Algo - Séance 5 de TD - 9 mars 2021

TD 4, exo 3 : Tri sélectif récursif

1) **Solution:**

Cette question tombe tout le temps en examen. Inutile de dire ce qui se passe, juste donner les affichages ! !

```
A l ppel T[ 0 , 6 ]= [3, 1, 8, 5, 1, 4, 3]
A l ppel T[ 1 , 6 ]= [3, 8, 5, 1, 4, 3]
A l ppel T[ 2 , 6 ]= [8, 5, 3, 4, 3]
A l ppel T[ 3 , 6 ]= [5, 8, 4, 3]
A l ppel T[ 4 , 6 ]= [8, 4, 5]
A l ppel T[ 5 , 6 ]= [8, 5]
En sortie T[ 5 , 6 ]= [5, 8]
En sortie T[ 4 , 6 ]= [4, 5, 8]
En sortie T[ 3 , 6 ]= [3, 4, 5, 8]
En sortie T[ 2 , 6 ]= [3, 3, 4, 5, 8]
En sortie T[ 1 , 6 ]= [1, 3, 3, 4, 5, 8]
En sortie T[ 0 , 6 ]= [1, 1, 3, 3, 4, 5, 8]
```

2) $\Pi(n)$: pour $n = f-d+1$, $0 \leq d \leq f \leq n-1$ (taille de Tab), alors l'appel à TriSelecRec(T, d, f) se termine et ~~renvoie~~ trie (en place) le sous-tableau $T[d..f]$ en ordre croissant.

Base : $n=1$, $d=f$. La condition "if $d<f$" n'est pas valide et le tableau n'a qu'une case donc il est trié. La fonction termine après 1 condition "if".

Induction : Récurrence faible : supposons $\Pi(n)$ est vraie. Prenons un tableau T, d et f tq $n+1 = f-d+1$

Comme $n+1 \geq 2$, on a $f > d$ donc on entre dans le "if" :

imin ← Recherche min qui renvoie le min (exo 2 q 2)

Donc imin contient l'indice de la valeur min de tab[d...f]

Ensuite, on "swap" (intervertit) T[d] et T[imin]. Donc le plus petit élément est en position d. Puis l'appel récursif TriSelecRec (T, d+1, f) correspond à $n' = f-(d+1)+1 = n$

donc par hypothèse de récurrence, ça termine avec T trié. D'où $\Pi(n+1)$

Conclusion : la fonction trie bien T.

Sachant que RechercheMin fait $\Theta(f-d)$ comparaisons, quelle est la complexité de TriSelecR ?

3) Complexité en nb de comparaisons c_n pour $n = f-d+1$

Base : $c_0 = 1 = c_1$.

L'appel pour $n > 1$ effectue RechercheMin(d, f) avec $f-d$ comp. en plus du "if". D'où $c_n = \underbrace{f-d}_{\text{Min}} + \underbrace{1}_{\text{if}} + \underbrace{c_{n-1}}_{\text{appel rec}} = n + c_{n-1}$

$$= n + (n-1) + (n-2) + \dots + (n-i+1) + c_{n-i}$$

$$= \sum_{i=0}^{n-1} n-i \quad + c_1 \quad = \boxed{\frac{n(n-1)}{2} + 1}$$

D'où complexité $\Theta(n^2)$

càd $\boxed{\Theta((f-d)^2)}$

Rappel de cours sur les tris

Tris simples (plutôt directs à programmer) :

$O(n^2)$
- tri par sélection (TD 4 exo 3 ci-dessus) : on trouve le plus petit elmt et on le met au début
- $\Omega(n)$ - tri par insertion : le début du tableau est trié, et on insère l'elmt suivant à la bonne place
- tri à bulles (TD 5 exo 1 ci-dessous) : parcourt le tableau en remontant l'élmt pointé si possible

$O(n^2)$ Tri rapide (quicksort, TD 5 exo 3) : on choisit un pivot x, on sépare le tableau entre L1 (elmts ≤ x) et
$\Omega(n \log n)$ L2 (elmts ≥ x) et on les trie (récursivement), puis on renvoie L1 . x . L2 (concaténation de listes)
moyenne $\Theta(n \log n)$

Tri fusion (merge sort, TD 5 exo 2) : sépare le tableau en deux parties ±égales, on les trie chacune de son côté, puis on les fusionne.
$\Theta(n \log n)$ Fusion : on regarde le début des deux listes triées, et on sélectionne le plus petit élmt qu'on voit.

Remarque : on peut prouver qu'un tri requiert au moins n log n comparaisons dans le pire des cas.
Donc tous les tris qui ont une complexité en n log n sont considérés optimaux.

TD5 exo
1 → Magenta
2 → Vert
3 → Jaune, Bleu
$\Rightarrow$ 10h05

TD5, exo 1 : bubble sort

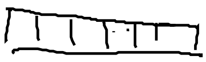
1) Push(T, k) avec $1 \leq k < n$ fait $k-1$ boucles $(j = 1, 2, \dots, k-1)$ avec chacune 1 comparaison. Donc $k-1$ comparaisons.

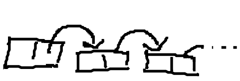
2) On fait des boucles pour $i = 0, 1, \dots, n-1$ donc n tours. Chaque tour appelle Push(T, n−i), donc au total,

$$c = \sum_{i=0}^{n-1} c_i = \sum_{i=0}^{n-1} (\underbrace{n-i}_{k} - 1) = \sum_{j=0}^{n-1} j \quad (\text{en posant } j = n-i-1)$$

$$= \frac{n(n-1)}{2}$$

D'où une complexité en $\boxed{\Theta(n^2)}$

3)  tableau

 liste chaînée

C'est possible d'utiliser une liste chaînée, mais il faudra garder un pointeur vers la valeur précédente pour pouvoir swapper :

TD5, exo 2 : Tri fusion récursif

1) 7 9 2 1 6 4 3 1

$\underbrace{7\ 9\ 2\ 1}\quad\underbrace{6\ 4\ 3\ 1}$

$\underbrace{7\ 9}\ \underbrace{2\ 1}\quad\underbrace{6\ 4}\ \underbrace{3\ 1}$

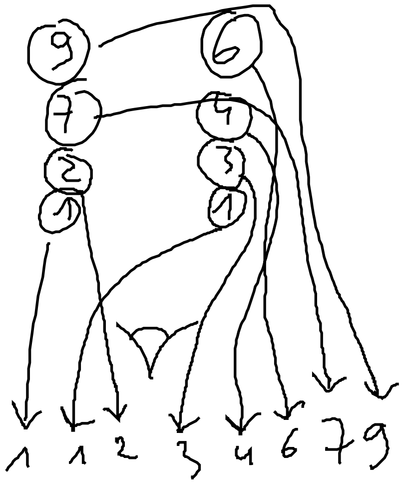
$\underbrace{7\quad 9}\ \ \underbrace{2\quad 1}\quad\underbrace{6\quad 4}\ \ \underbrace{3\quad 1}$

$\underbrace{7\ 9\quad 1\ 2}\quad\underbrace{4\ 6\quad 1\ 3}$

$\underbrace{1\ 2\ 7\ 9\qquad 1\ 3\ 4\ 6}$

1 1 2 3 4 6 7 9

$\sim \log_2 n$



Solution:
Commencer par l'arbre des appels, et en déduire la liste chronologique des appels.

```
>>> TriFusionRec(L)
Appel de TriFusion pour L= [7, 9, 2, 1, 6, 4, 3, 1]
Appel de TriFusion pour L= [7, 9, 2, 1]
Appel de TriFusion pour L= [7, 9]
Appel de TriFusion pour L= [7]
Renvoie de L= [7]
Appel de TriFusion pour L= [9]
Renvoie de L= [9]
Renvoie de L= [7, 9]
Appel de TriFusion pour L= [2, 1]
Appel de TriFusion pour L= [2]
Renvoie de L= [2]
Appel de TriFusion pour L= [1]
Renvoie de L= [1]
Renvoie de L= [1, 2]
Renvoie de L= [1, 2, 7, 9]
Appel de TriFusion pour L= [6, 4, 3, 1
Appel de TriFusion pour L= [6, 4]
Appel de TriFusion pour L= [6]
Renvoie de L= [6]
Appel de TriFusion pour L= [4]
Renvoie de L= [4]
Renvoie de L= [4, 6]
Appel de TriFusion pour L= [3, 1]
Appel de TriFusion pour L= [3]
Renvoie de L= [3]
Appel de TriFusion pour L= [1]
Renvoie de L= [1]
Renvoie de L= [1, 3]
Renvoie de L= [1, 3, 4, 6]
Renvoie de L= [1, 1, 2, 3, 4, 6, 7, 9]
[1, 1, 2, 3, 4, 6, 7, 9]
```

$P(n)$: toute liste de taille i

2) Mq $\forall i \leq n$, ~~la liste $L[0 \dots i]$~~ est triée par TriFusionRec(L)

<u>Base</u> : $n = 0$, liste vide, c'est bon
$n = 1$, il n'y a qu'un élément, qui est donc "trié".

<u>Induction</u> : Récurrence forte car on s'intéresse à propriété pour $\frac{n}{2}$ et pas juste pour $n-1$.

Montrons $P(n+1)$: on pose $m = \lfloor \frac{n+1}{2} \rfloor \geq 1$ car $n+1 \geq 2$

on crée deux listes $\underbrace{L_1 = L[0, \dots m)}_{m+1}$, $L_2\ \underbrace{[m+1, \dots, n]}$

$n - m = \lceil \frac{n}{2} \rceil = \lceil \frac{n+1}{2} \rceil$

Les deux listes sont de tailles comprises entre 1 et n.

Par hypothèses $P(m+1)$ et $P(\lceil \frac{n+1}{2} \rceil)$, ces listes sont bien triée par TriFusionRec. La fusion de deux liste triées renvoie une liste triée (voir cours). Donc L est triée.

<u>Conclusion</u> : l'algo trie L quelque soit sa taille.

3) Pour construire les 2 sous-listes, il faut parcourir toute la liste principale : $\Theta(n)$. La fusion nécessite de regarder toutes les valeurs : $\Theta(n)$. La complexité vérifie donc

$$c(n) = \underbrace{\varepsilon n}_{\text{éclatement}} + \underbrace{2c(\lfloor \tfrac{n}{2} \rfloor)}_{\text{récursion}} + \underbrace{\varphi n}_{\text{fusion}} = (\varepsilon+\varphi) n + 2(\varepsilon+\varphi)\frac{n}{2} + 2^2 c\left(\frac{n}{4}\right)$$

$$= \dots = (\varepsilon + \varphi)\, n \log_2 n + n.c(1)$$

La complexité est donc $\boxed{\Theta(n \log n)}$

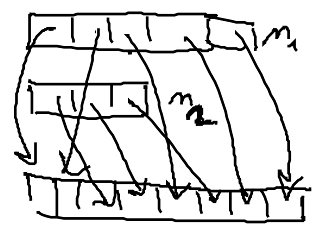
À chaque nouvelle couche de division de la liste, on a une complexité $(\varepsilon+\varphi) n$. En tout, il y a $\log_2 n$ couches. Ex : si $n = 8 = 2^3$, on a $3 = \log_2 n$ couches.

<u>Rappel</u> :

- meilleur cas : Ω — si $f \in \Omega(g)$, $\exists a \in \mathbb{R}^+$, $f_n \geq a \cdot g_n$ àpcr
- pire cas : O — si $f \in O(g)$, $\exists b \in \mathbb{R}^+$, $f_n \leq b \cdot g_n$ apcr
- les deux : Θ — si $f \in \Theta(g)$, $\exists a, b \in \mathbb{R}^+$, $a \cdot g_n \leq f_n \leq b \cdot g_n$ apcr

4) Avec des tableaux, l'éclatement ne pose pas de problème et

triés



peut être fait en temps constant. Par contre, la fusion implique de copier les données dans un nouveau tableau : temps $\Theta(n_1+n_2)$
espace $\Theta(n_1+n_2)$

La complexité totale est donc $\Theta(n \log n)$ en temps
$\Theta(n)$ en espace

Exo 3 : Tri rapide

On choisit un pivot x, on met à gauche tous les éléments ≤ x, à droite tous les éléments > x, et on applique le tri récursivement.

1)
```
Entree : tab= [7, 5, 9, 2, 1, 6, 4] i= -1
tab= [7, 5, 9, 2, 1, 6, 4] i= -1 j= 0
tab= [7, 5, 9, 2, 1, 6, 4] i= -1 j= 1
tab= [7, 5, 9, 2, 1, 6, 4] i= -1 j= 2
tab= [2, 5, 9, 7, 1, 6, 4] i= 0 j= 3
tab= [2, 1, 9, 7, 5, 6, 4] i= 1 j= 4
tab= [2, 1, 9, 7, 5, 6, 4] i= 1 j= 5
Sortie : tab= [2, 1, 4, 7, 5, 6, 9] i+1= 2
```
$[\underbrace{2, 1, 4}_{\leq 4}, \underbrace{7, 5, 6, 9}_{> 4}]$

Cette fonction ré-organise le tableau $tab[d \dots f]$ en prenant comme pivot le dernier élément. Tous les éléments plus petits sont placés avant le pivot, les plus grands après.

2) On note tab_j le tableau à la fin de la boucle j.

invariant :
- avant la boucle, le tab est intouché.
- les éléments de tab_j[debut .. fin] sont ceux de tab[debut .. fin] mais réorganisés, et les autres éléments sont intouchés
- tab_j[debut .. i+1] contient les éléments ≤ pivot
  tab_j[i+1 ... fin] ——— > pivot
- le pivot est dans tab_j(fin)

En sortie de boucle, on a $j = fin - 1$ car range(a, b) va de a inclus à b exclus

Donc tab_{fin-1} vérifie :
- les éléments hors [debut .. fin] sont intouchés
- tab[debut .. i+1] contient les ≤
  tab[i+1 -- fin] ——— >
- le pivot est à tab[fin]

Ensuite, on swap (intervertit) le pivot avec tab[i+1] et on retourne i+1 qui est donc l'index du pivot avec avant, les ≤ et après, les >.

3) Complexité : chaque tour de boucle ne fait que des opérations élémentaires : $\Theta(1)$. j varie de debut (inclus) à fin (exclus), donc $\boxed{\Theta(f-d)}$

4)

```
Appel tab= [7, 5, 9, 2, 1, 6, 4] debut= 0 fin= 6
Appel tab= [2, 1] debut= 0 fin= 1
Retour tab= [1, 2] debut= 0 fin= 1
Appel tab= [7, 5, 6, 9] debut= 3 fin= 6
Appel tab= [7, 5, 6] debut= 3 fin= 5
Retour tab= [5, 6, 7] debut= 3 fin= 5
Retour tab= [5, 6, 7, 9] debut= 3 fin= 6
Retour tab= [1, 2, 4, 5, 6, 7, 9] debut= 0 fin= 6
```

5) Montrer par récurrence forte que $\forall k \in [\![0, n-1]\!]$,

avec $k = f - d$, $P(k)$: "quicksort(tab, d, f) ~~renvoie~~ trie tab[d..f] par ordre croissant".

Base : $k = 0$.

Induction : $P(k+1)$? Le tableau est coupé en deux parties de taille $\leq k$ donc par hypothèse, quicksort les trie.

Conclusion : quicksort termine et est valide.

6) Pire des cas : le pivot place tous les éléments du même côté.

$$c(k) = \alpha k + c(k-1) = \dots = \alpha k + \alpha(k-1) + \dots = \alpha \frac{k(k-1)}{2} + c(0)$$
$$= O(k^2)$$

Meilleur cas : le pivot est pile au milieu à chaque récursion.

$$c(k) = \alpha k + 2c\left(\frac{k}{2}\right) = \alpha k + 2\left(\alpha \frac{k}{2} + 2c\left(\frac{k}{4}\right)\right)$$
$$= 2\alpha k + 4 \times c\left(\frac{k}{4}\right)$$
$$= 3\alpha k + 8 \times c\left(\frac{k}{8}\right)$$
$$\vdots$$
$$= \log_2 k \cdot k + k \times c(1)$$

$\Omega(k \log k)$