TD5

Exercice 1

Q1

. Il y a une comparaison des éléments du tableau par tour de boucle

. Il y a $k-1$ tours de boucle

Donc il y a $\underline{k-1 \text{ comparaisons}}$

Q2

1. Complexité du corps de boucle
2. On somme sur tous les tours de boucle

(Complexité en nombre de comparaisons)

1 → Soit $i \in [\![0, n[\![$.

Nombre de comparaisons dans le corps de boucle

= nombre de comparaisons dans $\underline{\text{Push}(T, n-i)}$

$= n - i - 1$

$$\sum_{x=0}^{N} x = \frac{N(N+1)}{2}$$

2 → soit $c(n)$ la complexité

$$c(n) = \sum_{i=0}^{n-1} (n-i-1) \overset{k=n-1-i}{=} \sum_{k=0}^{n-1} k = \frac{n(n-1)}{2}$$

(sinon $c(n) = \underbrace{\sum_{i=0}^{n-1} n}_{=n^2} - \underbrace{\sum_{i=0}^{n-1} i}_{=\frac{n(n-1)}{2}} - \underbrace{\sum_{i=0}^{n-1} 1}_{=n} \in \Theta(n^2)$)

Donc $\underline{c(n) = \Theta(n^2)}$.

Q3

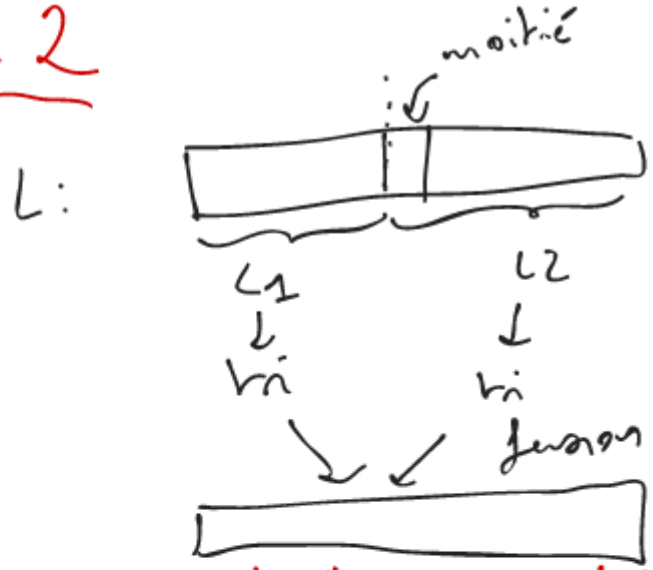
- Liste simplement chaînée : pas possible car dans la fonction push on a besoin d'accéder au prédecesseur d'un élément en plus de son successeur.

- Liste doublement chaînée : on peut directement accéder au prédecesseur et au successeur donc on peut implémenter la fonction Push telle quelle.

↳ c'est possible, et l'algorithme reste le même, les échanges et comparaison se font en même complexité que pour un tableau

Donc complexité en $\Theta(n^2)$.

Exercice 2



y'a cours

Rappel

fusion (L_1, L_2) : construit une seule liste à partir de listes L_1 et L_2 en prenant de manière récursive en premier $\min(L_1[0], L_2[0])$

$L_1 =$ [2, 8, 9, 10, 1, 12, 4, 5, 15, 3]

$L_2 =$ [9, 12, 1, 5]

fusion $(L_1, L_2) =$ [2, 8, 9, 10, 1, 12, 4, 5, 15, 3]

Intérêt de la fusion :

si L_1 et L_2 sont triées par ordre croissant alors fusion (L_1, L_2) est triée par ordre croissant

$L_1 = [1, 3, 5]$
$L_2 = [2, 7]$

fusion$(L_1, L_2) = [1, 2, 3, 5, 7]$

Q1

Au brouillon : faire l'arbre des appels récursifs

Brouillon

triFusionRec([7, 9, 2, 1, 6, 4, 3, 1])

Appel de TriFusionRec pour L = [7, 9, 2, 1, 6, 4, 3, 1]

[7, 9, 2, 1] | [6, 4, 3, 1]

[7, 9] | [2, 1]

[7] [9]

Renvoi de L = [7]

Appel [9]

Renvoi [9]

Renvoi [7, 9] fusion([7], [9])

Appel [2, 1]

Appel [2]

Renvoi [2]

Appel [1]

Renvoi [1]

Renvoi [1, 2]

Renvoi [1, 2, 7, 9] fusion([7, 9], [1, 2])

Arbre des appels

[7, 9, 2, 1] | [6, 4, 3, 1]

[7, 9, 2, 1]

[7, 9] [2, 1]

[7] [9] [2] [1]

idem

...

Renvoi [1, 3, 4, 6]

fusion

Renvoi de L = [1, 1, 2, 3, 4, 6, 7, 9]

→ c'est trié !

Q2 Par récurrence forte sur la taille de la liste.

Q3

liste chaînée

. L(:moitie) se construit en $\alpha_2 \times n$ opérations ($n \geq 2$)

α_2 constant

. fusion (L_1, L_2)

s'exécute en $\alpha_2 \times n$ opérations

Complexité de TriFusionRec :

$$c(n) = c_{\text{construction L(:moitie)}}(n) + c_{\text{fusion}}(n) + c\left(\frac{n}{2}\right) + c\left(\frac{n}{2}\right) + A$$

$A > 0$.

αn αn

$$\boxed{c(n) = \alpha . n + 2 \cdot c\left(\frac{n}{2}\right) + A} \quad (\alpha = \alpha_1 + \alpha_2)$$

$$c(1) = 1.$$

Quand on coupe une liste, la complexité est linéaire en la taille de la liste

Substitution

$$c(n) = \alpha n + 2\, c\left(\frac{n}{2}\right) + A$$

$$= \alpha n + \cancel{2}\alpha \frac{n}{\cancel{2}} + 2^2 c\left(\frac{n}{2^2}\right) + A + A$$

$$= \underbrace{\alpha n + \alpha n}_{3\alpha n} + \cancel{2^2}\alpha \frac{n}{\cancel{2^2}} + 2^3 c\left(\frac{n}{2^3}\right) + \underbrace{A + A + A}_{3A}$$

$$= 4\alpha n + 2^4 c\left(\frac{n}{2^4}\right) + 4A$$

$$= \dots$$

$$= k\alpha n + 2^k c\left(\frac{n}{2^k}\right) + kA$$

objectif de la substitution : se ramener à $c(1)$.

Donc, pour $k = \log_2(n)$

$X = 2^{\log_2(X)}$

On obtient $c(n) = \alpha . n . \log_2(n) + n\,c(1) + \log_2(n).A$

Donc $c(n)$ est en $\Theta(n \log_2(n))$ (on regarde que le terme dominant)

Bilan : Si $c(n)$ dépend de $c(\frac{n}{2})$ (ou similaire)

→ On calcule $c(n)$ par substitution jusqu'à $\log_2(n)$

---

(24)

tableau : découpage en moitié en temps constant

mais la fusion nécessite de créer un tableau de taille n, donc complexité $K \times n$.

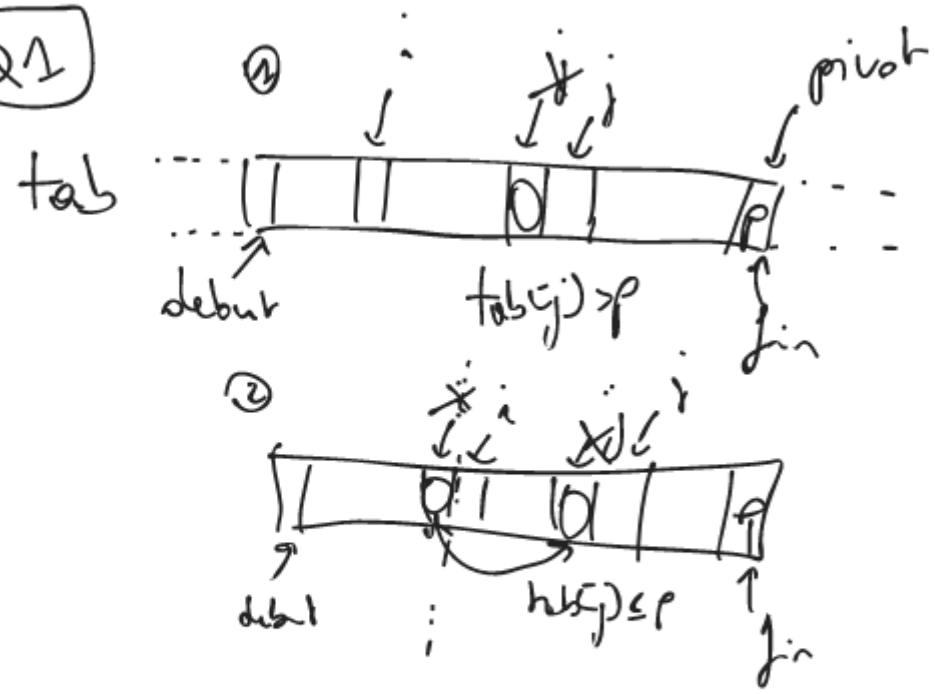
↳ Construction + fusion = complexité linéaire

↳ appels récursifs : toujours $2 \times c(\frac{n}{2})$

⇒ La complexité est la même, $\Theta(n \log_2(n))$

Exercice 3

Q1



partition([7, 5, 9, 2, 1, 6, 4], 0, 6)

debut = 0 ; fin = 6 ; pivot = 4

Entree : tab = [7, 5, 9, 2, 1, 6, 4] i = -1

7 > 4 : tab = [7, 5, 9, 2, 1, 6, 4] , i = -1 , j = 0

5 > 4 : tab = [7, 5, 9, 2, 1, 6, 4] , i = -1 , j = 1

9 > 4 : tab = [7, 5, 9, 2, 1, 6, 4] , i = -1 , j = 2

2 ≤ 4 : tab = [2, 5, 9, 7, 1, 6, 4] , i = 0 , j = 3

1 ≤ 4 : tab = [2, 1, 9, 7, 5, 6, 4] , i = 1 , j = 4

6 > 4 : tab = [2, 1, 9, 7, 5, 6, 4] , i = 1 , j = 5

⚠ for j in range(debut fin) : j va de debut à fin - 1

Donc on sort de la boucle

Sortie : tab = [2, 1, 4, 7, 5, 6, 9] , i + 1 = 2

Cette fonction :

→ réorganise tab[début...fin) de telle sorte que :
- entre début et l'indice du pivot, les valeurs sont ≤ pivot
- entre l'indice du pivot et fin, les valeurs sont > pivot

→ renvoie l'indice du pivot

Q2

Soit tab_j le tableau à l'itération j

Invariant de boucle

- tab_j[début...i] contient des valeurs ≤ pivot
- tab_j[i+1...j] contient des valeurs > pivot
- le pivot est dans tab_j[fin]

En sortie de boucle : j = fin - 1

et tab[début...fin] est réorganisé comme :
- tab[début...i) contient les valeurs ≤ pivot
- tab[i+1...fin-1) contient les valeurs > pivot
- tab[fin) contient le pivot

Validité :

Après l'échange entre les indices fin et i+1, on obtient bien le résultat énoncé en Q1.

Astuce : entre A et B inclus il y a $(B-A+1)$ nombres

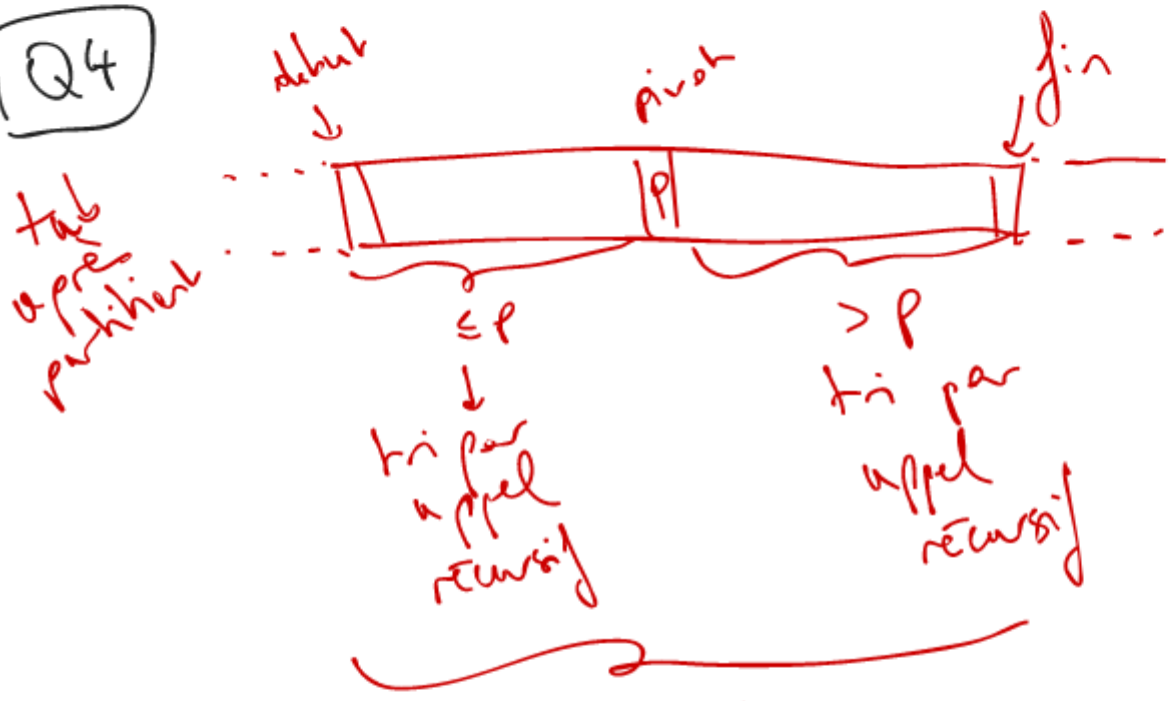
Q3

La fonction fait $fin - debut$ tours de boucle

donc elle fait $\alpha(fin-debut)+\beta$ opérations

Complexité $\Theta(fin-debut)$.

Q4



(tri = par ordre croissant)

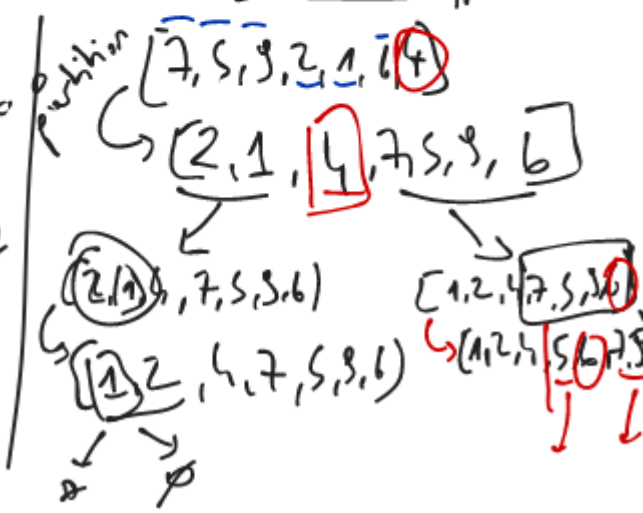
⚠ tab[debut:fin+1] = les valeurs entre debut et fin

* faire l'arbre des appels
* donner les affichages sans justifier

quicksort([7,5,9,2,1,6,4], 0, 6)

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

Nbre des appels



retour tab = [1,2,4,5,6,7,9] debut = 0, fin = 6

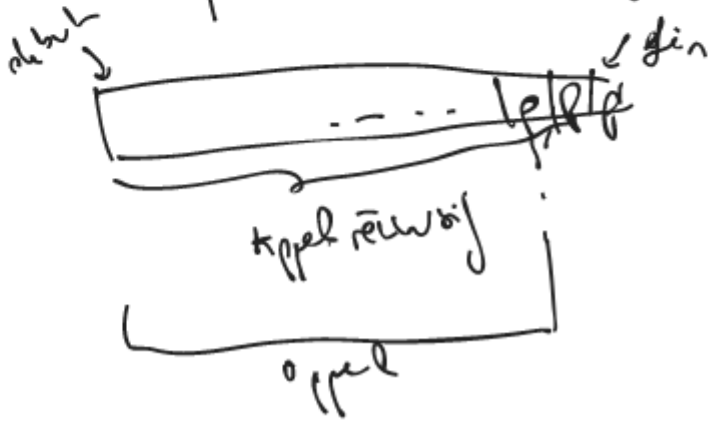
Q5 plus tard

Q6 Posons $m = fin - debut$

pire cas = O

meilleur cas = Ω

- pire cas : le pivot reste toujours à la fin



pire cas : gros appel récursif on enlève juste 1 élément

→ en tout on fait m appels récursifs

La complexité générale de la fonction est

$$c(m) = c_{partition}(m) + c_{\text{Appel récursif}}(m) + c_{\text{Appel récursif}}(m)$$

Rappel $c_{partition}(m) = \alpha m + \beta$

Dans le pire cas (celui avec le + d'appels récursifs)

$c_{\text{Appel récursif}}(m) = c(m-1)$ gros appel récursif (pire cas)

$c_{\text{Appel récursif}}(m) = c(0) = 0.$

Donc dans le pire cas:

$$c(n) = \alpha n + \beta + c(n-1)$$

substitution $\Rightarrow$

$$= \alpha n + \alpha(n-1) + \beta + \beta + c(n-2)$$

$$= \alpha n + \alpha(n-1) + \alpha(n-2) + \beta + \beta + \beta + c(n-3)$$

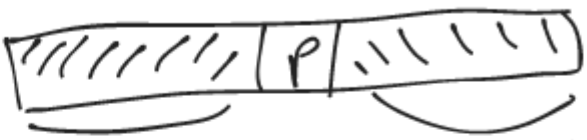
$$= \dots$$

$$= \alpha \left(\sum_{i=0}^{n} i \right) + n \times \beta + c(0)$$

$$= \alpha \frac{n(n+1)}{2} + n \times \beta$$

Donc dans le pire cas, la complexité est en $O(n^2)$ $\quad O((fin - debut)^2)$

- Dans le meilleur cas



A_1 (appel récursif 1) $\quad$ appel récursif 2) A_2

Dans le meilleur cas, les appels récursifs sur A_1 et A_2 se font sur des sous-tableaux les plus petits possible.
C'est-à-dire des tableaux de taille $n/2$
$\rightarrow$ le pivot est placé au milieu

Dans le meilleur cas : $c(n) = c_{partition}(n) + c\left(\frac{n}{2}\right) + c\left(\frac{n}{2}\right)$

Donc dans le meilleur cas :

$$c(n) = \alpha n + \beta + 2c\left(\frac{n}{2}\right)$$

$$= \alpha n + \cancel{2}\alpha\frac{n}{\cancel{2}} + \beta + 2\beta + 2^2 c\left(\frac{n}{2^2}\right)$$

$$= \alpha n + \alpha n + 2^2\alpha\frac{n}{2^2} + \beta + 2\beta + 2^2\beta + 2^3 c\left(\frac{n}{2^3}\right)$$

$$= \dots$$

Réflexe : on pose $k = \log_2(n)$

Abo $c(n) = k \times \alpha n + \beta \times \underbrace{\sum_{i=0}^{k-1} 2^i}_{\frac{1-2^k}{1-2} = 2^k - 1} + 2^k c\left(\frac{n}{2^k}\right)$

$$= \underbrace{\log_2(n) \times \alpha \times n} + \beta \times (n-1) + n \times c(1)$$

Donc dans le meilleur cas $c(n) = \Omega(n \times \log_2(n))$

Q5 On montre par récurrence forte sur $k \geq 0$ la propriété suivante

$P(k)$: Si $0 \leq d \leq f \leq n-1$ avec $k = f - d$, alors quickSort(tab, d, f) termine et trie tab[d f] par ordre croissant

Base Si $k = 0$, $d = f$, la fonction termine et tab[d f] est trié car ne contient qu'un élément

Induction Soit $k > 0$ fixé On suppose que pour tout $k' < k$ $P(k')$ est vérifiée

Mq $P(k)$ est vérifiée

Soit f et d tq $0 \leq d < f \leq n-1$ avec $k = f - d$

$k > 0$ donc $f > d$

partition(tab, d, f) réorganise tab et renvoie l'indice du pivot pivotPivot de telle sorte que

$\forall i \in \{d, \dots, pivotPivot - 1\}$ tab[i] $\leq$ tab[pivotPivot]

$\forall i \in \{pivotPivot + 1, \dots, f\}$ tab[i] $>$ tab[pivotPivot]

Donc l'élément tab[pivotPivot] est à sa place dans tab[d f]

De plus

- pivotPivot $- 1 < f$ donc pivotPivot $- 1 - d < k$, donc par HR sur pivotPivot $- 1 - d$, le premier appel récursif se termine et trie tab[d pivotPivot $- 1$] par ordre croissant
- De même, par HR sur $f - (pivotPivot + 1)$, le deuxième appel récursif termine et trie tab[pivotPivot $+ 1$. f] par ordre croissant

Donc, l'appel quickSort(tab, d, f) termine et trie tab[d f] par ordre croissant

Donc $P(k)$ est vérifiée

- Conclusion Par récurrence forte $\forall k \geq 0$ $P(k)$ est vérifiée

Donc la fonction termine et est valide.