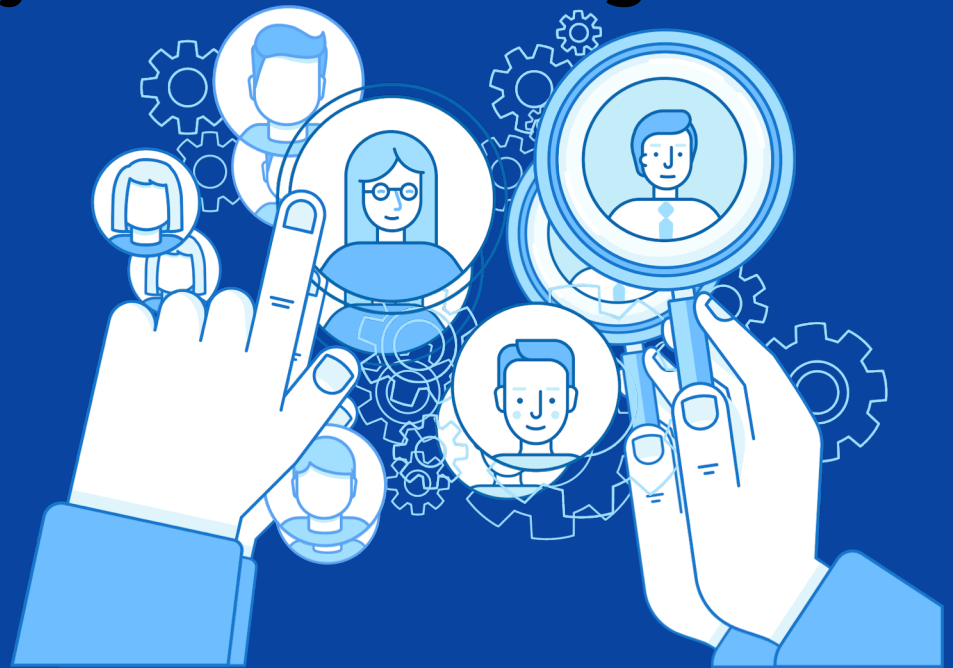
Digital Career Institute

Python Course - Algorithmic Thinking



Digital Career Institute

Goal of the Submodule

The goal of this submodule is to help the learners understand the concepts of algorithms, how to think in terms of algorithms, and what to watch out for when choosing a particular algorithm for a given task.

Topics

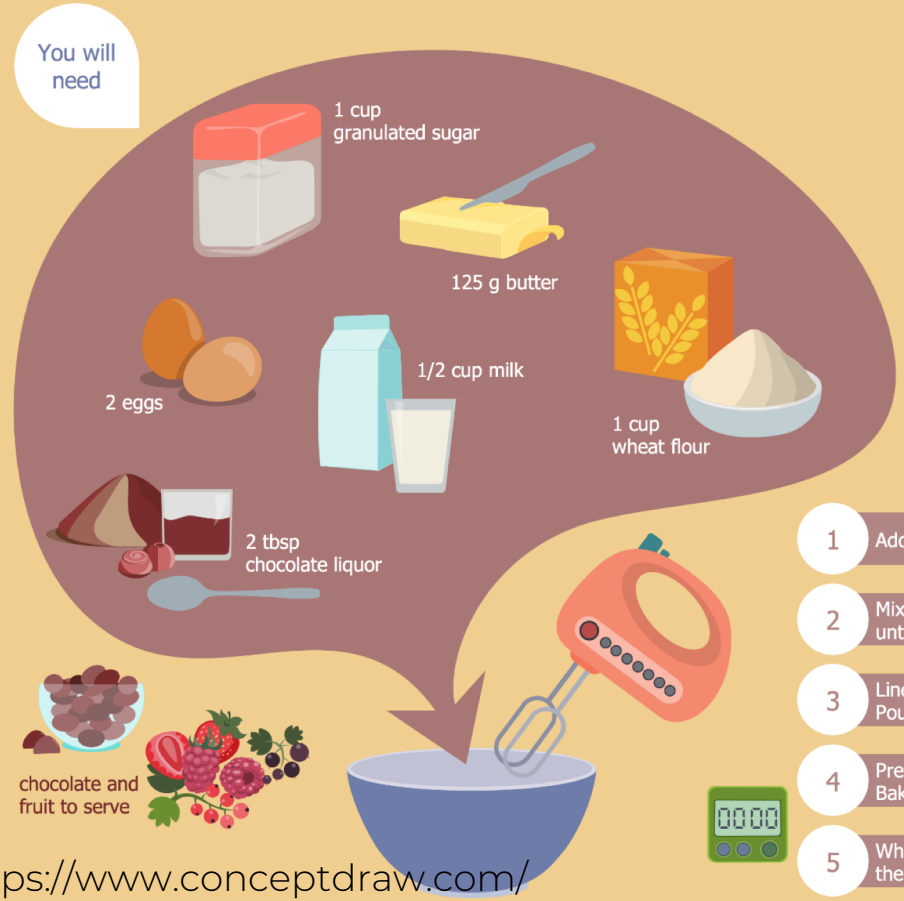
- Algorithms
- Algorithmic thinking in general and in computer programming
- Analyzing speed and memory usage of algorithms
- Quicksort
- Mergesort
- O-notation

Digital Career Institute

Algorithms and Algorithmic Thinking

Digital Career Institute

CHOCOLATE CAKE



Method

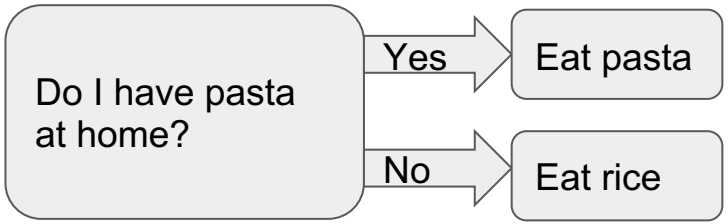
1. Add all the ingredients in a medium sized bowl.
2. Mix with an electric mixer for about 2-3 minutes or until the mixture is smooth.
3. Line a 20 cm round cake tin with baking paper. Pour the cake mixture into the tin.
4. Preheat the oven to 180 degrees C. Bake for approximately 45 minutes.
5. When you remove the cake from oven, let it cool in the pan for about 5 minutes. Leave to cool on a rack.

Source: https://www.conceptdraw.com/

Algorithms - What Is an Algorithm

- A step-by-step guide for the computer to be followed when in a particular situation.
- Can be used for decision making processes.
  - Example: What stories to show to a particular user of a social network.
- Can be used as instructions to go from the current state of things to a desired result.
- Humans use algorithms in their lives in order to make decisions or act in a specific way to achieve a predetermined result:

What should I eat?



Decision making

I want to eat rice. How do I prepare it?

Reach specific result

Algorithmic Thinking

Digital Career Institute

- Splitting a task into single step instructions.
- Make precise instructions for how to arrive at a decision.
- Make a process replicable by following instructions.

Easier to turn into algorithm	**Harder to turn into algorithm**

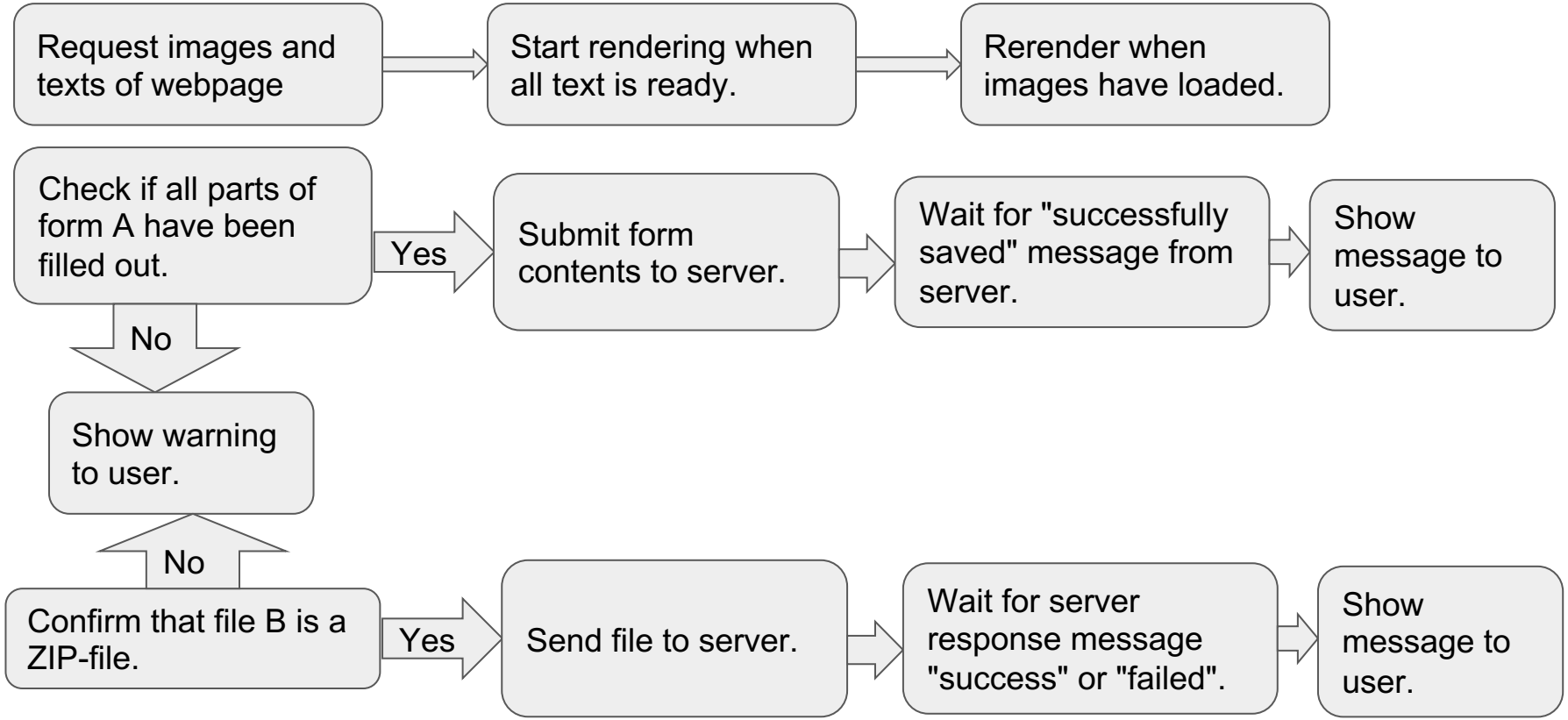
● How does one cook instant noodles?
● How should one drive in the UK?
● How does one enter the US/EU?
  ○ Which visa should one choose?
  ○ What order should one follow?
● Should you eat pasta today?
  ○ Do you have pasta available?
  ○ Have you eaten pasta the past 7 days in a row?
	● How can you become a world class famous chef?
	● Which clothing should you wear today?
	● How does one become a millionaire on the stock market?
	● What is the meaning of life?
	● Which user interface alternative of a given app will users prefer?

Algorithms in Computer Programming

Digital Career Institute

- A snippet of a computer program, telling the program how to behave in a specific situation.
- Possible situations:
  - User has entered the page.
  - User clicked a button.
  - User has started to scroll.
  - User has been inactive for X seconds.
  - User has uploaded file Y.
  - The credit card number the user has provided has been denied by the issuing bank.

Typical Programming Algorithms



Algorithms in Computer Programming

Digital Career Institute

- Very often, the operations required in the code are very common, some other times these operations are rarer.
- Operations of lower level (doing a single well-defined and specific thing) tend to be very common operations. Whereas higher level operations (implementing our business logic) tend to be specific to our application.
- Common low-level operations and algorithms have been extensively analysed and one or more implementations can be found in almost any programming language.
- Examples of common operations:
  - Sorting a list of items.
  - Searching for an item in a sequence.
  - Comparing two items.
  - Hashing an item.
- When an algorithm is **generic** and there are **different implementations** available, they tend to come with **advantages and trade-offs**, often involving a difference in performance and quality of the output.

At the Core of the Lesson

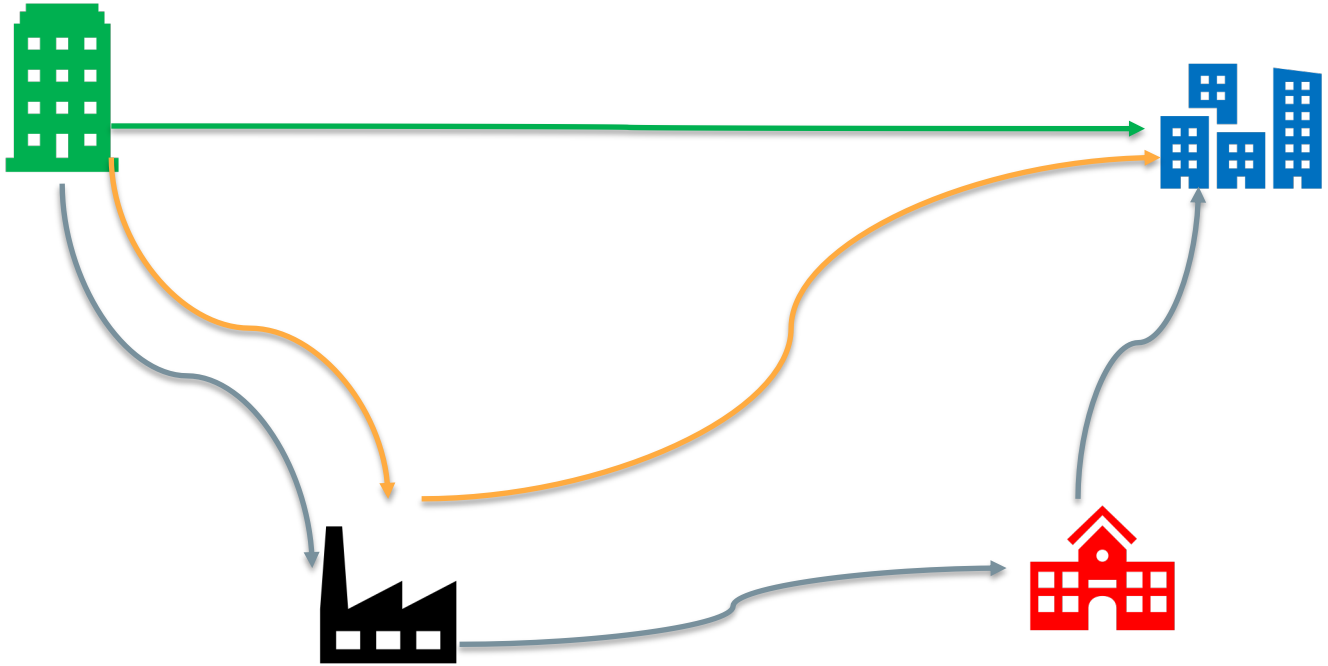
Digital Career Institute

Algorithms and algorithmic thinking

- Humans use algorithms as recipes of how to behave in certain situations or to obtain a specific result.
- Algorithmic thinking involves describing actions to be taken in a step-by-step guide, thereby making it reproducible for others.
- Some things we do are more easily translated into algorithms than others.
- Computer programs consist of many algorithms.
- Most of the low-level algorithms, dealing with basic operations, have been extensively studied and there are often one or more implementations available.

Algorithm Analysis

Algorithm Analysis



Algorithm Analysis

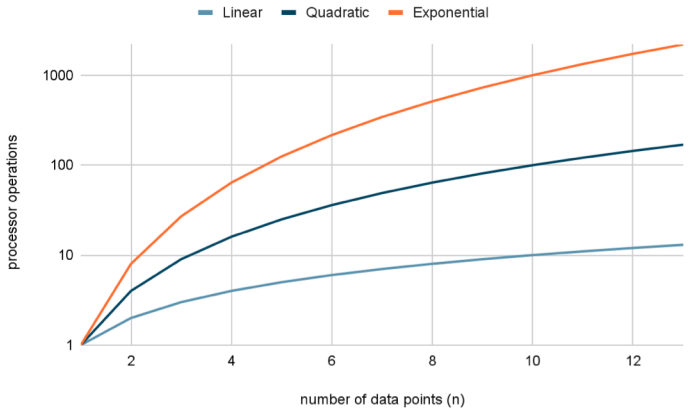
Digital Career Institute

- Algorithm analysis helps us to determine which algorithm is most efficient in terms of time and space consumed.

- The goal of the *analysis of algorithms* is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.)

Algorithm Analysis

- Look at an algorithm and predict how much longer the algorithm will take or how much more memory it will require if it is run on more or more complex arrays/objects.
- More advanced: takes I/O and net activity into account.
- Try to avoid using algorithms that will cause resource increase in production settings.
- Test data and test settings on the laptops of developers are often smaller than real world production data - therefore problems of algorithms that do not do well with larger amounts of data will often not be directly visible before the application has been deployed.
- Execution time and memory usage will almost always grow when processing more data. What matters is the rate of growth:
    - linear
    - quadratic
    - exponential
    - etc.



Algorithm Analysis

Digital Career Institute

Which algorithm to choose (run-time)?

```
def prefix_averages(x: list[int]) -> list[int]:
        n = len(x)
        a = []
        for i in range(0, n):
                s = 0
                for j in range(0, i+1):
                        s = s + x[j]
                a.append(s / (i + 1))
        return a
```

```
def prefix_averages(x: list[int]) -> list[int]:
        n = len(x)
        a = []
        s = 0
        for i in range(0, n):
                s = s + x[i]
                a.append(s / (i + 1))
        return a
```

Algorithm Analysis

Digital Career Institute

Which algorithm to choose (memory)?

```
def compare_lists(x: list[int], y: list[int]) ->
list[object]:
            n = len(x)
            m = len(y)
            a = []
            for i in range(0, n):
                        b = []
                        for j in range(0, m):
                                    b.append([x[i],
y[j]])
                        a.append(b)
            c = []
            for a_item in a:
                        d = []
                        for b_item in a_item:

            d.append(b_item[0] == b_item[1])
                        c.append(d)
```

```
def compare_lists(x: list[int], y: list[int]) ->
list[object]:
            a = []
            for i in x:
                        b = []
                        for j in y:
                                    b.append(i == j)
                        a.append(b)
            return a
```

Algorithm Analysis (Aspects To Consider)

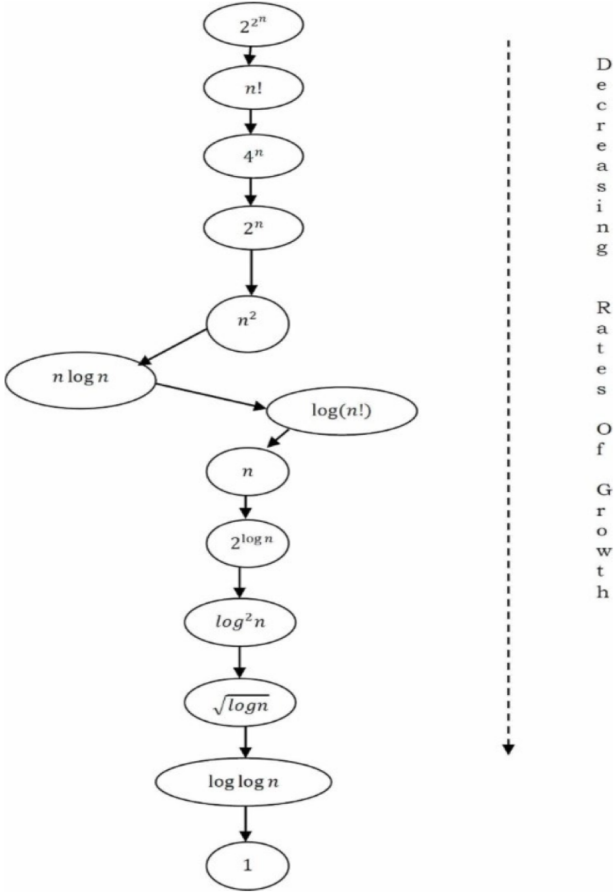
Digital Career Institute

- **Memory:** How much information needs to be stored in memory for data of size X.
- Prevent Computer from Crashing

Algorithm Analysis (Aspects To Consider)

- **Run-time:**
- It is the process of determining how processing time increases as the size of the problem (input size) increases. Input size is the number of elements in the input, and depending on the problem type, the input may be of different types.
    - Count primitive operations.
    - Look for loops on input data:
        - single loops: linear growth.
        - loops inside of loops: quadratic growth.
        - multiple level of loops: exponential growth.

Algorithm Analysis (Aspects To Consider)



Algorithm Analysis (Aspects To Consider)

Digital Career Institute

Time Complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
$logn$	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
$nlogn$	Linear Logarithmic	Sorting n items by 'divide-and-conquer' - Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

Algorithm Analysis (Aspects To Consider)

Digital Career Institute

- **How algorithm will behave:**
  - worst-case - important: can cause app to halt/slow down significantly.
  - best case.
  - average.
  - amortized (averaged over time).

Algorithm Analysis

Worst case vs. best case

```
def x_contains_y(x: list[int], y: int) -> bool:
        n = len(x)
        for i in range(0, n):
                if x[i] == y:
                        return True
        return False
```

- Best case: y == x[0].
- Worst case: y not in x.
- **Worst case creates a problem** so we mostly focus on worst case scenarios.

At the Core of the Lesson

- Algorithm analysis is used to quickly spot problems in programs when the amount of data increases.
- Program run-time and memory usage are key points to look at.
- We look at best case, worst case and averages, but worst-case scenarios are extra important as they can create the biggest kinds of problems.

O-Notation

O-Notation

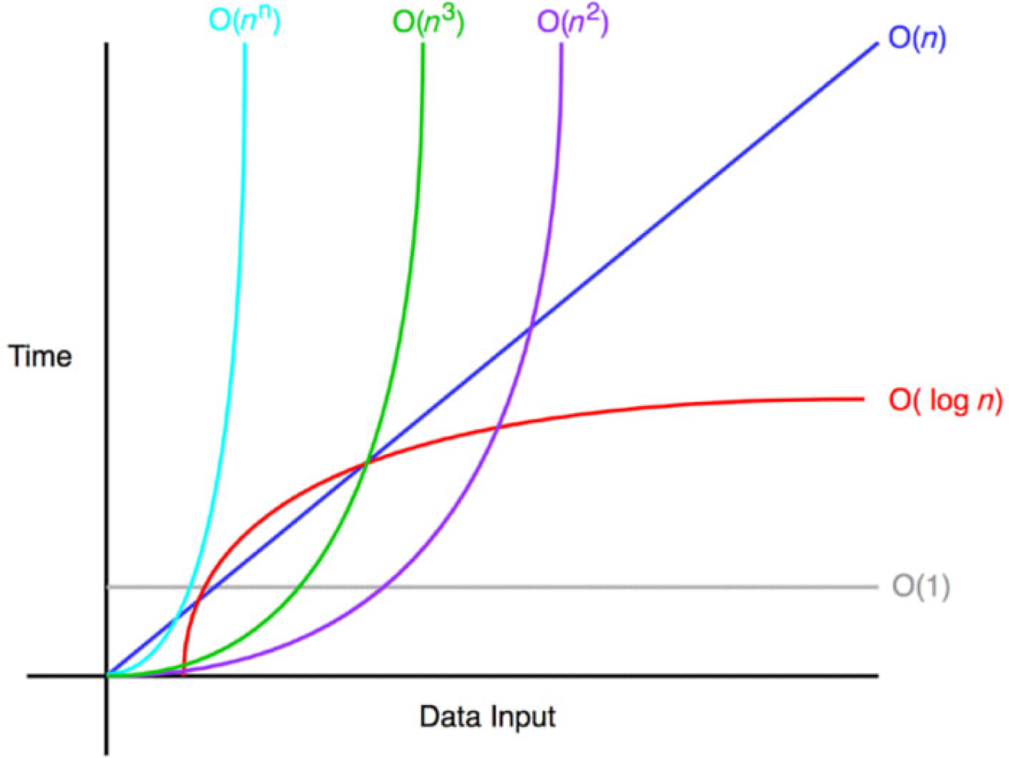
- It is a way to describe how execution time/space requirements increase when the number of input data objects (N) increases.
- It makes it possible to compare algorithms with one-another in terms of speed/memory usage.
- If the resources used do not depend on N at all, the o-notation will be O(1).
- If the resources used do depend on N, we will look at how steep the curve of the o-notation function is.

O-Notation

Digital Career Institute

Time Complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
$logn$	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
$nlogn$	Linear Logarithmic	Sorting n items by 'divide-and-conquer' - Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

Some common O-notations



Source: https://dev.to/b0nbon1/understanding-big-o-notation-with-javascript-25mc

O-Notation

- In simple programs, the o-notation can be calculated manually.

```
def my_program(N):
        print('hello')
        for i in N:
                ...
                for j in N:
                        ...
```

Does not depend on size of N, so ignore it.

Is run once for each N, so we are at O(N).

A loop within a loop on N. Now we are at $O(N^2)$.

Bubble sort

- The complexity grows exponentially: $O(n^2)$ (average + worst case).
- The best-case is O(n).

At the Core of the Lesson

- O-notation can help us judge how an algorithm will perform given worst-case scenarios and large amounts of data.
- It allows for a comparison of different algorithms in terms of performance under certain settings.
- Logarithmic and linear O-notation is generally preferable when working on large datasets.

Sorting Problems

Where Does Sorting Occur in an App?

Digital Career Institute

Database	Backend code	Frontend code
• Requests for large amounts of complex data, often with pagination of results, etc. can slow the database down. • Sorting is built-in into the database software. • To prevent slow downs, we need to store the data in the database in a way that makes it easy for the database to respond with the right data. • We should add 'indexes' for data that is frequently requested.	• We sort things that cannot easily be done in database (for example contents of json field). • Algorithms that take lots of memory or lots of computing time **affect all users** currently connected to the server. • **Easy to program**: All the data is available on the server.	• Some data can be sorted inside the browser code - for example for visual appearance of a data table. • Inefficient algorithms **affect only local user**. • **Difficult to program**: Sorting can only be done on the that has been loaded into the browser and that is not always possible when dealing with millions of data points.

Bottlenecks to Think About

Digital Career Institute

Processor	Memory	Input/Output (IO)	Networking
• How many operations does the processor need to do? • Newer processors work a lot faster than older processors, but if the number of calculations increases rapidly then also the newer processor may not be able to calculate reasonably fast.	• Can the algorithm work on the data "in place" or does it need to store copies of the data in memory? • If it needs copies of data - how much extra space does it require? • How much data are we going to work on? If we need to store more data than what we have space for in RAM, then we will have to store part of the RAM on disc.	• Is the algorithm depending on any connected devices? • Is the algorithm depending on users typing on the keyboard or using the mouse? Will this slow it down?	• Is the algorithm depending on accessing files and resources on the internet? • If it does require to download or receive answers from the internet, how fast is the connection? Can it break down at times?

Algorithm Analysis

Digital Career Institute

Bubble sort

```
def bubble_sort(our_list):
    has_swapped = True

    while(has_swapped):
        has_swapped = False
        for i in range(len(our_list) - 1):
            if our_list[i] > our_list[i+1]:
                # Swap
                our_list[i], our_list[i+1] = our_list[i+1], our_list[i]
                has_swapped = True
    return our_list
```

- It is simple and easy to understand.
- It is not very fast.

At the Core of the Lesson

- Sorting can be done on various levels within a web app.
- Inefficient sorting on the server affects everyone, sorting in the browser only the local user.
- The browser may not have all the data needed to do sorting (due to pagination), and it is a trade-off sending all data to the client every time.
- Bubble sort is easy to understand, but it is not very efficient.

Quicksort

Quicksort

Digital Career Institute

- It is a common sorting algorithm.
- It was invented in 1959/61 by Tony Hoare.
- It does not require much extra memory because the sorting takes place "in-place".
- It is a "divide and conquer" algorithm (breaks problem down into smaller pieces until solvable directly).
- It has been implemented in every computer language.
- The Chrome browser uses it for sorting lists , other browsers use other algorithms.
- Worst case: Execution time $O(n^2)$.
- Code: https://github.com/dci-python-course/Python-algorithmic_thinking-quicksort .

At the Core of the Lesson

- Quicksort is an old sorting algorithm that is still being used.
- It uses little extra memory as sorting is done "in place"..
- The worst case run-time is $O(n^2)$.

Digital Career Institute

Mergesort

Mergesort

- It is another common sorting algorithm.
- It was invented in 1945 by John von Neumann.
- It does require extra memory because the sorting does not take place "in-place".
- It is more efficient for larger datasets than quicksort.
- The best/worst/average run-time case is O(n log n).
- It is also a "divide and conquer" algorithm (breaks problem down into smaller pieces until solvable directly).
- It has also been implemented in every computer language,
- Code: https://github.com/dci-python-course/Python-algorithmic_thinking-mergesort .

At the Core of the Lesson

- Mergesort is an old sorting algorithm that is still being used.
- It uses more memory than quicksort as sorting is NOT done "in place".
- It is more time efficient for large datasets than quicksort.

Documentation

Read More

Digital Career Institute

Algorithms (edu.gfcglobal.org)
Analysis of Algorithms (tutorialspoint.com)
Analysis of Algorithms (slideplayer.com)
Sorting in JavaScript: Handling Google Chrome's Unstable Sort (dcminfo.com)
The Quicksort algorithm in Python (realpython.com)
Quick sort vs. Merge sort (geeksforgeeks.org)
OpenDSA (opendsa-server.cs.vt.edu)

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute