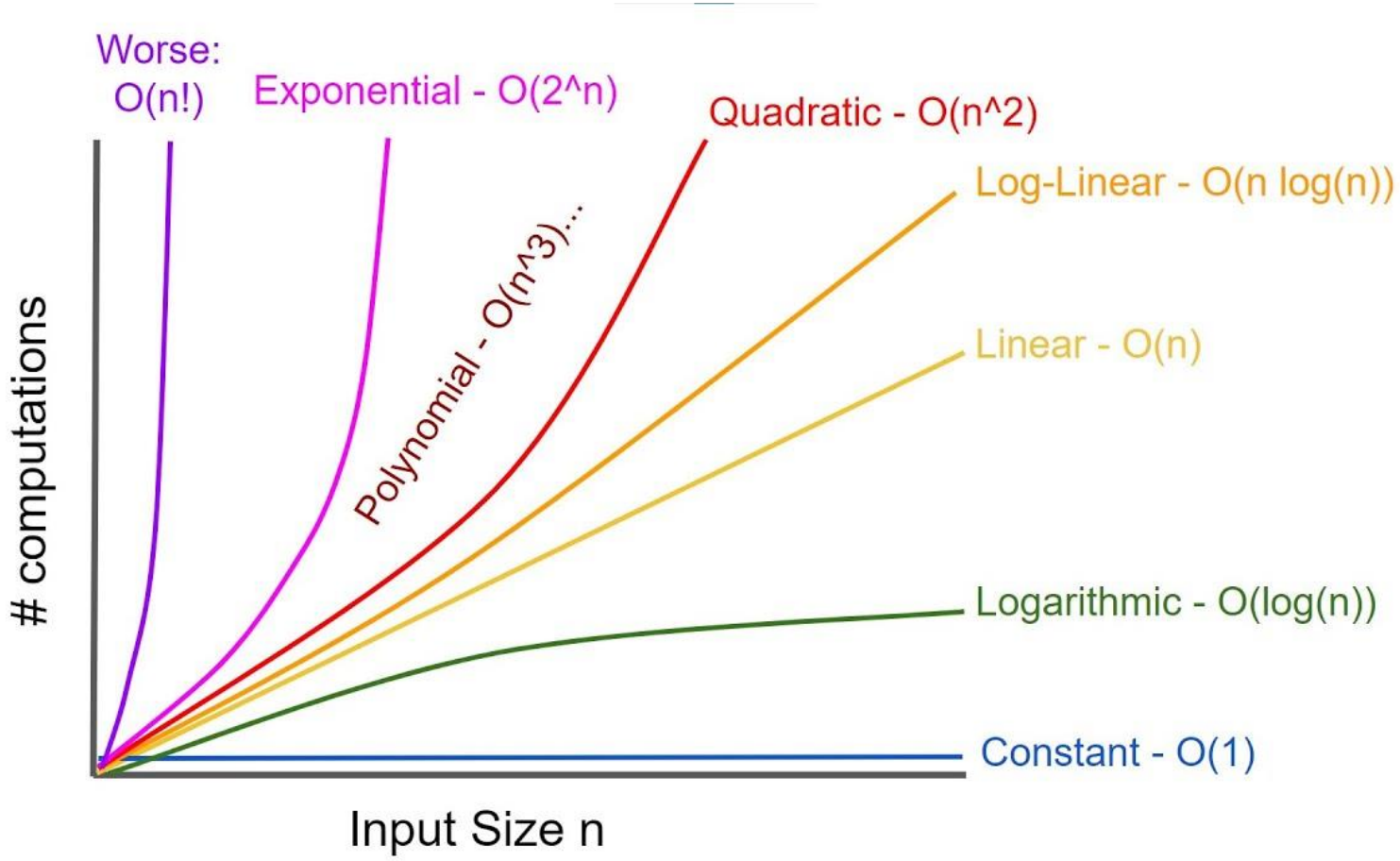
Defines a class of problem instances for which an algorithm exhibits its best runtime behavior. For these instances, the algorithm does the least work. In reality, the best case rarely occurs.

Python Algorithms

❑ Performance Families:

- We compare algorithms by evaluating their performance on problem instances of size n. By doing so, we can determine which algorithms scale to solve problems of a nontrivial size by evaluating the running time needed by the algorithm in relation to the size of the provided input.

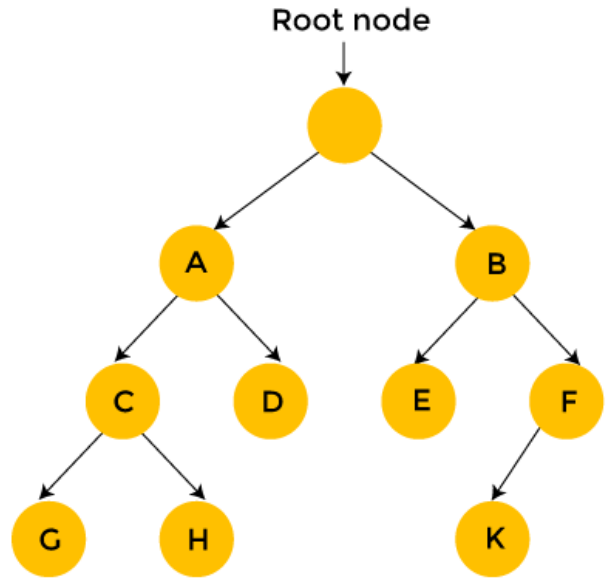
  - Constant: $O(1)$
  - Logarithmic: $O(\log n)$
  - Sublinear: $O(n^d)$ for $d < 1$
  - Linear: $O(n)$
  - Linearithmic: $O(n \log n)$
  - Quadratic: $O(n^2)$
  - Exponential: $O(2^n)$



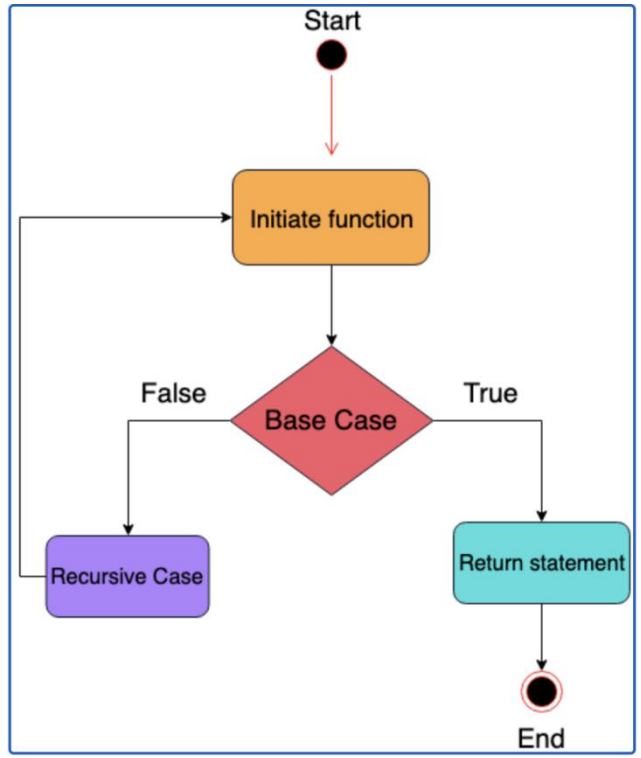
Python Algorithms

❑ Types of Algorithms:

- Brute Force Algorithm: It is the simplest approach for a problem. A brute force algorithm is the first approach that comes to finding when we see a problem.



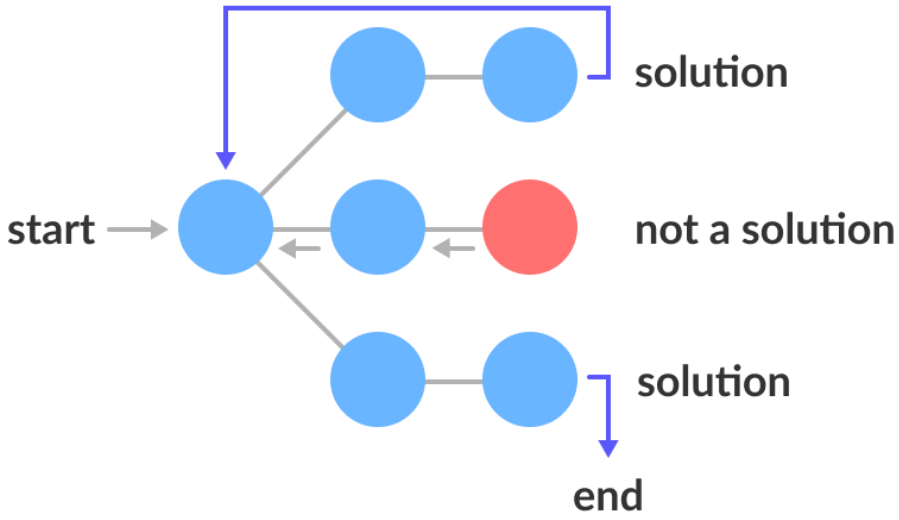
- Recursive Algorithm: A recursive algorithm is based on recursion. In this case, a problem is broken into several sub-parts and called the same function again and again.



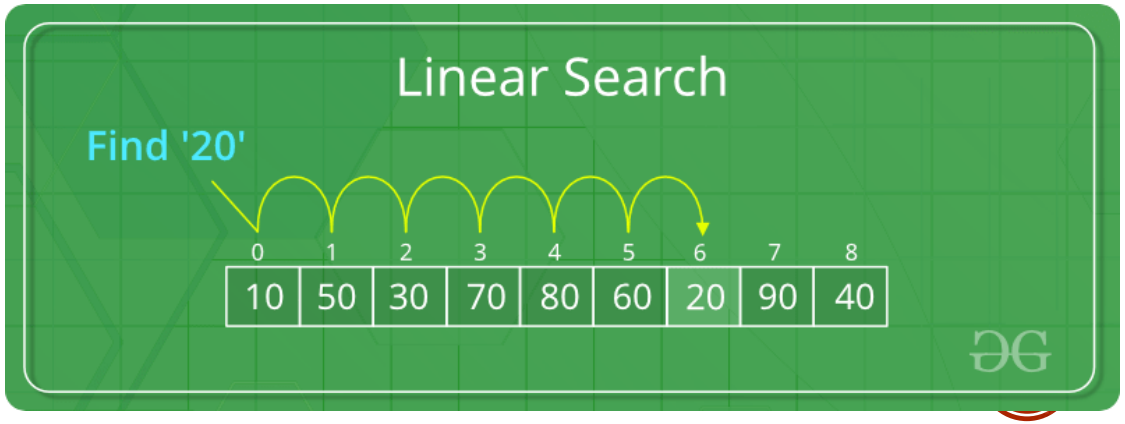
Python Algorithms

❑ Types of Algorithms:

- Backtracking Algorithm: The backtracking algorithm basically builds the solution by searching among all possible solutions. Using this algorithm, we keep on building the solution following criteria. Whenever a solution fails we trace back to the failure point and build on the next solution and continue this process till we find the solution or all possible solutions are looked after.



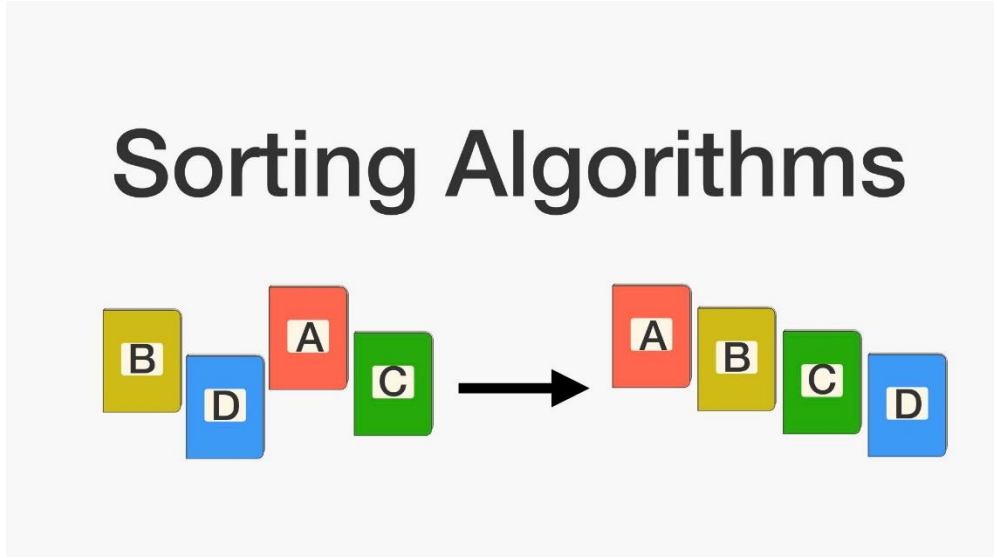
- Searching Algorithm: Searching algorithms are the ones that are used for searching elements or groups of elements from a particular data structure. They can be of different types based on their approach or the data structure in which the element should be found.



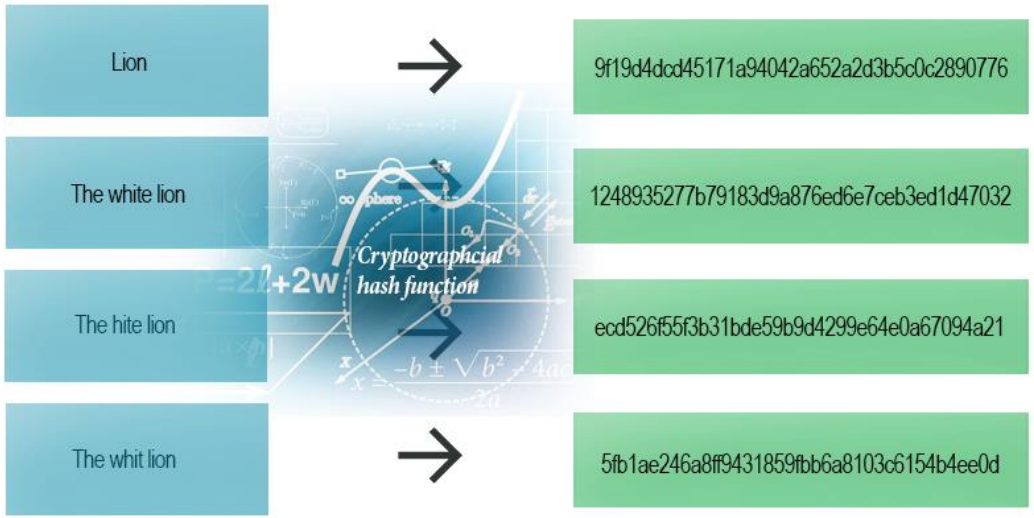
Python Algorithms

❑ Types of Algorithms:

- Sorting Algorithm: Sorting is arranging a group of data in a particular manner according to the requirement. The algorithms which help in performing this function are called sorting algorithms. Generally sorting algorithms are used to sort groups of data in an increasing or decreasing manner.



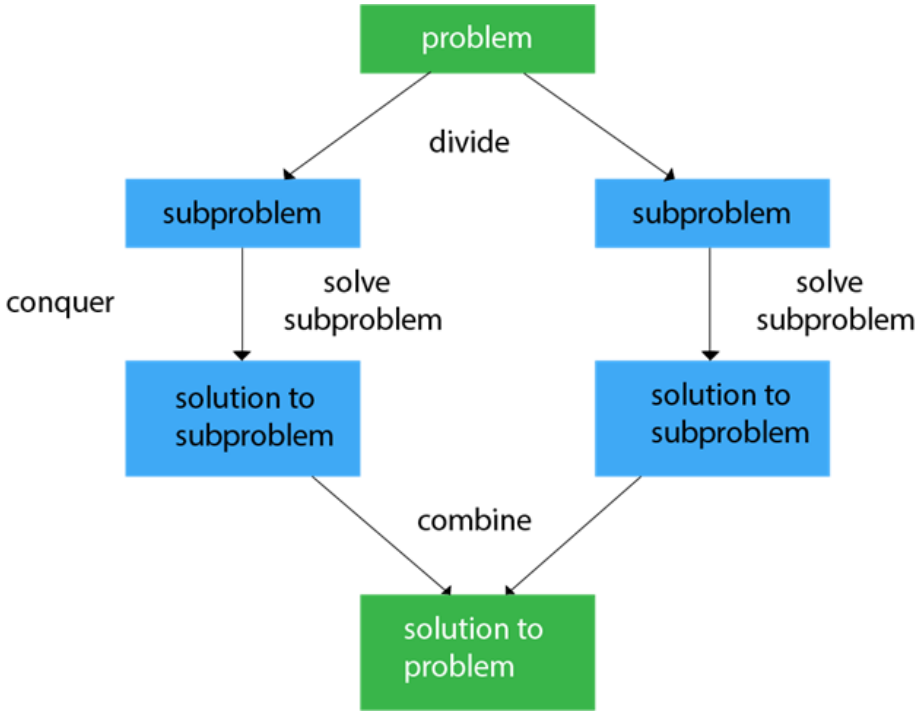
- Hashing Algorithm: Hashing algorithms work similarly to the searching algorithm. But they contain an index with a key ID. In hashing, a key is assigned to specific data.



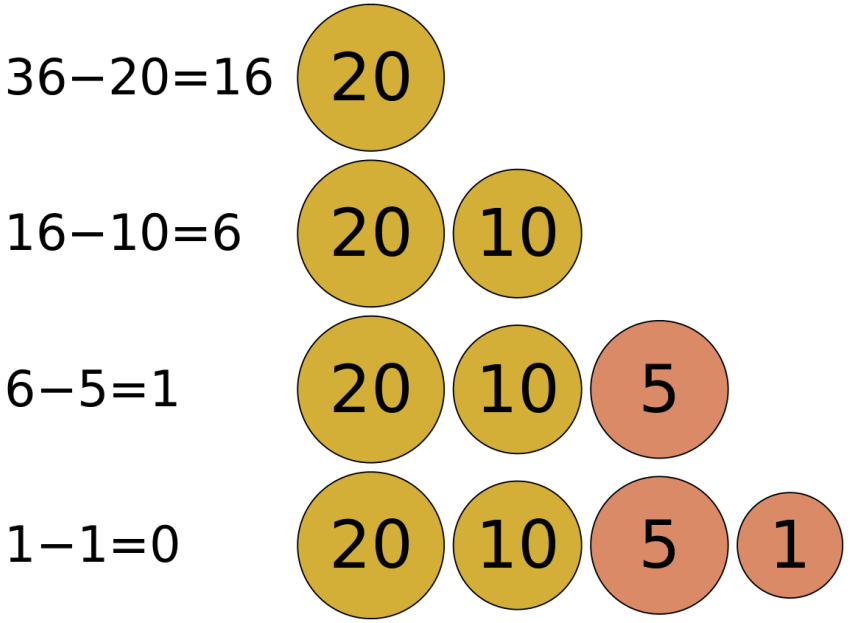
Python Algorithms

❑ Types of Algorithms:

- Divide and Conquer Algorithm: This algorithm breaks a problem into sub-problems, solves a single sub-problem and merges the solutions together to get the final solution. It consists of the following three steps: Divide – Solve - Combine



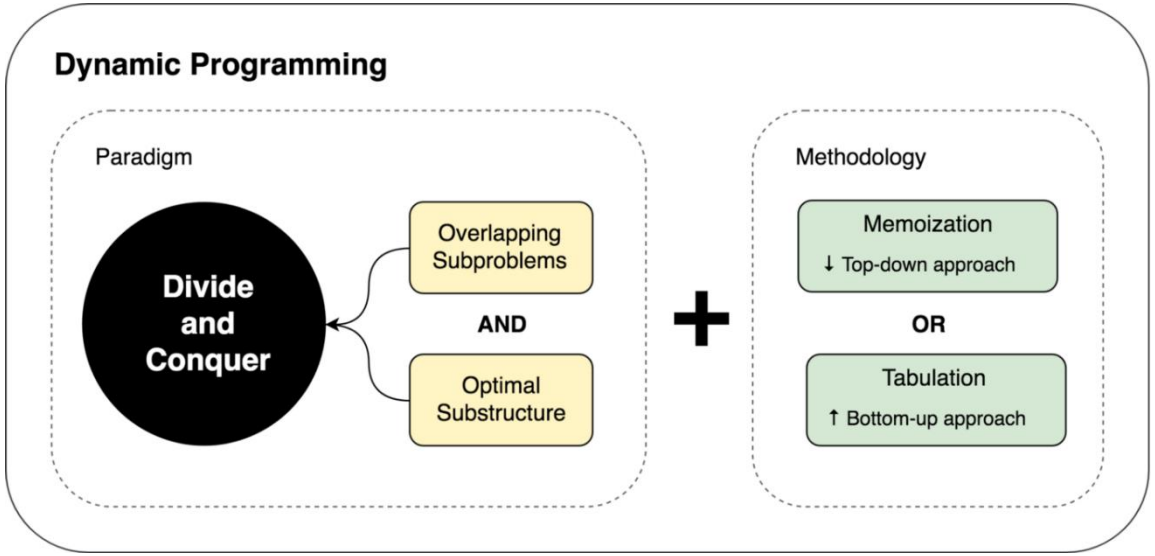
- Greedy Algorithm: In this type of algorithm the solution is built part by part. The solution of the next part is built based on the immediate benefit of the next part. The one solution giving the most benefit will be chosen as the solution for the next part.



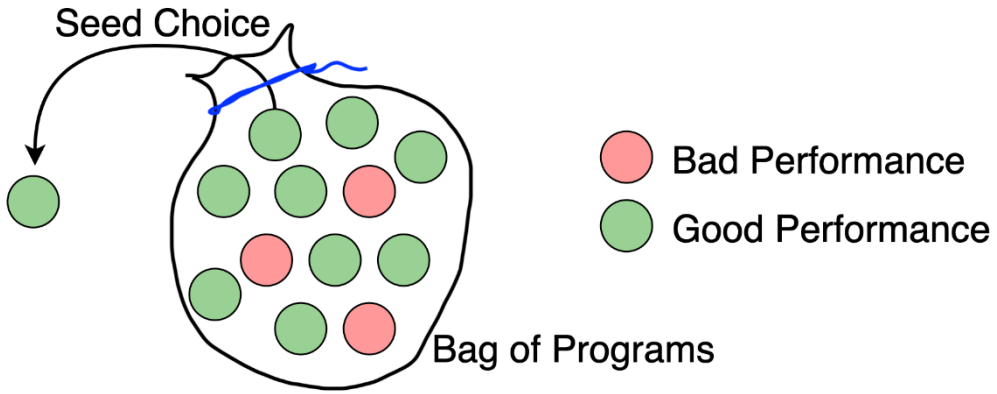
Python Algorithms

❑ Types of Algorithms:

- Dynamic Programming Algorithm: This algorithm uses the concept of using the already found solution to avoid repetitive calculation of the same part of the problem. It divides the problem into smaller overlapping subproblems and solves them.



- Randomized Algorithm: In the randomized algorithm we use a random number so it gives immediate benefit. The random number helps in deciding the expected outcome.



Python Algorithms

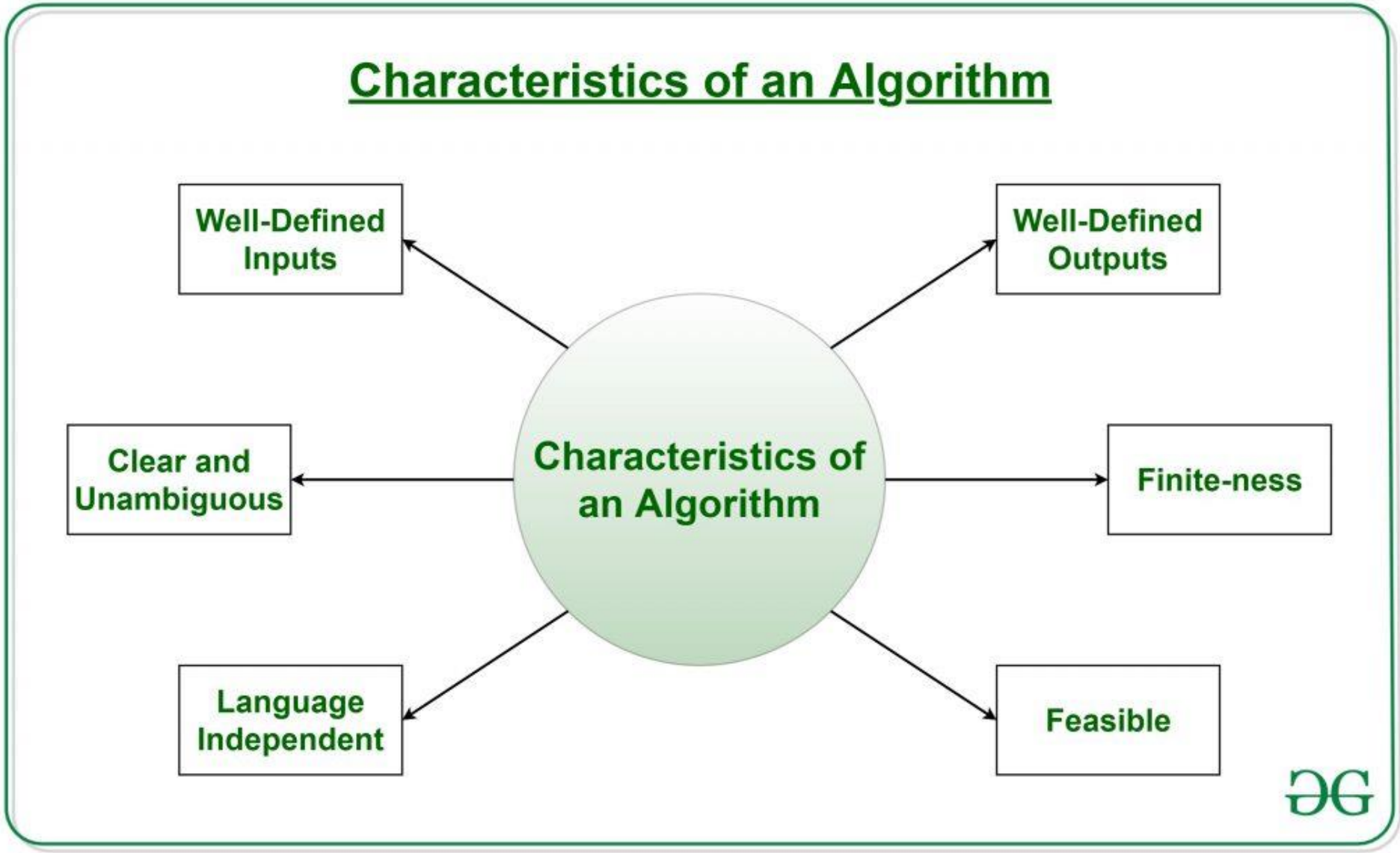
❑ Types of Algorithms:

- A search algorithm: Graph search algorithm that finds a path from a given initial node to a given goal node. It employs a heuristic estimate that ranks each node by an estimate of the best route that goes through that node.
- Binary search: Technique for finding a particular value in a linear array, by ruling out half of the data at each step.
- Branch and bound: A general algorithmic method for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization.
- Dijkstra's algorithm: Algorithm that solves the single-source shortest path problem for a directed graph with nonnegative edge weights.
- Gradient descent: Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to negative of the gradient (or approximate gradient) of the function at the current point.
- Newton's method: Efficient algorithm for finding approximations to the zeros (or roots) of a real-valued function in one or more dimensions. It can also be used to find local maxima and local minima of functions.
- Merge sort: A sorting algorithm for rearranging lists (or any other data structure accessed sequentially, e.g. file streams) into a specified order.
- Maximum flow: The maximum flow problem is finding a legal flow through a flow network that is maximal.

……

Python Algorithms

- **How to Design an Algorithm**

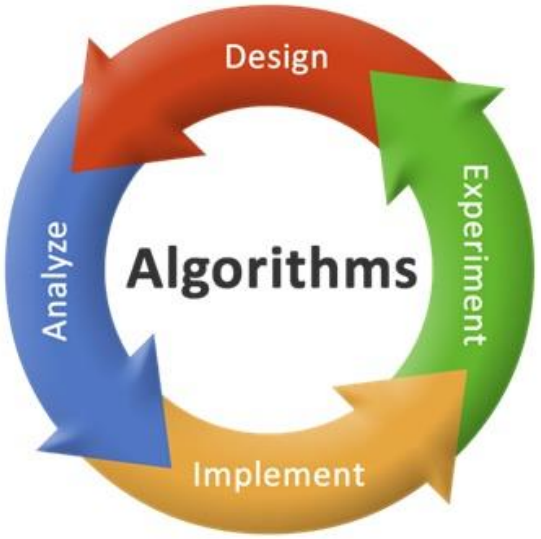


Python Algorithms

❑ How to Design an Algorithm:

In order to write an algorithm, the following things are needed as a pre-requisite:

- The problem that is to be solved by this algorithm i.e. clear problem definition.
- The constraints of the problem must be considered while solving the problem.
- The input to be taken to solve the problem.
- The output to be expected when the problem is solved.
- The solution to this problem, is within the given constraints.



Algorithm to add 3 numbers and print their sum:

1. START
2. Declare 3 integer variables num1, num2 and num3.
3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
4. Declare an integer variable sum to store the resultant sum of the 3 numbers.
5. Add the 3 numbers and store the result in the variable sum.
6. Print the value of the variable sum
7. END

Python Algorithms

❑ How to Design an Algorithm: Qualities of a Good Algorithm

- Input and output should be defined precisely.
- Each step in the algorithm should be clear and unambiguous.
- Algorithms should be most effective among many different ways to solve a problem.
- An algorithm shouldn't include computer code. Instead, the algorithm should be written in such a way that it can be used in different programming languages.

Algorithm 1: Add two numbers entered by the user

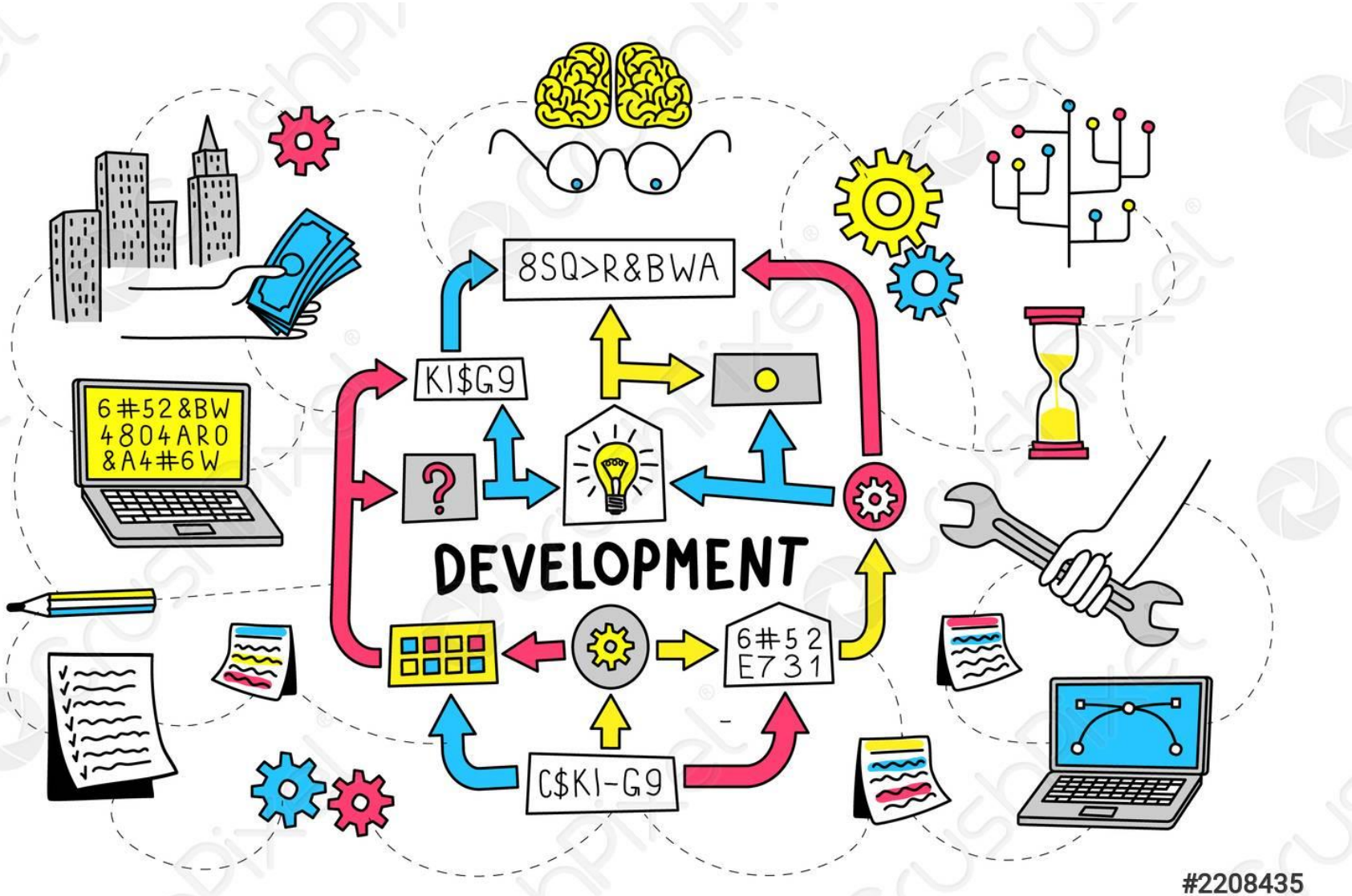
```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
        sum←num1+num2
Step 5: Display sum
Step 6: Stop
```

Algorithm 2: Find the largest number among three numbers

```
Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a > b
           If a > c
              Display a is the largest number.
           Else
              Display c is the largest number.
        Else
           If b > c
              Display b is the largest number.
           Else
              Display c is the greatest number.
Step 5: Stop
```

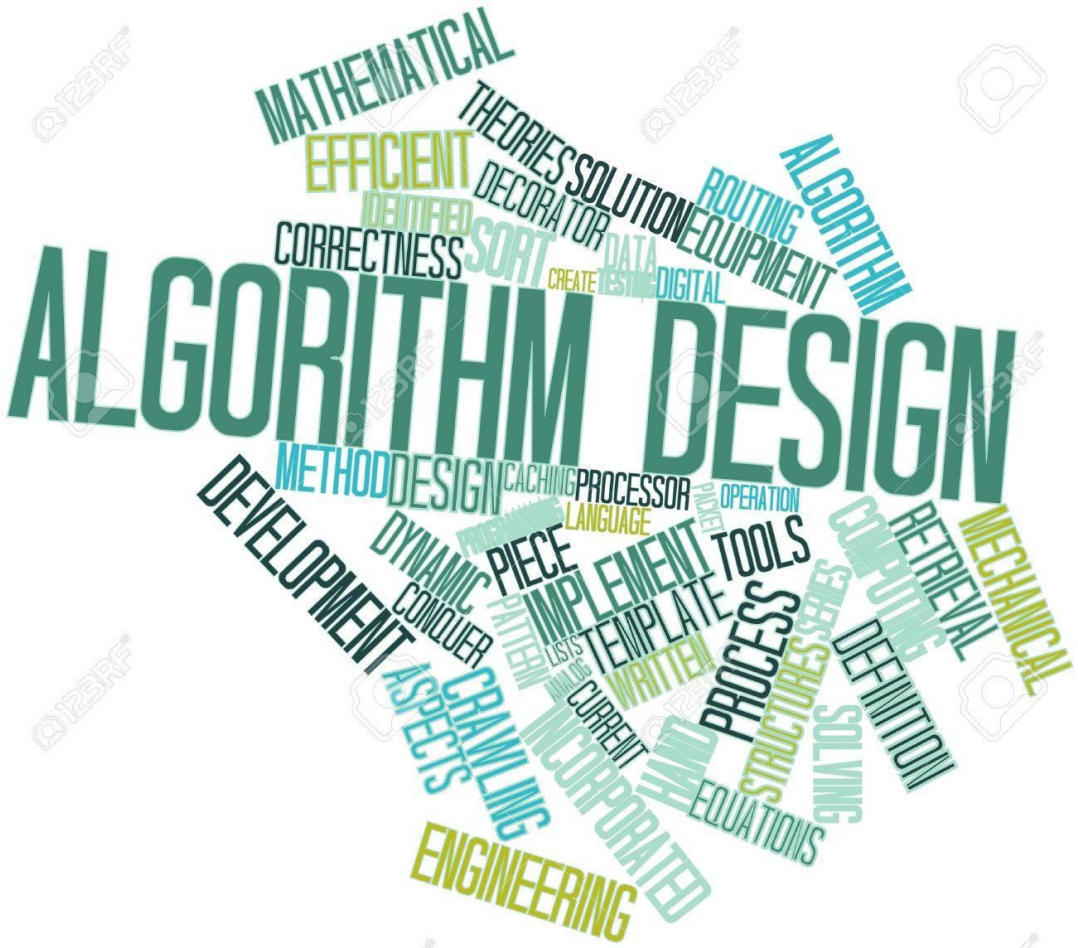
Python Algorithms

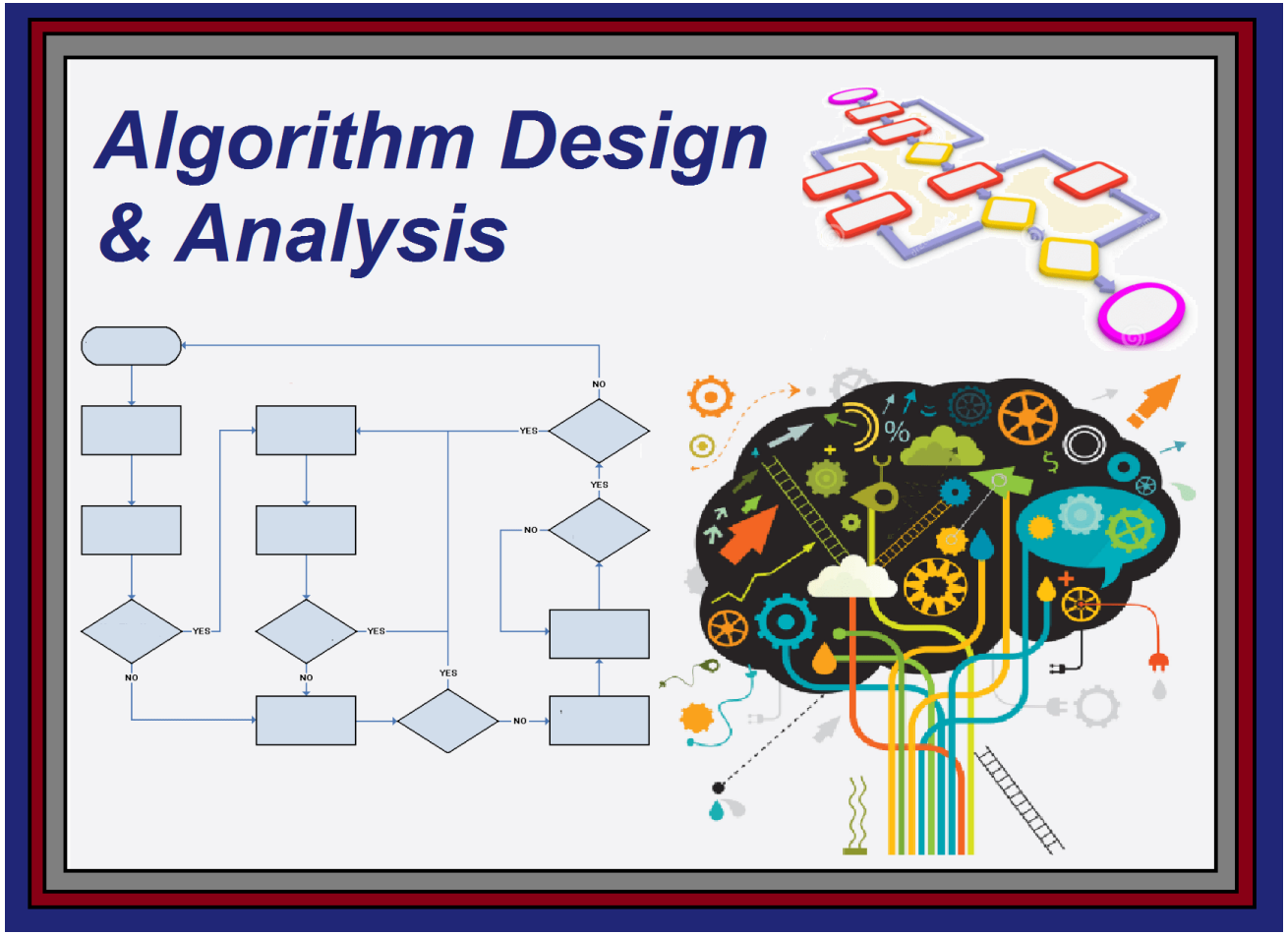
Design of Algorithm in Python Programming



Python Algorithms

Design of Algorithm in Python Programming





Python Algorithms

Algorithm 1 : What is algorithm that we do for the real purpose/problem

1: **Input**: What are ”data/information” you known from input

2: **Output**: What are ”results” you want to find from output

3: **Initialization**: The values of inputs for problem solving. Set $n := 0$

4: **Repeat**

5: Step 1: Doing something / checking / decision / analyzing.

6: Step 2: Doing something / checking / decision / analyzing.

7: Step ...

8: Step N: Doing something / checking / decision / analyzing.

9: Set $n := n + 1$

10: **Stop** Convergence: satifying the requirements. Collect the results as ”SOLUTION”.

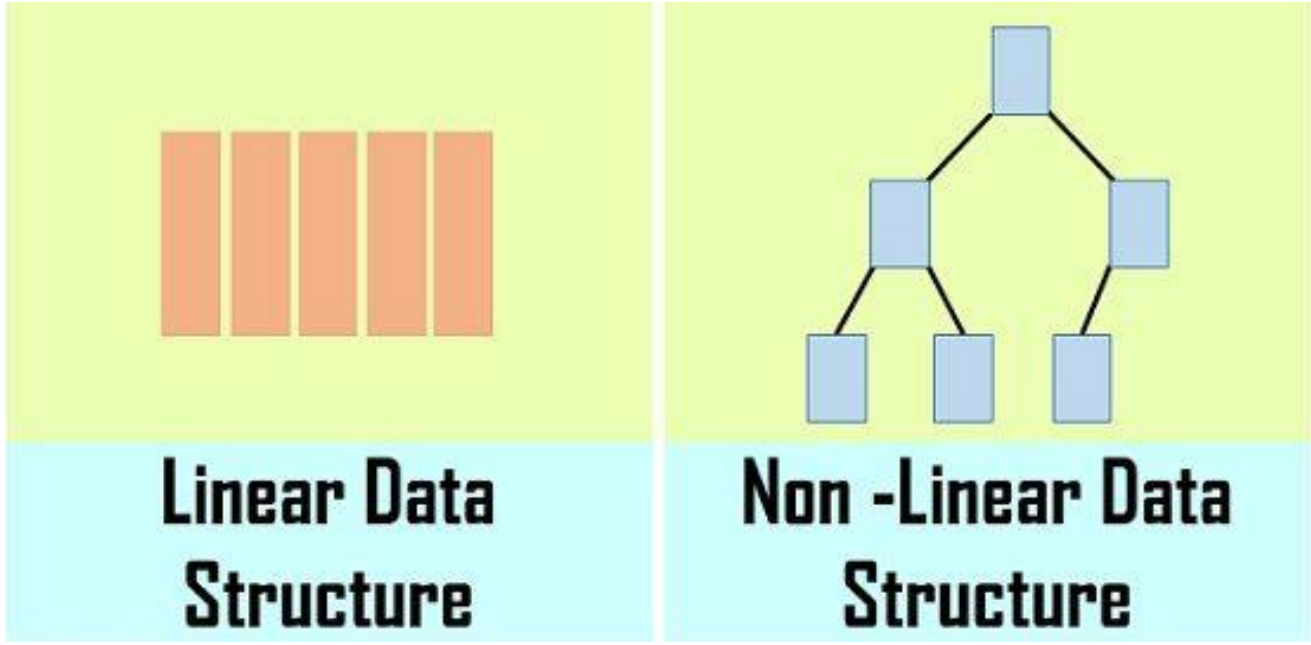
Python Algorithms

❑ Data Structure and Types:

- Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Types of Data Structure. Basically, data structures are divided into two categories:

- Linear data structure
- Non-linear data structure



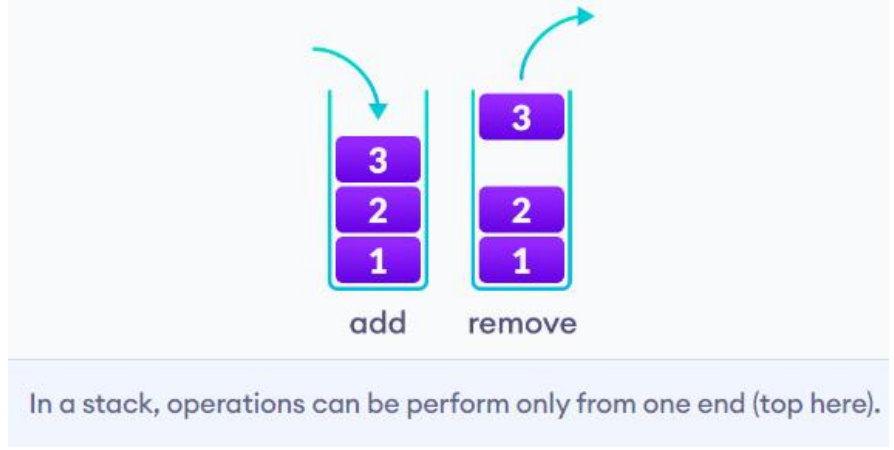
Python Algorithms

❑ Data Structure: Linear data structure

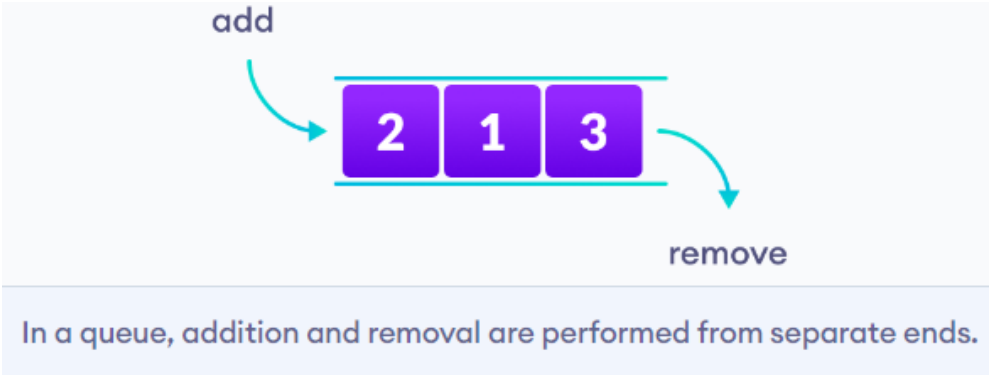
- In linear data structures, the elements are arranged in sequence one after the other. Since elements are arranged in particular order, they are easy to implement.

1. Array Data Structure
2. Stack Data Structure
3. Queue Data Structure
4. Linked List Data Structure

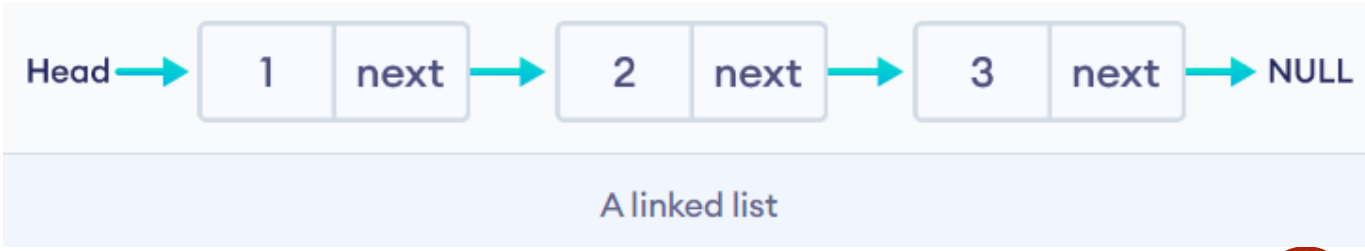
An array with each element represented by an index



In a stack, operations can be perform only from one end (top here).



In a queue, addition and removal are performed from separate ends.



A linked list

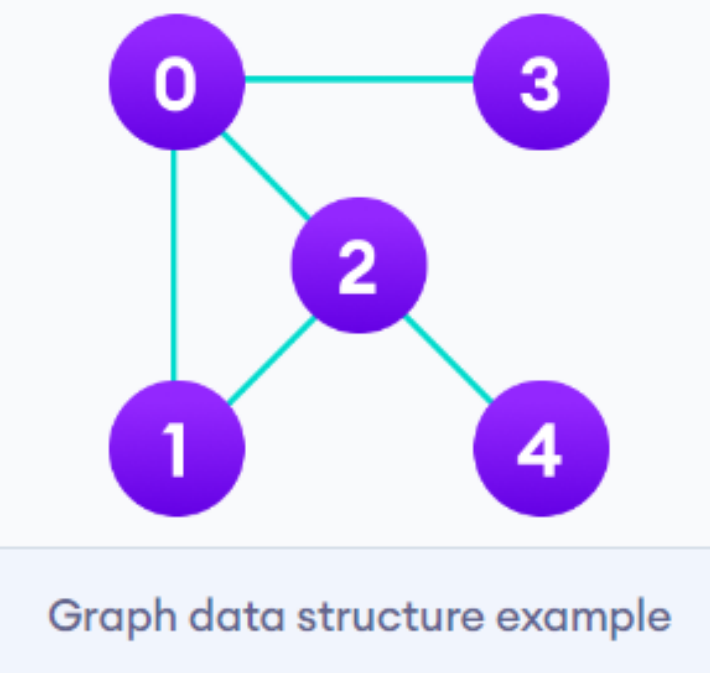
Python Algorithms

❑ Data Structure: Non linear data structures

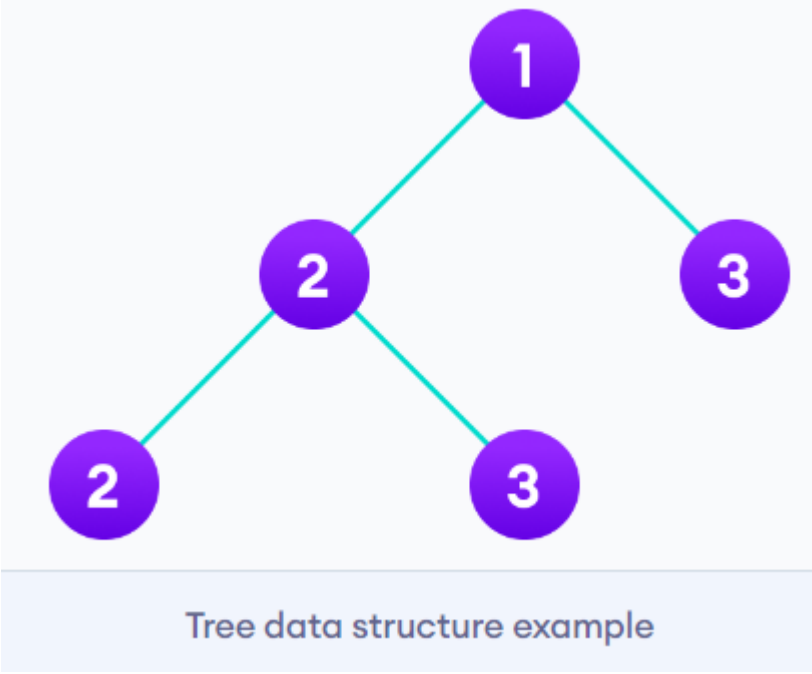
- Elements in non-linear data structures are not in any sequence. Instead they are arranged in a hierarchical manner where one element will be connected to one or more elements.

1. Graph Data Structure

2. Trees Data Structure



Graph data structure example



Tree data structure example

Python Algorithms

❑ Why Data Structures and Algorithms?

- Programming is all about data structures and algorithms. Data structures are used to hold data while algorithms are used to solve the problem using that data.
- Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

Python Algorithms

❑ Use of Data Structures and Algorithms to Make Your Code Scalable

- ***Time is precious***

Two of the most valuable resources for a computer program are time and memory.

```
Time to run code = number of instructions * time to execute each instruction
```

The number of instructions depends on the code you used, and the time taken to execute each code depends on your machine and compiler.

Example: solve a simple problem of finding the sum of the first 10^{11} natural numbers.

Let us assume that a computer can execute y = 108 instructions in one second (it can vary subject to machine configuration). The time taken to run above code is

```
Time to run y instructions = 1 second
Time to run 1 instruction = 1 / y seconds
Time to run x instructions = x * (1/y) seconds = x / y seconds

Hence,
Time to run the code = x / y
                     = (2 * 10^11 + 3) / 10^8 (greater than 33 minutes)
```

Python Algorithms

❑ Use of Data Structures and Algorithms to Make Your Code Scalable

- ***More on Scalability***

Scalability is scale plus ability, which means the quality of an algorithm/system to handle the problem of larger size.

Example:

Consider the problem of setting up a classroom of 5-8 students. One of the simplest solutions is to book a room, get a blackboard, a few chalks, and the problem is solved.

But what if the size of the problem increases? What if the number of students increased to 80 or 200?
→ The solution still holds but it needs more resources. → In this case, you will probably need programming/algorithm.

What if the number of students increased to 1000?

The solution fails or uses a lot of resources when the size of the problem increases. This means, your solution wasn't scalable.

Python Algorithms

❑ Use of Data Structures and Algorithms to Make Your Code Scalable

- ***Memory is expensive***

Memory is not always available in abundance. While dealing with code/system which requires you to store or produce a lot of data, it is critical for your algorithm to save the usage of memory wherever possible.

For example: While storing data about people, you can save memory by storing only their date of birth, not their age. You can always calculate it on the fly using their date of birth and current date.

Python Algorithms

❑ Experimental Studies:

We perform an analysis directly on a high-level description of the algorithm (either in the form of an actual code fragment, or language-independent pseudo-code). We define a set of primitive operations such as the following:

- Assigning an identifier to an object
- Determining the object associated with an identifier
- Performing an arithmetic operation (for example, adding two numbers)
- Comparing two numbers
- Accessing a single element of a Python list by index
- Calling a function (excluding operations executed within the function)
- Returning from a function.

Python Algorithms

- **Experimental Studies:**

An Algorithm, the processes share use of a computer's central processing unit (or CPU), the elapsed time will depend on what other processes are running on the computer when the test is performed.

```
from time import time
start_time = time( )                    # record the starting time
run algorithm
end_time = time( )                      # record the ending time
elapsed = end_time − start_time         # compute the elapsed time
```

A story of Computer programming

Once Upon an Algorithm

https://interestingengineering.com/science/15-of-the-most-important-algorithms-that-helped-define-mathematics-computing-and-physics

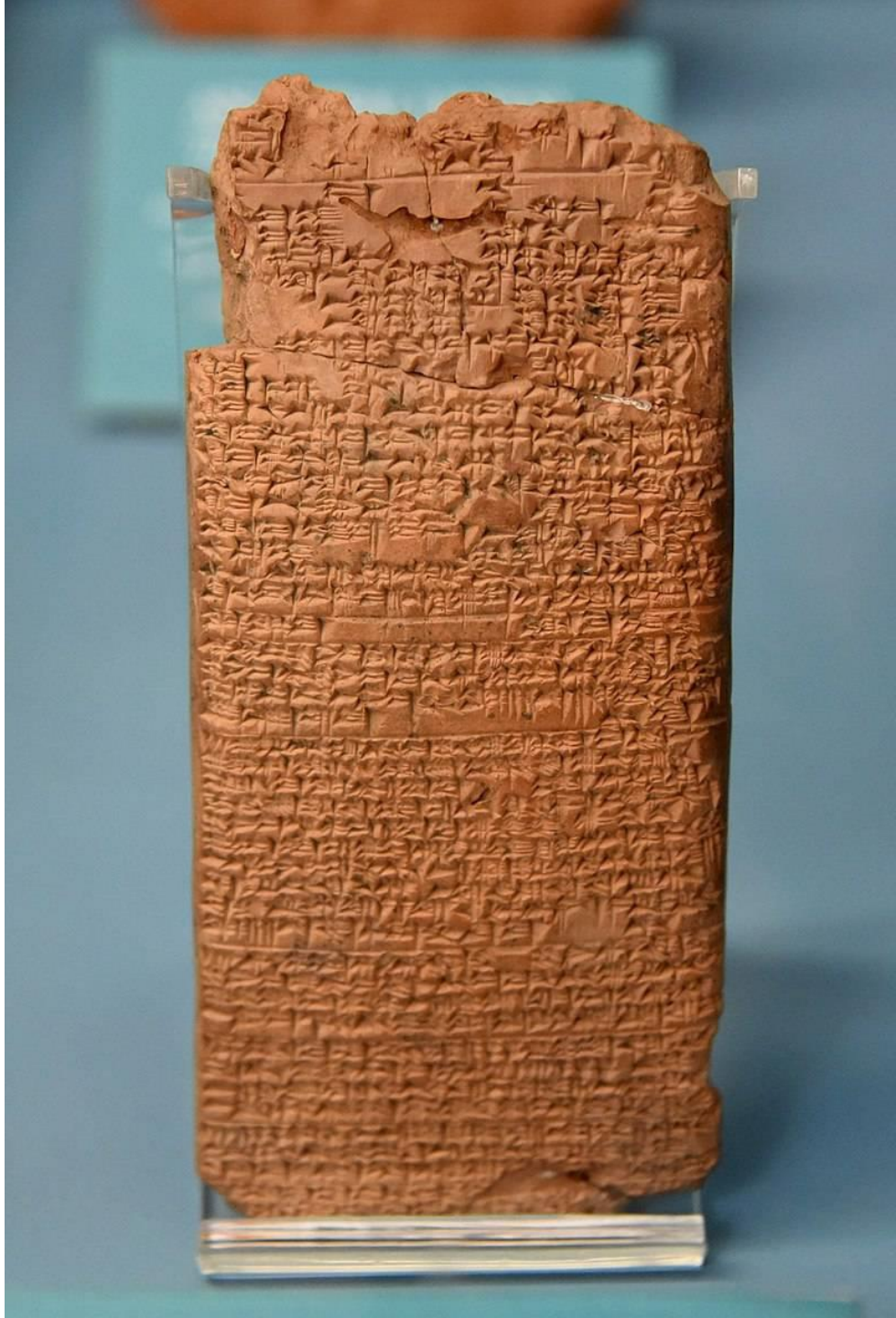
Once Upon an Algorithm

1. Babylonian algorithms are the oldest ever found

The world's first known algorithm

When it was created: 1600 BC

Các nhà toán học Babylon không chỉ giới hạn trong các quá trình cộng, trừ, nhân và chia; họ rất thành thạo trong việc giải nhiều loại phương trình đại số. Nhưng họ không có một ký hiệu đại số rõ ràng như của chúng ta; họ đã biểu diễn từng công thức bằng một danh sách từng bước các quy tắc để đánh giá nó, tức là bằng một thuật toán để tính toán công thức đó. Trên thực tế, họ đã làm việc với biểu diễn công thức 'ngôn ngữ máy' thay vì ngôn ngữ ký hiệu.



Once Upon an Algorithm

1. Babylonian algorithms are the oldest ever found

A (rectangular) cistern.
The height is 3,20, and a volume of 27,46,40 has been excavated.
The length exceeds the width by 50. (The object is to find the length and the width.)
You should take the reciprocal of the height, 3,20, obtaining 18.
Multiply this by the volume, 27,46,40, obtaining 8,20. (This is the length times the width; the problem has been reduced to finding x and y, given that $x - y = 50$ and $xy = 8,20$. A standard procedure for solving such equations, which occurs repeatedly in Babylonian manuscripts, is now used.)
Take half of 50 and square it, obtaining 10,25.
Add 8,20, and you get 8,30,25. (Remember that the radix point position always needs to be supplied. In this case, 50 stands for 5/6 and 8,20 stands for $8\frac{1}{3}$, taking into account the sizes of typical cisterns!)
The square root is 2,55.
Make two copies of this, adding (25) to the one and subtracting from the other.
You find that 3,20 (namely $3\frac{1}{3}$) is the length and 2,30 (namely $2\frac{1}{2}$) is the width.
This is the procedure.

Once Upon an Algorithm

2. Euclid's algorithm is still in use today

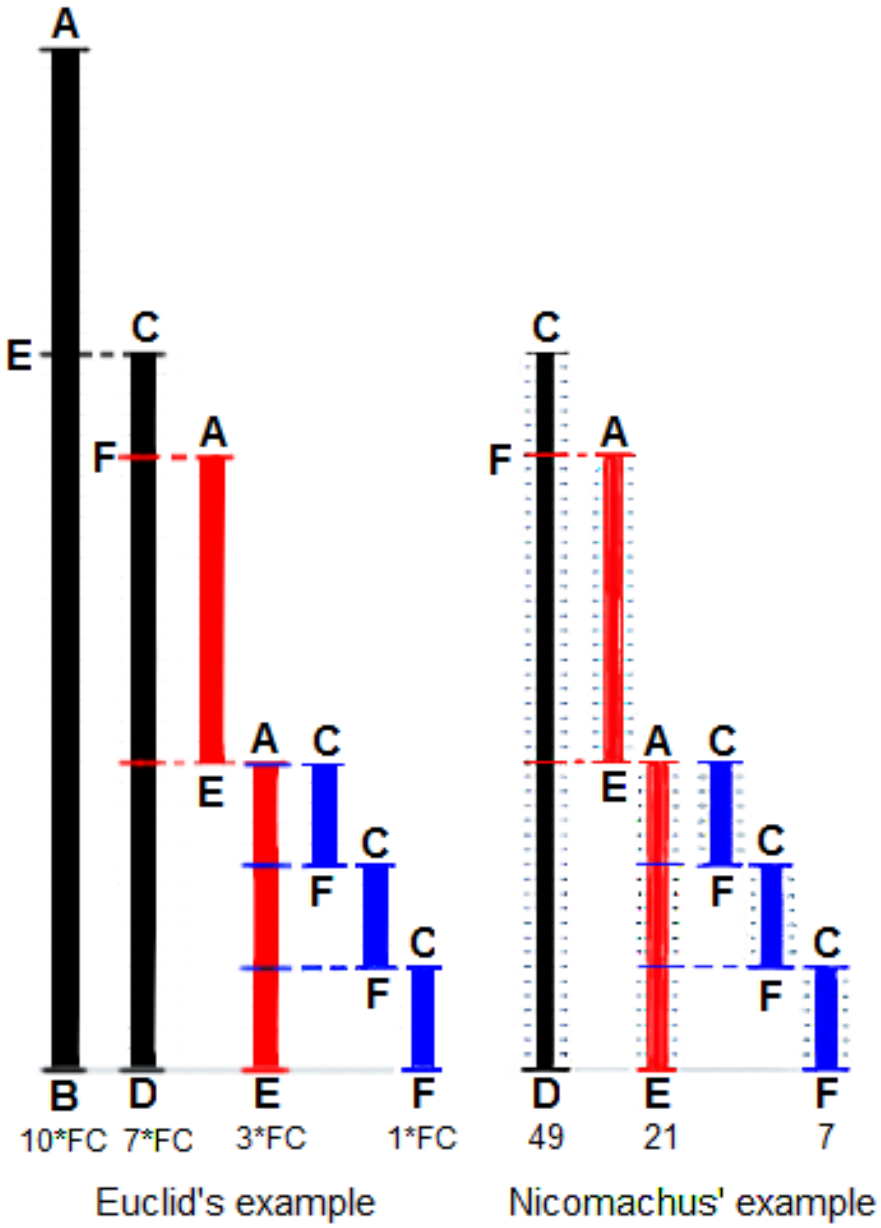
Creator: Euclid

When it was created: 300 BC

Euclid's algorithm is one of the earliest algorithms ever created and, with some alterations, is still used by computers today.

Thuật toán Euclide là một thủ tục được sử dụng để tìm ước chung lớn nhất (GCD) của hai số nguyên dương. Đó là một tính toán rất hiệu quả mà ngày nay máy tính vẫn sử dụng dưới dạng này hay dạng khác.

Thuật toán Euclid yêu cầu phép chia liên tiếp và tính các số dư cho đến khi đạt được kết quả.



Once Upon an Algorithm

2. Euclid's algorithm is still in use today

Ví dụ: GCD của 56 và 12 là gì?

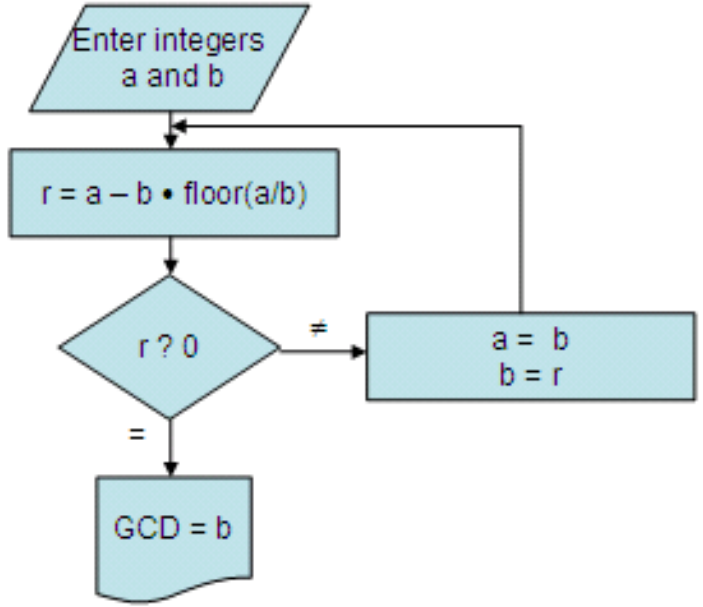
Bước 1 - Chia số lớn hơn cho số nhỏ hơn: 56/12 = 4 (dư 8)

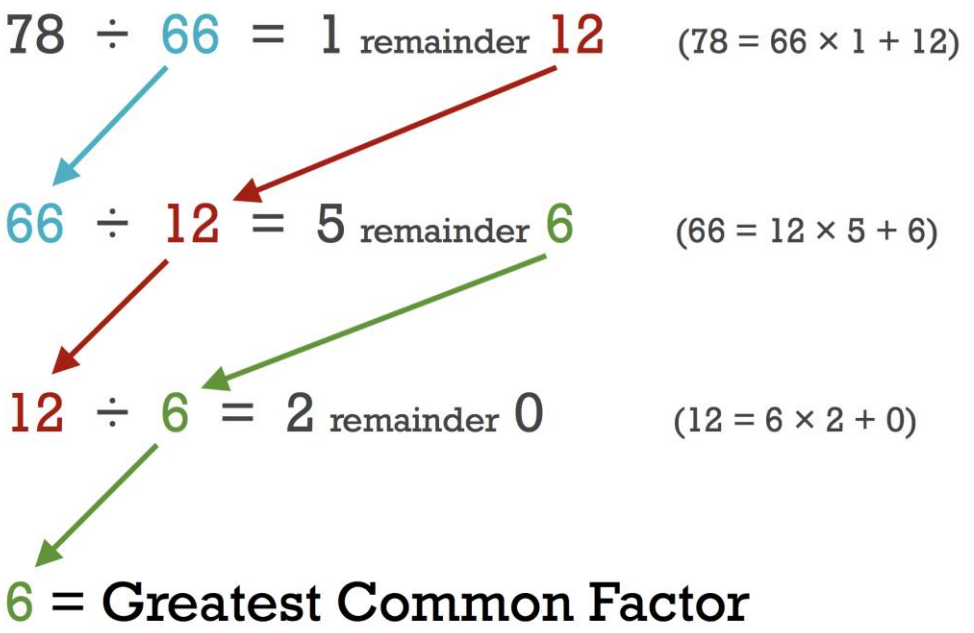
Bước 2 - Chia số chia cho số dư ở bước trước: 12/8 = 1 (dư 4)

Bước 3 - Tiếp tục bước 2 cho đến khi không còn phần dư nào (trong trường hợp này là quy trình 3 bước đơn giản): 8/4 = 2 (không dư)

Trong trường hợp này, GCD là 4.

Thuật toán này được sử dụng rộng rãi để rút gọn các phân số thông thường xuống các số hạng nhỏ nhất của chúng và trong các ứng dụng toán học cao cấp như tìm nghiệm nguyên cho phương trình tuyến tính.





Once Upon an Algorithm

3. The sieve of Eratosthenes is an ancient, simple algorithm

Creator: Eratosthenes

When it was created: 200 BC

The sieve of Eratosthenes is widely accepted as one of the most ancient algorithms of all time. It allows you to find all the prime numbers in a table of given numbers.

To perform it, you find all the numbers greater than 2, then cross out the ones divisible by 2. Then you do the same for non-crossed out numbers greater than 3, and so forth ad infinitum until all composite (non-prime) numbers are crossed out.

Để thực hiện, bạn tìm tất cả các số lớn hơn 2, sau đó gạch bỏ những số chia hết cho 2. Sau đó, bạn làm tương tự cho các số không bị gạch chéo lớn hơn 3, cứ tiếp tục như vậy cho đến khi tất cả hợp số (không phải số nguyên tố) số bị gạch bỏ.

Once Upon an Algorithm

3. The sieve of Eratosthenes is an ancient, simple algorithm

Once Upon an Algorithm

4. Boolean (binary) algebra was the foundation of the Information Age

Creator: George Boole

When it was created: 1847

Its impact/implications on the world: This algorithm is widely recognized as the foundation of modern computer coding. It is still in use today, especially in computer circuitry.

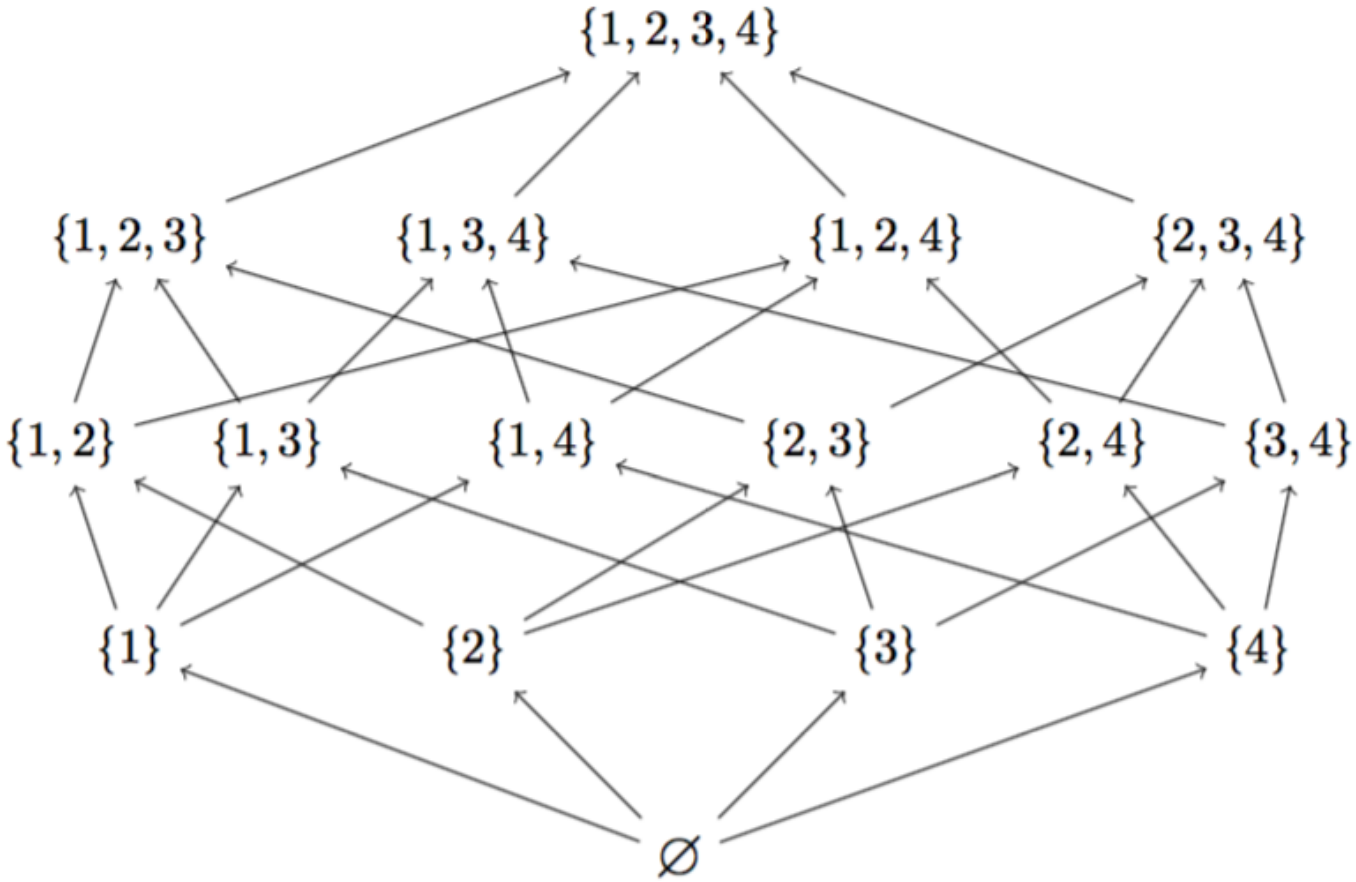
You might be familiar with the term Boolean from mathematics, logic, and computer coding.

Boolean algebra is a branch of algebra in which a variable can only ever be true or false - so-called truth values (usually binary 1 or 0).

Once Upon an Algorithm

4. Boolean (binary) algebra was the foundation of the Information Age

Creator: George Boole



Once Upon an Algorithm

5. Google's ranking algorithm (PageRank) could be the most widely used algorithm

Creator: Larry Page (mainly) and Sergey Brin

When it was created: 1996

Its impact/implications on the world: PageRank is, arguably, the most used algorithm in the world today. It is, of course, the foundation of the ranking of pages on Google's search engine.

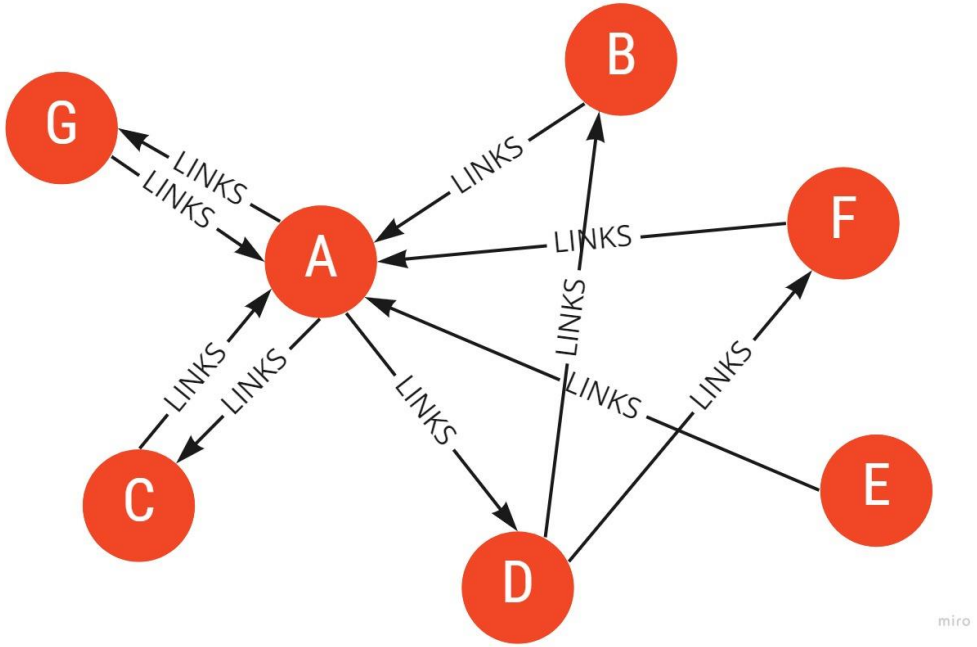
It is not the only algorithm that Google uses nowadays to order pages on its search result, but it is the oldest and best known of them.

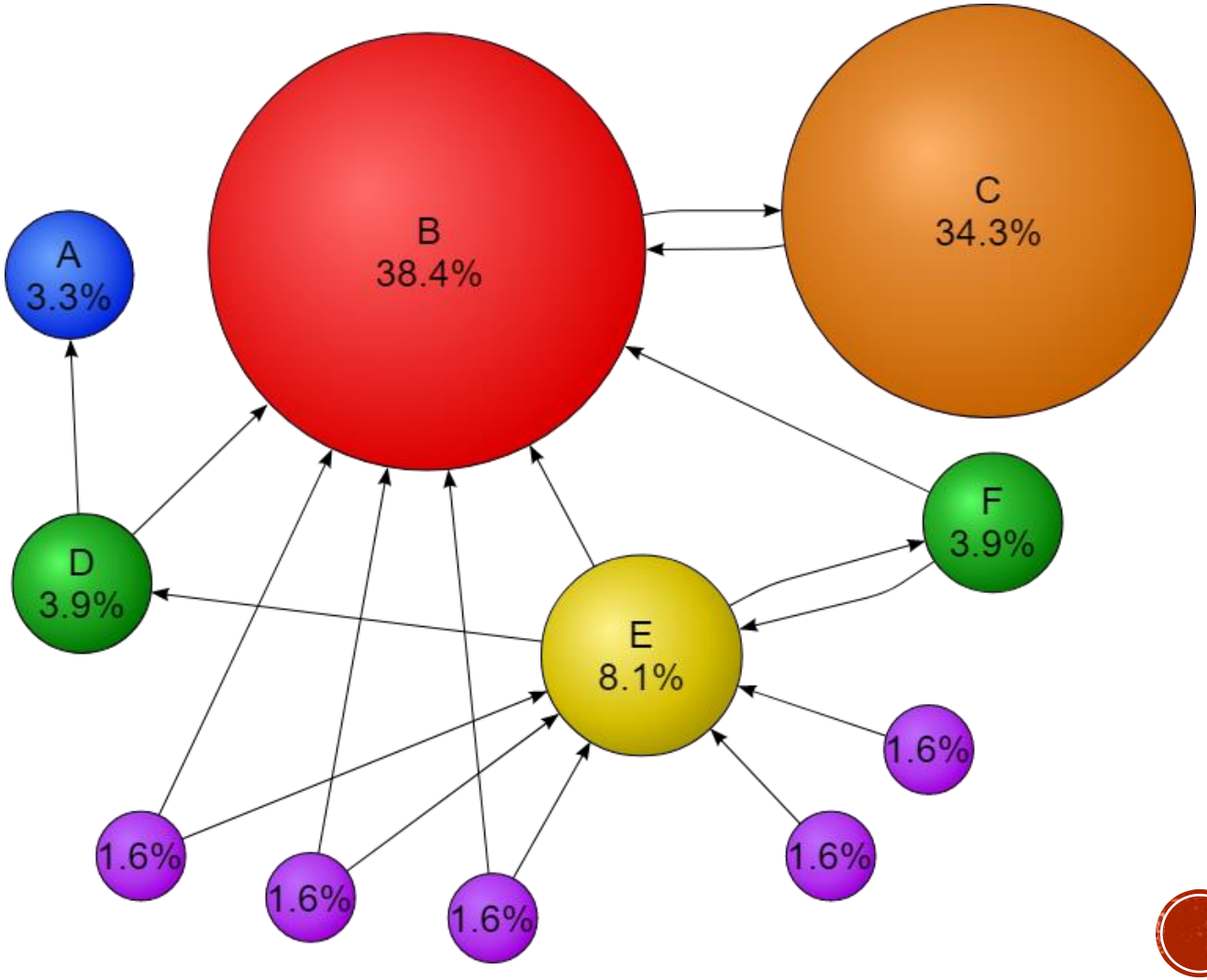
PageRank was devised by Larry Page and Sergey Brin while they were studying at Stanford University as part of their research project on a new kind of search engine.

The main premise was to rank pages based on their relative importance or popularity. Generally speaking, pages that appear higher in the hierarchy have more back-links or links to them.

Once Upon an Algorithm

5. Google's ranking algorithm (PageRank) could be the most widely used algorithm

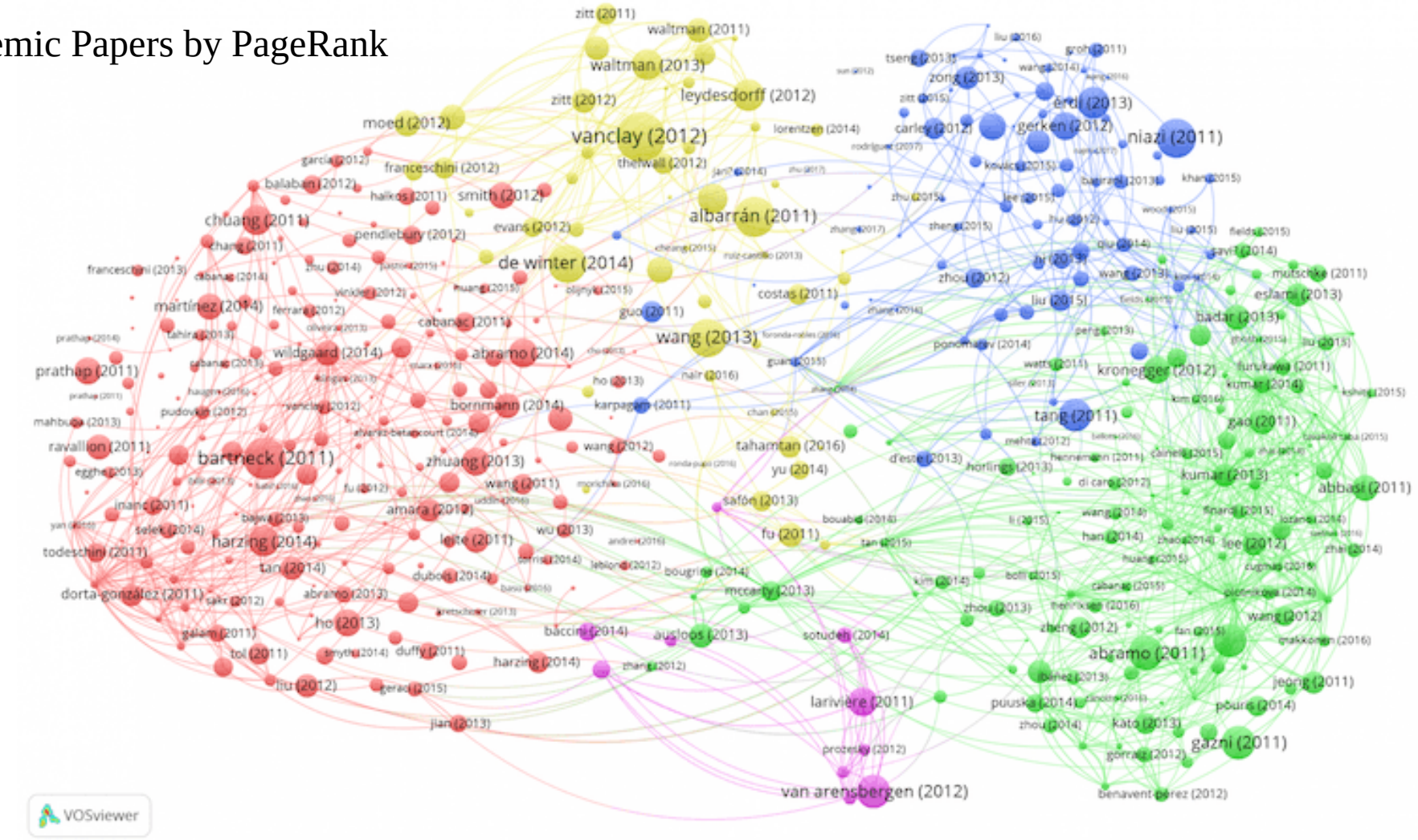




Once Upon an Algorithm

5. Google's ranking algorithm (PageRank) could be the most widely used algorithm

Evaluating Academic Papers by PageRank



Once Upon an Algorithm

5. Google's ranking algorithm (PageRank) could be the most widely used algorithm

Creator: Larry Page (mainly) and Sergey Brin

The **PageRank algorithm** is given by the following formula:

$$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

where:

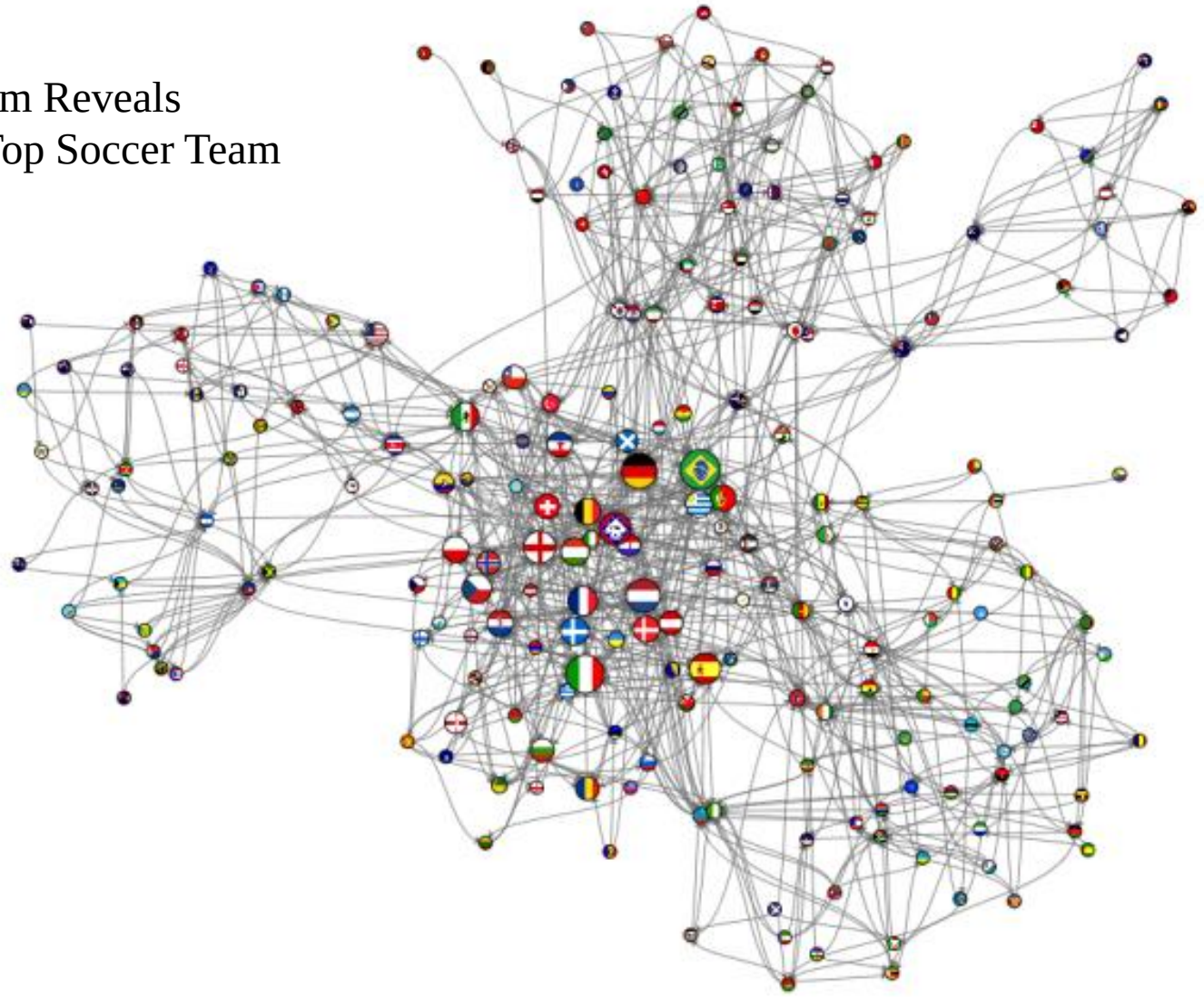
- PR(A) is the PageRank of page A,
- PR(Ti) is the PageRank of pages Ti which link to page A,
- C(Ti) is the number of outbound links on page Ti and;
- d is a damping factor which can be set between 0 and 1.

It can also be likened to a “pyramid scheme in which a particular page is ranked recursively depending on what other pages link to it".

Once Upon an Algorithm

5. Google's ranking algorithm (PageRank) could be the most widely used algorithm

PageRank Algorithm Reveals
World's All-Time Top Soccer Team

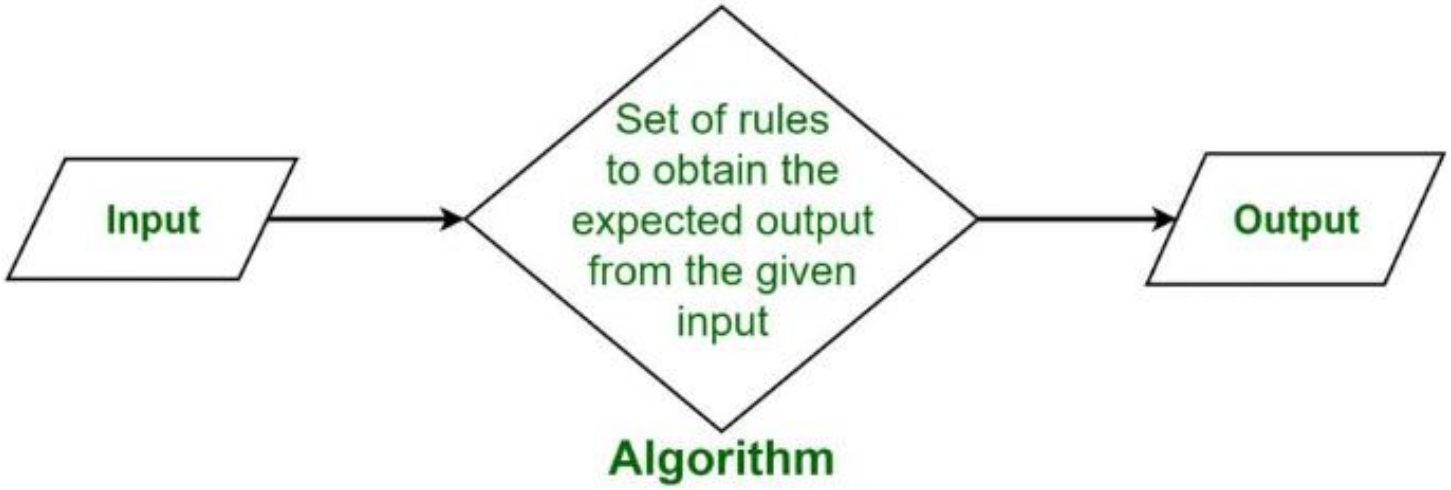


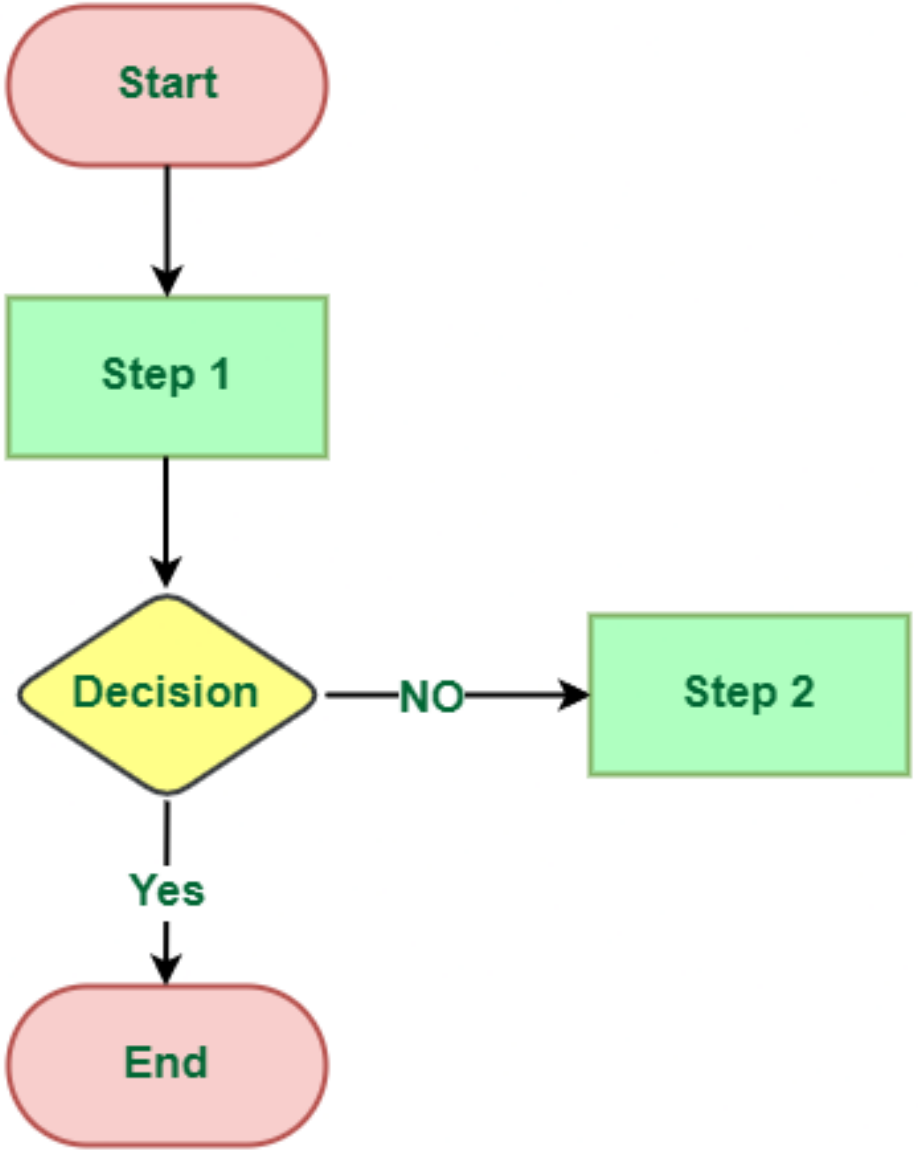
Practice

An Application of Algorithm

Practices: Algorithms

Design of Algorithm in Python Programming





Practices 02: Knapsack Algorithms

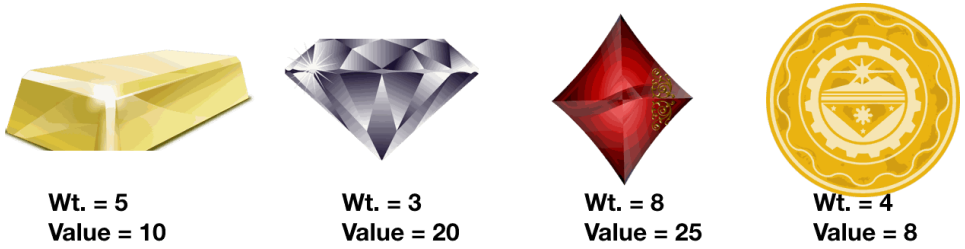
❑ Knapsack problem:

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision-makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

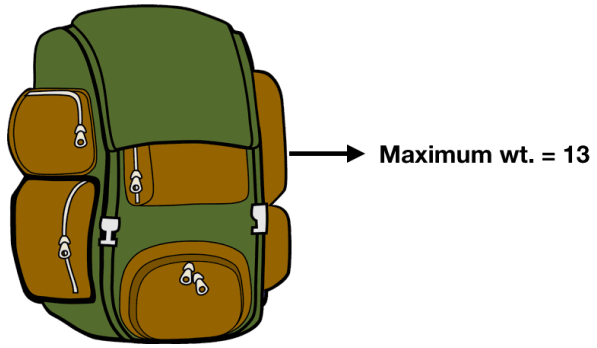
Bài toán chiếc ba lô là một bài toán trong tối ưu hóa tổ hợp: Cho trước một tập hợp các vật phẩm, mỗi vật phẩm có **trọng lượng** và **giá trị**, xác định **số lượng** của từng vật phẩm cần đưa vào **bộ sưu tập** sao cho **tổng trọng lượng nhỏ hơn hoặc bằng một giới hạn** đã cho và **tổng giá trị càng lớn càng tốt**.
Nó bắt nguồn từ tên của nó từ vấn đề mà một người phải đối mặt với một chiếc ba lô có kích thước cố định và phải lấp đầy nó bằng những món đồ có giá trị nhất.

Vấn đề thường phát sinh trong việc phân bổ nguồn lực, trong đó những người ra quyết định phải lựa chọn từ một tập hợp các dự án hoặc nhiệm vụ không thể chia nhỏ theo một ngân sách cố định hoặc hạn chế về thời gian, tương ứng.

Practices 02: Knapsack Algorithms



- **Knapsack problem:**



The most common problem being solved is the **0-1 knapsack problem**, which restricts the number x_i of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W,

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1\}.$$

Here x_i represents the number of instances of item i to include in the knapsack. Informally, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity.

Practices: Algorithms

- **Knapsack problem: Integer programming (IP)**

Classic Knapsack Problem:

You want to maximize the value of items you can pack into a single suitcase (or knapsack). However, you are limited to a weight of 50 lbs.

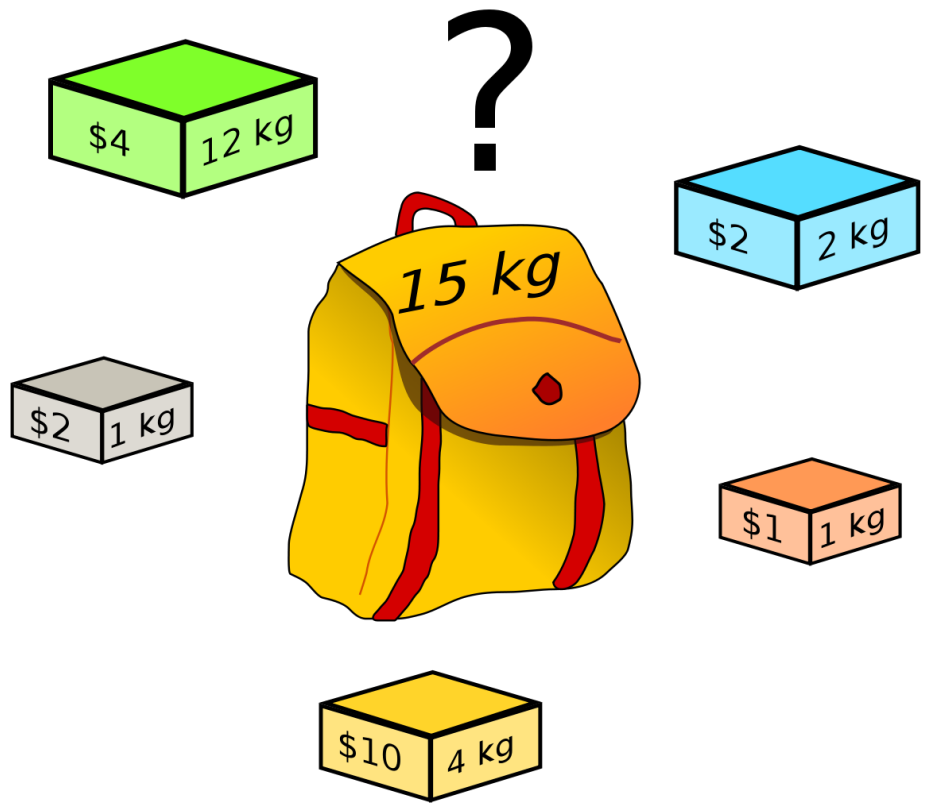
Item	Value ($)	Weight (lbs)
A	15	18
B	20	10
C	18	21
D	13	11
E	12	11

$x_i = 0$ if not chosen
$= 1$ if put in knapsack

$$Max \quad 15x_A + 20x_B + 18x_C + 13x_D + 12x_E$$

$$s.t. \quad 18x_A + 10x_B + 21x_C + 11x_D + 11x_E \leq 50$$

$$x_A, x_B, x_C, x_D, x_E = 0 \text{ or } 1$$



Practices: Algorithms

- **Knapsack problem**

Item	Value ($)	Weight (lbs)
A	15	18
B	20	10
C	18	21
D	13	11
E	12	11

Solution:

Pick up	Total Value	Total weight
A, B, C	53	49
A, B, D	48	39
A, B, E	47	39
B, C, D	51	42
C, D, E	**43**	43
A, B, D, E	**58**	50

Max-Min = $15

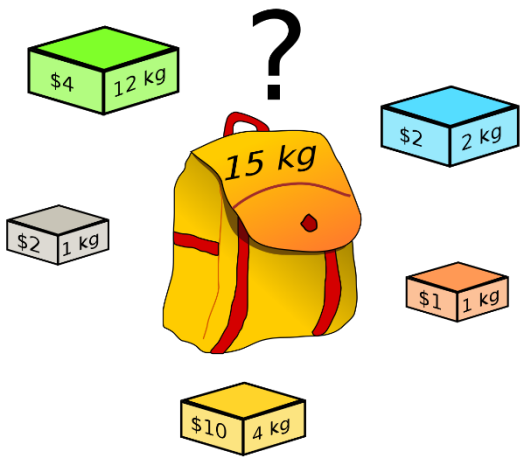
One person with $15

Ten person with $150

Long Thanh airport:
100 million passengers/year
→ 1.5 billion USD

4.3 billion passengers carried in 2018
→ 2 billion person with 30 billion USD

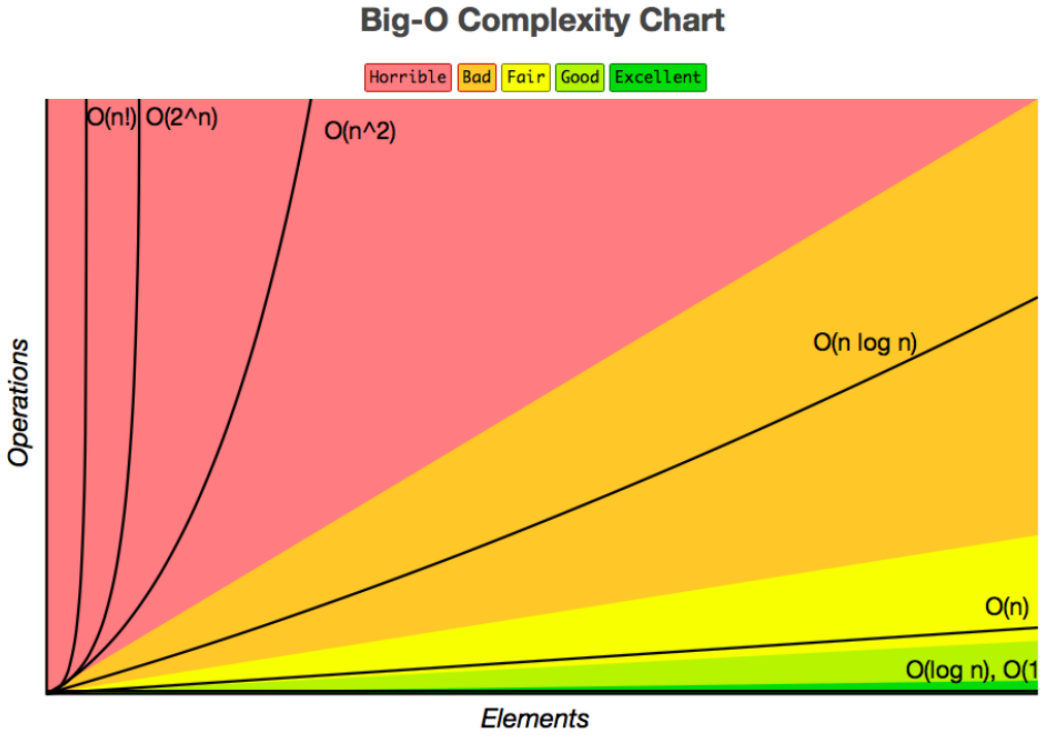
Practices: Algorithms

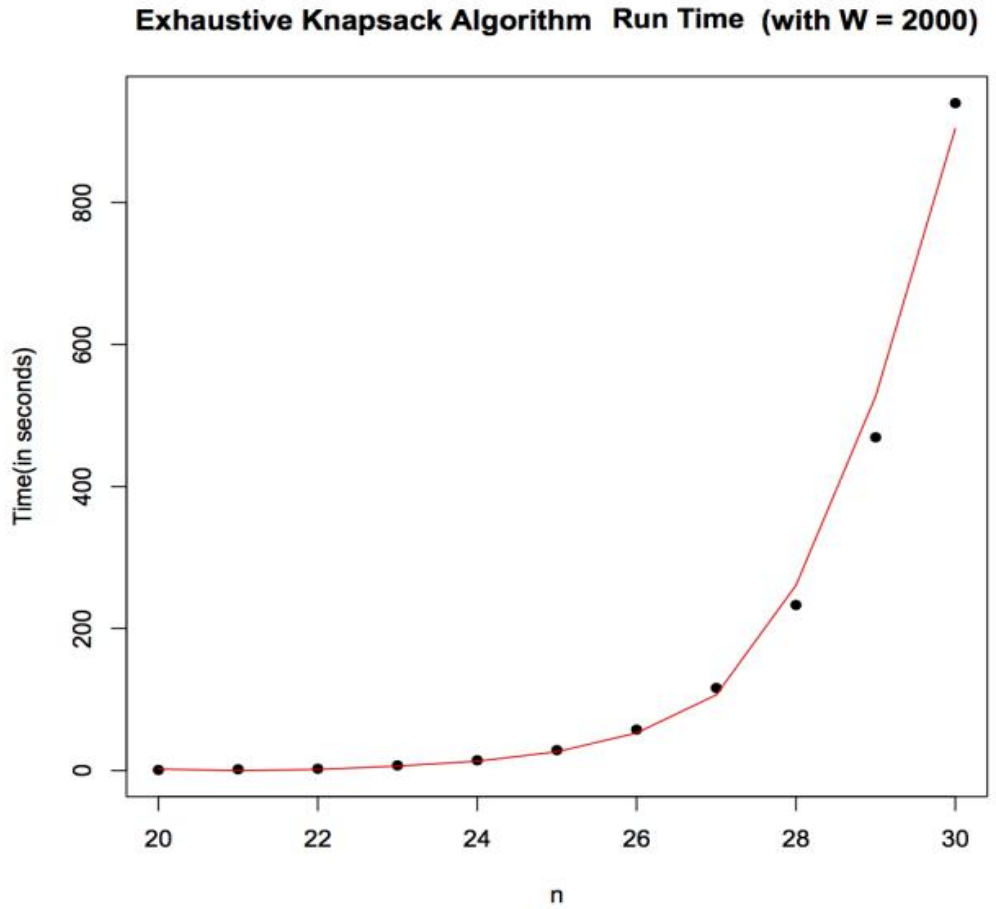


❑ 0/1 Knapsack Problem:

The time complexity of 0/1 Knapsack is **O(2^n)**,

n: number of variables



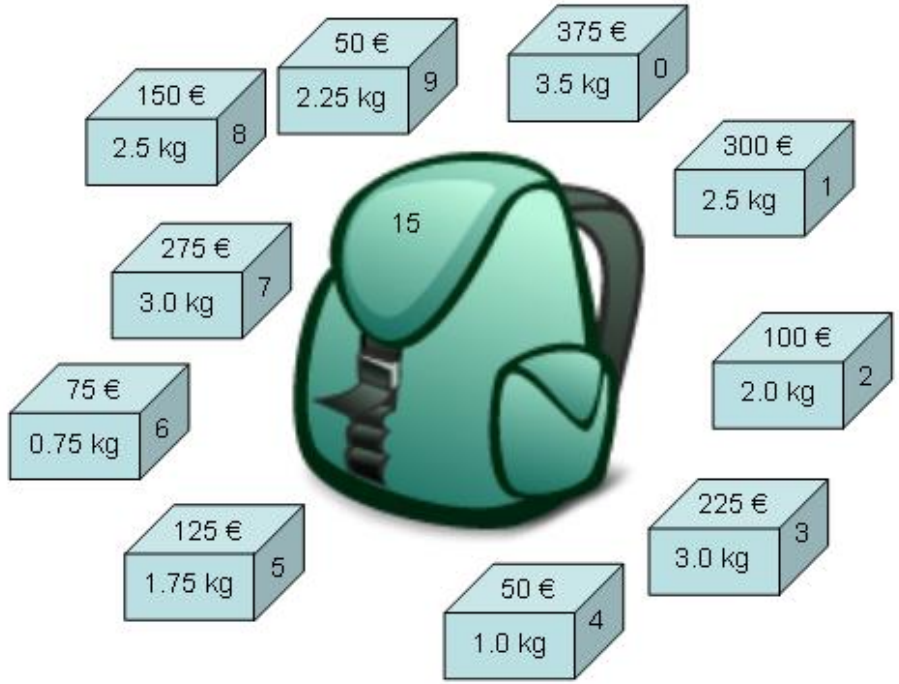


Practices 02

- **Knapsack problem:**

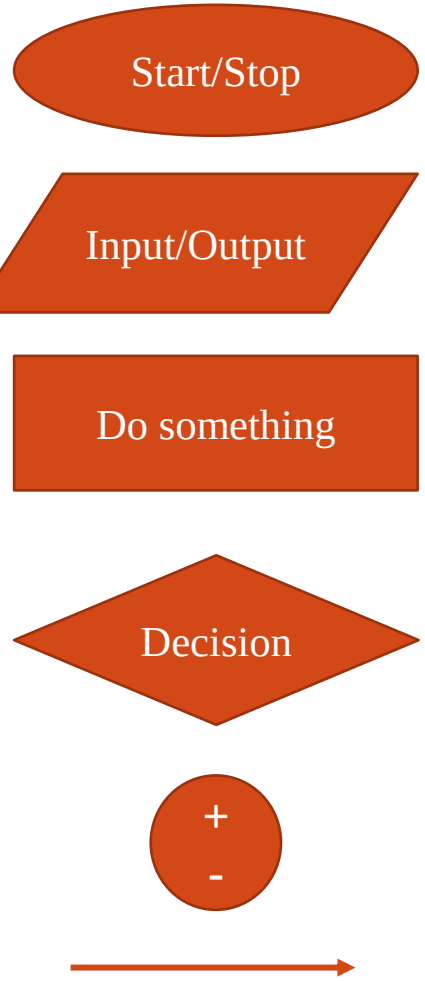
Giới hạn trọng lượng Knapsack: 23kg

Vật phẩm	Giá trị (vnđ)	Trọng lượng (kg)
A	250,000	6
B	900,000	5
C	700,000	7
D	650,000	9
E	1,000,000	9
F	850,000	2
G	550,000	4
H	1,250,000	8
I	750,000	6
J	450,000	3



Practices 02

- Design an Algorithm (steps) để giải quyết Knapsack problem.
- Giải quyết vấn đề → tìm ra solution (bằng tay)
- Viết code lập trình giải thuật: (đánh giá theo từng mức)
  - Function chứa giải thuật (Knapsack algorithm) cho bài toán Knapsack
  - Thể hiện rõ ràng các bước thực hiện giải thuật đề xuất
  - Code có thể mở rộng với bất kỳ giá trị “input” phù hợp
  - Testbed và tính toán thời gian thực thi giải thuật với các “input size” khác nhau



Practices: Algorithms

Design of an Algorithm in Python Programming

Algorithm 1 : What is algorithm that we do for the real purpose/problem

1: **Input**: What are ”data/information” you known from input
2: **Output**: What are ”results” you want to find from output
3: **Initialization**: The values of inputs for problem solving. Set $n := 0$
4: **Repeat**
5: Step 1: Doing something / checking / decision / analyzing.
6: Step 2: Doing something / checking / decision / analyzing.
7: Step ...
8: Step N: Doing something / checking / decision / analyzing.
9: Set $n := n + 1$
10: **Stop** Convergence: satifying the requirements. Collect the results as ”SOLUTION”.