Introduction

Processing math: 100%

Elementary operations can be grouped in 2 categories:

1. those that rely on 'external' resources, which are slow:
    - hard drives
    - network
2. thos that rely on 'internal' resources, which are fast
    - CPU/GPU
    - memory

The codes that run slowly due to the first family of causes are called **IO bound** problems. The other one are **CPU bound**.

Example of a *IO bound* problem

Let `f` be defined as:

In [1]:

```
import pandas as pd
import numpy as np

def f_IO(n, k):
    # CPU-like tasks
    arr1 = np.random.rand(n)
    arr1 = arr1**2 + arr1 * 5 + np.exp(arr1-10)

    arr2 = np.random.rand(k)
    sr = pd.Series(arr2)

    # IO-like tasks
    sr.to_csv('data.csv')
```

Processing math: 100%

This function creates 2 arrays of size `n` and `k`, do some mathematical operations on the first array and write the second one on disk.

Theorical time complexity

In theory, the function has the following time complexity:

$$C(n, k) = C_1 + C_2 \times n + C_3 \times k$$

With:

- C_1 overhead run time
- C_2 operations proportional to `n`: creation of `arr1`, various calculations using `arr1`
- C_3 operations proportional to `k`: creation of `arr2`, creation of the `Serie` object, export to disk

Asymptotically, time complexity grows the same way with respect to `k` or `n`. This is described by:

$$C(n, k) = O(n) + O(k) = O(n + k)$$

Real time complexity

Yet, the C_3 coefficient is much bigger than C_2 since **it is related to disk operations.** Thus **the asymptotical behaviour corresponds `n` values that are too large to correspond to any practical use**. Hence the real time complexity is much more something like:

$$C(n, k) = O(k)$$

Experimental running time

Let's measure the real running times of this function for:

- $n \in [10^3, 10^6]$
- $k \in [10^3, 10^6]$

Results, **in milliseconds**, are presented hereafter:

In [2]:

```
pd.DataFrame(data = {'$k=10^3$': [4.320, 48.6], '$k=10^6$': [2330, 2360]},
             index=['$n=10^3$', '$n=10^6$'])
```

Out[2]:

	$k = 10^3$	$k = 10^6$
$n = 10^3$	4.32	2330
$n = 10^6$	48.60	2360

Interpretation: the **contribution of `n` is negligible compared to the one of `k`**. A factor 1000 increase of `n` adds only a few milliseconds to the running time.

Processing math: 100%

Conclusion

The theoretical estimation of a problem time complexity can be very difficult.

1. Some simple estimators are:
    - search for overhead run times
    - search for disk/network operations, in opposition to CPU/RAM
2. Sometimes, experimental measurements are a better a choice.