- algo efficace :- programmer en c pas python
  - mesurer nbr opération (pour tric meilleur cas n log n)
  - quantité de mémoire nécessaire

// NB : quick sort → mélanger tab avant pour ne pas tomber dans le n²
mais plutôt dans le n log n

↳ complexité moyenne faire attention

// Programmer les algos en
// Partiel 3 nov
- diviser pr régner
- prog dynamique
Exam
TD noté peut-être
// Noter le cours si écrit sur tableau

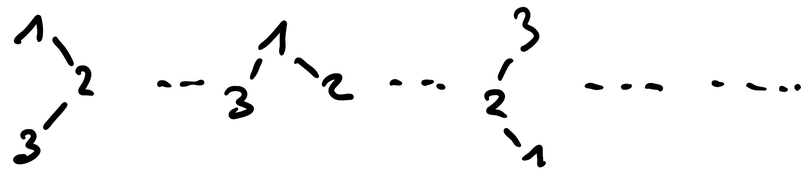
Efficacité Algo : ← nbr opération nécessaire pour algo A sur x donnée.

Pire cas : $C_A(n) = \max C_A(x)$ ← distribution de la prob sur n éléments

moyenne : $C_{A\ moy}(n) = \sum_{x,|x|=n} p(x).C_A(x)$

amortie : | coût cumulé sur n opérations
| ex : tableau dynamique (la taille double quand besoins place de + estimée)

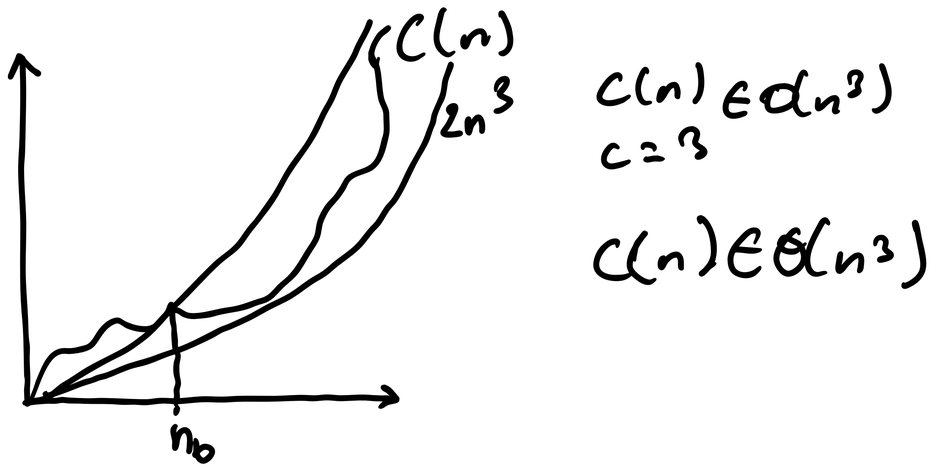
// ex arbre binaire avec 3 clés 1,2,3



pas la même prob d'apparaître.

Ordre grandeur Notation : $O()$, $\Omega()$, $\Theta()$

$n < n\log(n) < n^k < k^n < n! < n^n$ avec k constante



$C(n) \in O(n^3)$
$c = 3$

$C(n) \in \Theta(n^3)$

. trs chercher si existe algo meilleur

Algo diviser-pour-regner

Ce schéma algorithmique procède en 2 étapes :

① on divise le pb à résoudre en plusieurs sous problèmes qui leur ressemble ; puis

② on construit une solution pour le pb initial à partir des solutions des ss-problèmes

"divide and conquer and combine"

La recherche dichotomique

Rechercher x dans tab trié

On teste si x est l'élément médian si on regarde dans la partie gauche ou droite.

Jusqu'à trouver ou non la valeur.

```
def recherche(T, x, bg=0, bd=None):  # TODO
```

Complexité (pire cas)

$\mathcal{C}(n) = \mathcal{C}(\lceil \frac{n}{2} \rceil) + O(1)$ ← une certaine constante.

$\mathcal{C}(1) = 1$

Exemple :

Recherche le nbr d'occurrences dans un tab trié d'une valeur.

```
# TODO
```