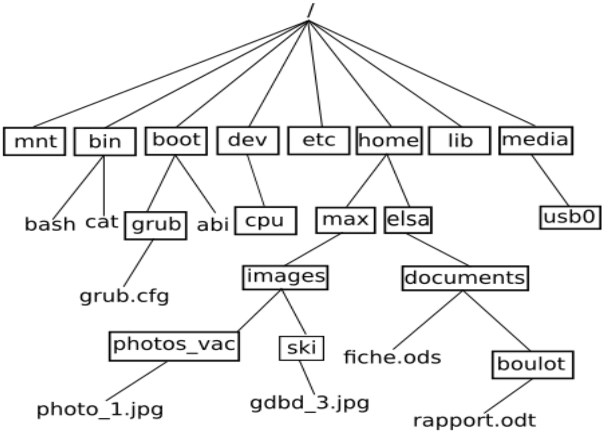
Quiz

- Define what these commands do:
  - cd
  - mkdir
  - touch
  - cp
  - mv
  - ls
- What's Git?
- What's the difference between Git and Github?
- Define the steps to update the github repo from local machine?
- What's the difference between "git push" and "git pull".

Quiz

Exercise:

- What's the absolute path of 'rapport.odt'?
- How can I list the contents of photos_vac from boulot?
- How can I copy the contents of photos_vac to boulot from images?



Algorithmic & Python Programming

Imad Kissami[1]
[1]Mohammed VI Polytechnic University, Benguerir, Morocco

Algorithm

Example

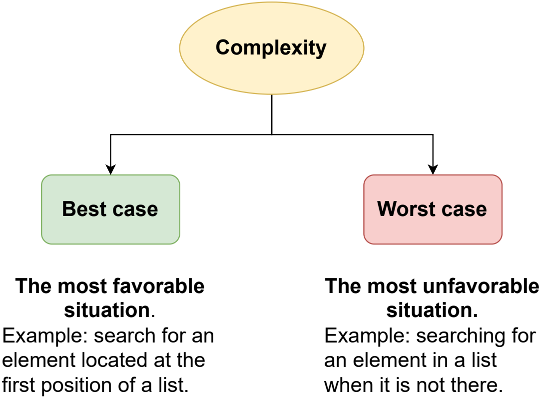
```
Function Multiply(Integer A, Integer B)
    Integer C = 0
    While A is greater than 0
        C = C + B
        A = A - 1
    End
    Return C
End
```

- The calculation of the complexity of an algorithm makes it possible to measure its performance. There are two types of complexity:
  - spatial complexity: quantifies memory usage
  - time complexity: quantifies the speed of execution
- Since it is only a question of comparing algorithms, the rules of this calculation must be independent of the:
  - programming language;
  - processor;
  - compiler.
- For the sake of simplicity, we will assume that all elementary operations are at equal cost, i.e. 1 "unit" of time.
  - Example: $a = b \times 3$: 1 multiplication $+$ 1 assignment $=$ 2 "units"

Algorithm

Complexity: Scenarios

- Example: Sequential search for an element in an unsorted list.



Algorithm

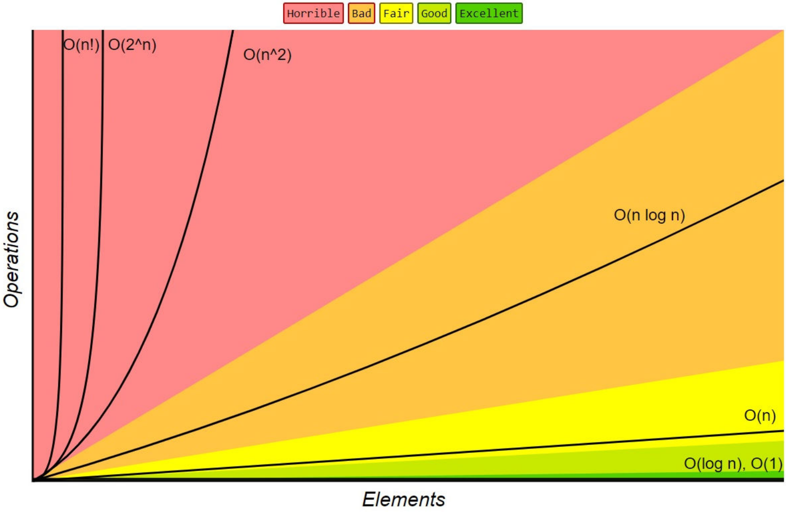
Temporal Complexity

- $\mathcal{O}(1)$: Pronounced **"order 1"** and denoting a function that runs in **constant time**
  - $c(n+1) = c(n) => \mathcal{O}(1)$
- $\mathcal{O}(n)$: Pronounced **"order n"** and denoting a function that runs in **linear time**
  - $c(n+1) = c(n) + 1 => \mathcal{O}(n)$
- $\mathcal{O}(\log n)$: Pronounced **"order log n"** and denoting a function that runs in **logarithmic time**
  - $c(n+1) = c(n) + \epsilon$; $\mathbf{(c(2n) = c(n) + 1)} => \mathcal{O}(\log n)$
- $\mathcal{O}(n^2)$: Pronounced **"order n squared"** and denoting a function that runs in **quadratic time**
  - $c(n+1) = c(n) + n => \mathcal{O}(n^2)$
- $\mathcal{O}(2^n)$: Pronounced **"order 2 power n"** and denoting a function that runs in **exponential time**
  - $c(n+1) = 2 * c(n) => \mathcal{O}(2^n)$

$$\mathcal{O}(1) < \mathcal{O}(\log n) < \mathcal{O}(\sqrt{n}) < \mathcal{O}(n) < \mathcal{O}(n \log n) < \mathcal{O}(n^2) < \mathcal{O}(n^3) < \mathcal{O}(2^n) < \mathcal{O}(10^n) < \mathcal{O}(n!)$$

Algorithm

Temporal Complexity



Algorithm

Temporal Complexity: Example

```python
def factorial(n):
    fact = 1
    i = 2

    while i <= n:
        fact = fact * i
        i = i + 1
    return fact
```

- Complexity

```
assignment: 1
assignment: 1
iterations: at most n − 1
    comparison: 1
    multiplication + assignment: 2
    addition + assignment: 2
```

Temporal Complexity: Example $\mathcal{O}(1)$

```
def f1(n):
    print("hello")
```

- Complexity: $T(n) = \mathcal{O}(1)$

```
def f2(n):
    s = input("enter a character")
    print(s)
```

- Complexity: $T(n) = \mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(2) = \mathcal{O}(1)$

Temporal Complexity: Example $\mathcal{O}(n)$

```python
def f3(n):
    for i in range(n):
        print("hello")
```

- Complexity: $T(n) = (\mathcal{O}(1) + \mathcal{O}(1)) \times n = \mathcal{O}(2n) = \mathcal{O}(n)$

```python
def f4(n):
    s = 0.
    for i in range(n):
        s = s + i
    for i in range(n):
        s = s * i
```

- Complexity: $T(n) = \mathcal{O}(1) + \mathcal{O}(3n) + \mathcal{O}(3n) = \mathcal{O}(6n+1) = \mathcal{O}(n)$

Algorithm

Temporal Complexity: Example $\mathcal{O}(n^2)$

```python
def f5(n):
    for i in range(n):
        for j in range(n):
            s = s + i*j
```

- Complexity: $T(n) = \mathcal{O}(5n^2) = \mathcal{O}(n^2)$

Algorithm

Temporal Complexity: Example $\mathcal{O}(\log n)$

```python
n = 300
count = 0
while n != 0:
    n //= 2
    count += 1
print("repetition count is:", count)
```

```
repetition count is: 9
```

- Number of repetition is: $1 + log_2(300) = 9.2 \approx 9$
- Complexity: $T(n) = \mathcal{O}(5 \times (1 + \log_2 n)) = \mathcal{O}(\log_2 n)$

Algorithm

Temporal Complexity: Example $\mathcal{O}(n \log n)$

```python
m = n = 300
count = 0

while m != 0:
    while n != 0:
        n  //= 4
        count += 1
    m -=1
print("repetition count is:", count)
```

```
repetition count is: 5
```

- Number of repetition is: $(1 + log_4(300)) = 4.1 \approx 5$
- Complexity: $T(n) = \mathcal{O}(8n \times (1 + \log_4 n)) = \mathcal{O}(n \log_4 n)$

Algorithm

Temporal Complexity: Example $\mathcal{O}(2^n)$

```python
def fibonacci(n):
    if(n <= 0):
        return n
    else:
        return (fibonacci(n-1) + fibonacci(n-2))
n = int(input("Entrez le nombre de termes:"))
print("Suite de Fibonacci en utilisant la recursion :")

for i in range(n):
    print(fibonacci(i))
```

- if n $= 1$: 3 calls $= 2^1 + 1$
- if n $= 2$: 5 calls $= 2^2 + 1$
- if n $= 3$: 9 calls $= 2^3 + 1$
- if n $= 4$: 15 calls $= 2^4 - 1$
- if n $= 5$: 25 calls $= 2^5 - 7$
- General case:
  - Complexity: $T(n) = \mathcal{O}(2^n + const) = \mathcal{O}(2^n)$