Diviser pour régner : méthode algorithme

Pour résoudre un pb de "grande taille"

- on divise le pb en sous pb de nature identique et de taille strictement inférieure
- on résout (récursivement) le(s) sous pb
- on déduit de sol° au(x) sous pb la solution du pb initial

→ exponentiation rapide : calcul de x^n

- exprimer x^n en fct de $(x^2)^{\lfloor n/2 \rfloor}$
- calculer $(x^2)^{\lfloor n/2 \rfloor}$ ($\lfloor n/2 \rfloor < n$ si $n \geq 1$)
- en déduire x^n

→ tri fusion : tri d'une liste ℓ

- si la longueur de ℓ est ≥ 2 :
  on coupe ℓ en 2 parties "égales" ℓ_1 et ℓ_2
- on trie ℓ_1 et ℓ_2
- on fusionne les deux listes triées.

→ tri pour pivot / tri rapide (quicksort)

[6; 10; 2; 8; 9; 3; 4; 7; 42; 1]

ℓ_1 : [2; 3; 4; 1] → ℓ_1t [1; 2; 3; 4]

6

ℓ_2 : [10; 8; 9; 7; 42] → ℓ_2t [7; 8; 9; 10; 42]

ℓ triée : ℓ_1t @ [6] @ ℓ_2t

```
let partition p l =
  let l1 = List.filter (fun x -> x <= p) l in
  let l2 = List.filter (fun x -> x > p) l in
  (l1, l2)
;;
```

List.filter : ('a → bool) → 'a list → 'a list

List.filter f l renvoie la liste des éléments x de l tels que f x vaut true

```
let rec quicksort l =
  match l with
  | [] -> []
  | t::q -> let (l1, l2) = partition t q in
            let l1t = quicksort l1 in
            let l2t = quicksort l2 in
            l1t@(t::l2t)
;;          l1t@[t]@l2t
```

- List.filter a une complexité temporelle en $O(n)$ où n est la longueur de la liste
  Donc partition est de complexité linéaire en la taille de liste

- Temps de calcul de quicksort ℓ si ℓ est non vide, de longueur $n \geq 1$.
  ① coût de partition
  ② deux appels récursifs
  ③ concaténation.

On a

① partition linéaire : il existe deux constantes α et β tq le coût de partition ℓ est majoré par $\alpha(n-1)+\beta$

$O(n)$: APCR : majoré par $\alpha \times (n-1)$; à partir d'un certain rang : majoré par β } majoré par $\alpha(n-1)+\beta$

③ Complexité de $\ell_1 @ \ell_2$: linéaire en la longueur de ℓ_1

Coût de $\ell_1 t @ (t :: \ell t)$: linéaire par rapport à la longueur de ℓ_1 qui est majorée par $n-1$

Il existe λ, μ deux constantes tq le coût de ③ est majoré par $\lambda(n-1)+\mu$

Pour ① et ③ : coût majoré par $an+b$ avec a, b deux constantes

Coût total : $\sum_{\ell \text{ sur lesquelles on appelle quicksort}} a|\ell|+b$ où $|\ell|$ est la taille de ℓ

$aS + bN$ où S : somme des $|\ell|$ de toutes les listes sur lesquelles on fait un appel

et N : nombre d'appels.

Si $m = \max\{a, b\}$, alors $aS + bN \leq m(S+N)$

Notons $C(n)$ la valeur maximale de $S+N$ pour une liste ℓ initiale de longueur n.

- Si $n = 0$, la liste est vide, pas d'appel récursif. donc $S = 0$ et $N = 1$ → appel initial

  $\boxed{C(0) = 1}$

- Si $n \geq 1$: la liste est séparée en trois.
  * un élément qui sert de pivot (sa tête t) 1
  * la liste ℓ_1 des éléments $\leq t$. j
  * la liste ℓ_2 des éléments $> t$. $n-1-j$

  Si on note $j = |\ell_1|$, alors $|\ell_2| = n-1-j$

  Il existe $k \in [\![0, n-1]\!]$ tel que

  $$C(n) \leq C(k) + C(n-1-k) + \underbrace{n}_{\text{longueur de } \ell} + 1$$

  $\underbrace{N}_{\text{nbr total d'appels}} = \underbrace{1}_{\text{l'appel en cours}} + \underbrace{N_1}_{\text{les appels pour } \ell_1} + \underbrace{N_2}_{\text{les appels pour } \ell_2}$

  $\underbrace{S}_{\text{somme des listes}} = \underbrace{n}_{\text{la liste courante}} + \underbrace{S_1}_{\text{la somme de 1er appel récursif}} + \underbrace{S_2}_{\text{pour le 2e appel}}$

  $N + S = (N_1 + S_1) + (N_2 + S_2) + n + 1$

  donc $C(n) \leq C(k) + C(n-1-k) + n + 1$

On montre par récurrence forte que $\forall n \in \mathbb{N}$, $C(n) \leq (n+1)^2$

* $C(0) = 1 \leq (0+1)^2$
* Soit $n \in \mathbb{N}^*$, on suppose que $\forall k \in [\![0, n-1]\!]$, $C(k) \leq (k+1)^2$

  alors, comme il existe $k \in [\![0, n-1]\!]$ tq $0 \leq k \leq n-1$ donc $0 \leq n-1-k \leq n-1$

  $C(n) \leq C(k) + C(n-1-k) + n + 1$

  $C(n) \leq (k+1)^2 + (n-k)^2 + n + 1$

  $(k+1)^2 + (n-k)^2 + n + 1 = k^2 + 2k + 1 + n^2 - 2nk + k^2 + n + 1$

  $= (n+1)^2 - n + 2k^2 + 2k + 1 - 2nk$

  $= (n+1)^2 - \underbrace{2k}_{\geq 0}\underbrace{(n-k-1)}_{\geq 0} - \underbrace{(n-1)}_{\geq 0}$

  $\leq (n+1)^2$ CQFD

Verdict : complexité est un $O(n^2)$, borne atteinte

Pire des cas : liste déjà triée Rmq : complexité moyenne en $O(n \log n)$