**Algorithm**

# Definition

Algorithm is a set of steps or instruction for performing a task.

We usually have many different algorithm for solving one problem, so for current problem it good to know the best one.

# Why?

We need to know algorithm for three mains raisons:

* There are some currents problems in programing and some of them have efficient way of solving them, we need to know these solutions (algorithm) to use them when we face such problem. (the goal is to know what work better for a specific context)
* To have an algorithm thinking, which is the capacity to understand a problem, to split it into small steps and to know which algorithm to use for solving each step.
* It’s important for jobs interviews

# Characteristics

## Algorithms guideline

Algorithm definition

* Is firstly define the problem that it solved, because we cannot split problem into steps if we do not have a clear idea of what we want to solve. So in defining the problem we need to know how the input is defined and how the output is look like when the algorithm has done it job. (A clearly defined problem statement, input and output)
* Must contains a specific set of instruction in particular order. Each task should be a distinct one (Can be split into subtasks)
* The algorithm should produce a result (so we know If it ‘working or not)
* It should complete and cannot take infinite amount of time

## Evaluate an algorithm

Is difficult say that an algorithm is the right solution for a specific problem, because is all about context. So here, we talk about two context: the correctness and the efficiency of an algorithm.

## Correctness

Correctness is proved by mathematics induction, but to make it simple and algorithm is correct if these two conditions are true.

* For a specific problem, on every run of the algorithm against all possible value in input data we always get the output we expect
* For any possible input the algorithm always terminates, or ends

## Efficiency

We introduce here time and space complexity.

We use running time of algorithm to measure it’s time complexity

## Big O

### Polynomial runtime

Big O is notation used to describe complexity.

* **Constant time (O(1)) :** No matter the input the runtime is still the same
* **Linear time (O(n)) :** The run time increases as the inputs increase
* **Logarithmic time (O(ln n) or O(log n)) :** the runtime increase just a little bit when the input increases
* **Quadric runtime (O(n2)) :** For ‘n’ number of input the algorithm execute n2 operations
* **Cubic runtime (O(n3)) :** For ‘n’ number of input the algorithm execute n3 operations these two are too expensive computationally, as we can see for a small changes ‘n’ there is significant change in the number of operations
* **Quasilinear runtimes (O(n log n)) :** So for every value of n we can have (n\*(log n)) operations it has a runtime which is between the quadric runtimes and the linear runtime(common to merge algorithm)

The algorithms are consider to have Polynomial runtime because for ‘n’ number of value, they worst case can be write has O(nk). These algorithms are consider being efficient and are likely used in practice.

### Exponential runtimes

When ‘n’ increases the number of operations increases exponentially, they are too expensive. With these algorithm the big O notation of some number raised to the nth power

* **The brute force algorithm (O(mn))** : where ‘m ’ is the possible number of value a character can have
* **Factorial/Combinatorial (O(n!))runtime** (eg: Traveling Salesman)

## 

## Search algorithms

## Linear search (sequential or simple search)

## Binary Search

Input must be a sorted list

# RECURSION AND SPACE COMPLEXITY

## Definitions

* Recursive function is that call itself in opposite of iterative functions
* Recursive depth is the number of time the function call it self

Functional language does not like to change the variables value during the function execution, reason why they prefer recursion.

**Note:** Space complexity of Recursive functions depends on programming language.

## Space complexity

Space complexity is a measure of how much working storage or **extra storage** is needed as a particular algorithm grows.

**Note:** We need to think about it when writing algorithms, because we want to design our algorithms to perform as efficiently as it can as the size on the data set grows

**Data Structures**

**Directive question:** Why do we need more data than programming languages offer?

# DEFINITION

Data structure is a data storage format. It is the collection of values and the format they are stored in, the relationships between the values in the collection as well as the operations applied on the date stored in the structure.

# Array

It’s a contiguous data structure, this means that the array is stored in in blocks memory that are right beside each other with no gaps, which make retrieving values very easy. In non-contiguous data structure, value as stored has reference to where the next value is. So to retrieve that next value the language as to follow that reference (pointer) to the next block of memory (which increases the runtime of common operations)

## Operations

There 4 basic operation for data structure:

* Access and read values
* Search for an arbitrary values
* Insert values at any point into the structure
* Delete values in the structure

## Access

As array is a contiguous data structure, when we create an array, the language memories only where the index 0 element is store (Bases storage). Therefore, when we try to access the value of an element inside the array, it is only calculate the distance from the base storage. This is the reason we get error when we try to access an index which is big than the array length, because we access memory space that is not allocate to the array.

## Search

In general, Arrays are not good for searching because they usually use linear search.

## Inserting

In general, most array implementations support three type of insert operations:

* The true insert. That insert an element at an index of the array. When it done, all old element has to shift from one right space in the memory. So it’s a linear runtimes operations
* Append. That add an element at the end of the list. Therefore, it is a constant runtime operation.
* Extend. The ability to add one list to another. It has a runtimes of ‘k’, in which ‘k’ is the number of element we want to add.

## Delete

As a linear runtime as when you delete a element from array, every other element on it’s right have to shift to left

# LINKED LIST

Why do we have to build a linked list?

Every data structure solve a particular problem. Linked List can be better than array in deleting and inserting data.

It is a linear data structure where each element in the list is contained in a separate object called a node. A node contains the data we want to store and the reference to the next node.

The first node is the **head**; it is the only node that the list maintains.

The last node is the **tail**, it is not pointing to anything.