Ch1 Introduction

3.3 Performance

t: liste
key: clé cherchée

```
for i in range(len(t)):
    if t[i] == key:
        print(i)
        break
```

complexité : ~ nombre d'opérations

- nombre d'el^ts de la liste
- nombre de x qu'on trouve la clé

ordre de grandeur : ~ len(t)

$O(n)$ n = taille des données

coût moyen
coût dans le pire des cas.)

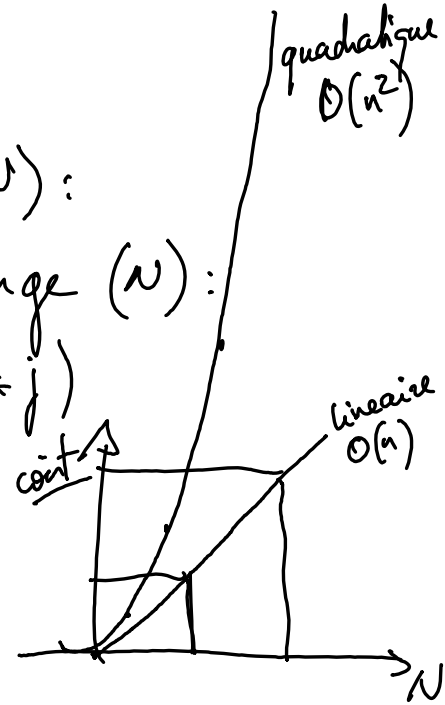
variante avec break :

- cas favorable key == t[0]
  ↳ coût indépendant de n (constant)

- cas défavorable
  $\forall i$ t[i] == key
  ↳ coût $\sim O(n)$

- cas moyen
  $\exists i$ t[i] == key
  coût $\sim O\left(\frac{n}{2}\right) \sim O(n)$

```
for i in range(N):
    for j in range(N):
        print(i*j)
```



$N = 10$

$i = 0$
- $j = 0$
- $j = 1$
- $j = 2$
- $j = 3$
- ⋮
- $j = 9$

(N)

$i = 1$
- ⋮

(N)

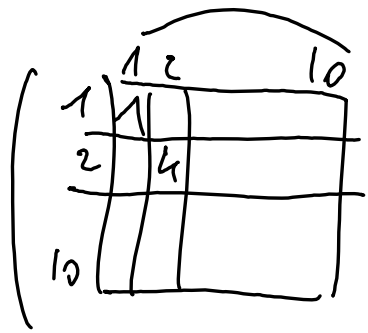
⋮

(N)

$N \times N$ tours

N^2

$10 \times 10 = 100$



exemples de complexité.

$O(n)$ linéaire recherche ds 1 liste

$O(\log_2 n)$ logarithmique recherche dichotomique

$O(n^2)$ quadratique tri de base

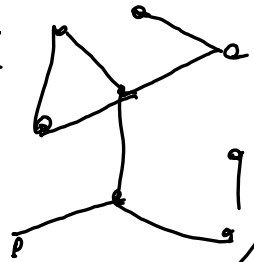
$O(n \log_2 n)$ log-linéaire quicksort.

$O(n^c)$ polynomial parsing des LAV n^3-n^7

$O(c^n)$ Exponentiel

$O(n!)$ combinatoriel voyageur de commerce

$n! = n \times (n-1) \times (n-2) \cdots \times 1$



.
.
.
.
.

$O(n^n)$

NP-complet