1 Introduction

Un des éléments de base de l'algorithmique — et donc de la programmation — est l'étude et l'utilisation d'algorithmes de tri.

Dans toute la suite, on considère une liste de taille *n strictement positive.*

On utilisera le fonction `randint` du module `random` afin de générer des listes aléatoires :

```
In [4]: from random import randint
```

2 Tri par sélection

Le principe est le suivant : on parcourt la liste une première fois en entier, en sélectionnant son plus grand élément pour l'échanger avec son dernier élément. Ainsi, à la fin du premier passage, la liste a son plus grand élément à la fin.

Ensuite, on itère le procédé sur la liste composée des $n-1$ premiers éléments de la liste initiale, et on recommence… forcément, on tombe à un moment sur la liste composé uniquement du premier élément de la liste, et on s'arrête là.

```
In [5]: def selec_tri(v):
            n = len(v)
            if n>1:
                i_max = 0
                for i in range(n):
                    if v[i]>v[i_max]:
                        i_max = i
                v[-1],v[i_max] = v[i_max],v[-1]

                l = v[:-1] ## permet d'éviter les problèmes d'adressage des listes en Python
                selec_tri(l) ## on trie la liste des n-1 premiers éléments
                v[:-1] = l ## on remplace les n-1 premiers éléments par leur liste triée
```

On peut donc essayer notre fonction `selec_tri` sur une liste générée au hasard : exécutez plusieurs fois la cellule suivante pour générer une liste aléatoire de 25 nombres entre -50 et 50, l'afficher, et la trier.

```
In [6]: n = int(input("Taille n de la liste : "))
        v = [randint(-250,250) for j in range(n)]
        print("Liste initiale :","\n",v)
        selec_tri(v)
        print(v)
```

```
Taille n de la liste : 20
Liste initiale :
 [-148, 176, -5, -238, -168, 41, 4, -127, -142, -107, 164, 164, 106, 94, 157, -38, -52, -75, -223, -
[-238, -223, -168, -148, -142, -142, -127, -107, -75, -52, -38, -5, 4, 41, 94, 106, 157, 164, 164, 1
```

La complexité de cet algorithme de tri est de l'ordre $O(n^2)$.

3 Tri rapide ou « *quick sort* »

Le principe de ce tri-là est un peu plus astucieux.

Dans le tri à bulle, on sélectionnait la plus grande valeur du tableau, on la mettait à part (à la fin), et on triait le reste. On réduisait donc la taille du problème de 1 à chaque fois.

Ici, on va faire encore plus fort: on va couper la liste en deux parties, trier séparément les deux parties et tout recoller ensemble à la fin. Bien sûr, on ne va pas couper la liste au milieu comme des brutes.

```
In [7]: def quick_tri(v):
            n = len(v)
            if n>1:
```

```
    piv = n//2 ## choix du pivot ; on prend ici le plus naïf, l'élément central de la li
    vsmal,vbig = [],[]
    for i,el in enumerate(v):
        if i!=piv:
            if el < v[piv]:
                vsmal.append(el)
            else:
                vbig.append(el)
    quick_tri(vsmal)
    quick_tri(vbig)
    v[:] = vsmal + [v[piv]] + vbig
```

On peut donc l'essayer sur une liste au hasard, encore :

```
In [8]: v = [randint(-50,50) for _ in range(25)]
        print("Liste initiale :","\n",v)
        quick_tri(v)
        print("Liste triée :","\n",v)

Liste initiale :
 [46, -33, -34, -48, 25, 21, 6, -7, 12, -2, 26, 32, 15, -6, 35, 41, 14, 22, -37, -47, 38, 47, -13, -
Liste triée :
 [-48, -47, -37, -34, -33, -29, -13, -10, -7, -6, -2, 6, 12, 14, 15, 21, 22, 25, 26, 32, 35, 38, 41,
```

** *Complexité* **

La complexité de cet algorithme de tri dépend du choix du pivot. Avec un bon pivot, on peut faire en sorte de diviser par un facteur 2 la taille de la liste à chaque fois. Alors, on est amené à une complexité vérifiant la relation

$$C_n = 1 + 3 + \sum_{k=1}^{n} O(1) + 2C_{\left\lfloor \frac{n}{2} \right\rfloor} = 2C_{\left\lfloor \frac{n}{2} \right\rfloor} + \Theta(n)$$

Ce type de relation de récurrence est quasiment impossible à résoudre dans le cas général ; cependant, un théorème important en analyse des algorithmes, nommé le *Master's theorem*, nous fournit le résultat: la complexité est quasi-linéaire.

$$C_n = \Theta(n \log n)$$

Maintenant, avec un pivot complètement naze (par exemple le plus grand élément de la liste), on sépare la liste en une liste comportant le pivot tout seul et une autre qui est donc de taille $n - 1$… et on est ramené à un tri par sélection ! Donc **dans le pire des cas,** la complexité est en $O(n^2)$.