22.03.2021 (présentiel)

EA 4 TD 7 : Quick select

Exercice 1

nb de comparaisons

1)

153	159	53	135	106	75	36	73

$k = 5$ pivot : 153
gauche : 53 135 106 75 36 73 → len = 6 — 6
droite : 159 — 1

rang_pivot = 6 + 1 = 7 > k = 5 — 1

53	135	106	75	36	73

$k = 5$ pivot : 53
gauche : 36 → len = 1 — 1
droite : 135 106 75 73 — 4

rang_pivot = 1 + 1 = 2 < k = 5 — 1

135	106	75	73

$k = 5 - 2 = 3$ pivot : 135
gauche : 106 75 73 → len = 3 — 3
droite : ∅

rang_pivot = 3 + 1 = 4 > k = 3 — 1

106	75	73

$k = 3$ pivot : 106
gauche : 75 73 → len = 2 — 2
droite : ∅

rang_pivot = 2 + 1 = 3 = k — 1

⇒ return pivot = 106

= 21 comparaisons

2)

106	153	159	53	135	75	36	73

$k = 5$ pivot : 106
gauche : 53 75 36 73 → len = 4 — 4
droite : 153 159 135 — 3

rang_pivot = 4 + 1 = 5 = k — 1

= 8 comparaisons

⇒ return 106

106	153	159	53	135	75	36	73

$k = 4$ pivot : 106
gauche : 53 75 36 73 → len = 4 — 4
droite : 153 159 135 — 3

rang-pivot = 4 + 1 = 5 > k = 4 — 1

53	75	36	73

$k = 4$ pivot : 53
gauche : 36 → len = 1 — 1
droite : 75 73 — 1

rang-pivot = 1 + 1 = 2 < k = 4 — 1

75	73

$k = 4 - 2 = 2$ pivot : 75
gauche : 73 → len = 1 — 1
droite : ϕ

rang-pivot : 1 + 1 = 2 = k — 1

return pivot : 75

= 13 comparaisons

3) Dans le meilleur cas : n (car il faut juste faire la partition)

Dans le pire cas : n^2 (on trie le tableau avec Quick-sort)

4) $A(n) = \text{partion}(n) + A\left(\frac{n}{2}\right)$

$\Rightarrow A(n) = \Theta(n) + A\left(\frac{n}{2}\right)$

Donc $A_n = \Theta(n)$

calcul : $n = 2^k$ $B(k) = A(2^k)$

$B(k) = 2^k + B(k-1)$

$\Rightarrow B(k) - B(k-1) = 2^k$

si on somme : $\sum_{i=1}^{k} (B(i) - B(i-1)) = \sum_{i=1}^{k} 2^i$

$B(n) - B(0) = 2^k - 1 = \Theta(n)$

Exercice 2

1) a)

1	3	6	2	4	8	5	7

pile :

7	5,7	5,7,8	4,5,7,8	2,4,5,7,8	2,4,5,6,7,8	2,3,4,5,6,7,8	1,2,3,4,5,6,7,8
5	8	4	2	6	3	1	
8	4	2	6	3	1		
4	2	6	3	1			
2	6	3	1				
6	3	1					
3	1						
1							

b) Pour montrer que le tri est valide, il faut montrer l'invariant de boucle suivant :
" à chaque tour de boucle :
- le tableau au sommet de la pile est trié (1)
- et la pile contient tous les éléments de T (2) "

initialisation :
La pile contient tous les éléments de T => (2) OK
Le tableau au sommet de la pile ne contient qu'un seul élément, donc il est trié => (1) OK

hérédité :
Montrons que ces hypothèses restent vraies après un tour de boucle.
Soit p la pile et $p[0]$ l'élément au sommet.

Par hypothèse, $p[0]$ est trié et la pile contient tous les éléments de T.

On fusionne $p[0]$ et $p[1]$ et on enlève $p[0]$ et $p[1]$ de p.

On ajoute le nouveau tableau en haut de la pile. On a bien remis tous les éléments qu'on avait enlevés sur la pile.

Et on a bien $p[0]$ trié car fusion($p[0]$, $p[1]$) a renvoyé un tableau trié.

Donc l'invariant est vrai.

Preuve de l'algo :

Au début, la pile contient tous les éléments de T.
À la fin de la boucle, la taille de la pile est 1, car à chaque tour la longueur diminue de 1.
Grâce à l'invariant, on sait que $p[0]$ est trié et que la pile contient tous les éléments de T.

La pile est de taille 1, donc $p[0]$ est un tableau trié qui contient tous les éléments du tableau de départ.

On a donc bien trié T.

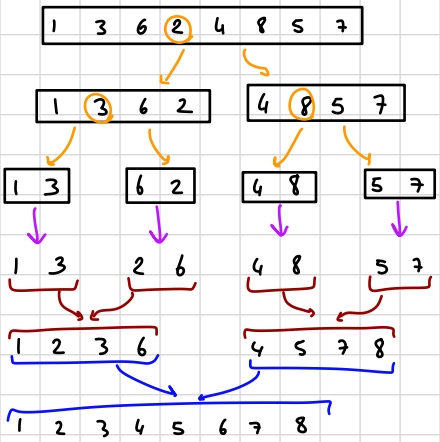
c) Complexité en espace : un tableau de taille n qui contient au plus n-1 tableaux de taille 1, et un tableau de taille n.

Complexité en temps : la taille de la pile diminue de 1 à chaque tour.
=> donc n tours de boucle
À chaque tour : fusion à un coût $\Theta(n)$

Donc $\Theta(n^2)$

2) Avec des indices, séparer la partie triée de la partie non-explorée du tableau.

3) a)



b) Dans une même ligne, on peut fusionner les tableaux dans l'ordre qu'on veut.

ex :

c)
```
TriItératifFusion(T):
    s = 1
    n = len(T)
    etape = 1
    while s < n:
        buffer = []
        p = 0
        q = s-1
        r = 2*s - 1
        while r < n:
            buffer.append((tab, p, q, r))
            p += 2s
            q += 2s
            r += 2s
        s *= 2
        etape += 1
        i = 0
        for xg in buffer:
```

4)
```
monotonies(T):
    m = []
    res = []
    for i in range(1, len(T)):
        if (T[i] < T[i-1]):
            res.append(m)
            m = []
        else:
            m.append(T[i-1])

        if i == len(T)-1:
            m.append(T[i])
            res.append(m)

    return res
```

5)
```
TriFusionNaturel(T):
    pile = monotonies(T)
```