UORAN1 / Epreuve finale 2024/2025 – M1/TC/ Algorithmique Avancée et Complexité / Durée 01h40

Exercice 1 (11 pts) : analyse des algorithmes .

1) (3p) Dérouler le tri par tas sur T=[0, 30, -10, 40].
2) Soit un algorithme *Algo* en $\theta(2^{k \cdot n})$, où k est une constante entière non nulle.
    a. (2p) Démontrer si *Algo* est en $\theta(2^n)$?
    b. (1p) Peut-on démontrer que *Algo* est en $\Omega(2^n)$?
3) Soit l'algorithme (TripleTriFusion) de tri fusion en partitionnant le tableau en trois sous-tableaux :
    a. (2p) Démontrer que l'équation récurrente de TripleTriFusion est de la forme : $T(n) = a\,T(n/a) + b \times n$, où a et b sont des constantes à déterminer.
    b. (2p) Résoudre l'équation récurrente $T(n) = a\,T(n/a) + b \times n$
    c. (1p) Expliquer l'étape de fusion de TripleTriFusion et montrer si l'algorithme TripleTriFusion est plus performant que TriFusion ?

RAPPEL :

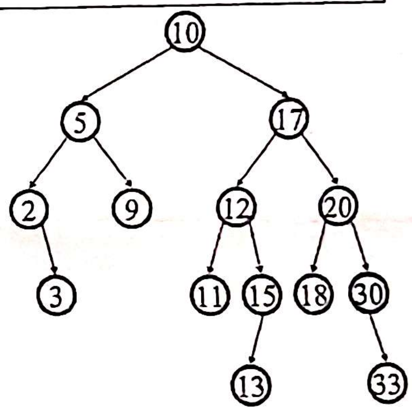
```
Algorithm TriFusion(A, p, r):
  if p < r then
     q ← ⌊(p + r)/2⌋
     Tri-Fusion(A, p, q)
     Tri-Fusion(A, q + 1, r)
     Fusionner(A, p, q, r)
  endif
```

```
Algorithm TripleTriFusion(A, p, r):
  if p < r then
     q1 ← p + ⌊(r - p)/3⌋
     q2 ← p + ⌊(r - p)*2/3⌋
     Tri-Fusion(A, p, q1)
     Tri-Fusion(A, q1 + 1, q2)
     Tri-Fusion(A, q2 + 1, r)
     Fusionner(A, p, q1, q2)
     Fusionner(A, p, q2, r)
  endif
```

Exercice 2 (4 pts) : arbres de recherche

Soit un arbre binaire de recherche des clés [2, 3, …, 33] :

1) (0.5p) Cet arbre est-il un AVL ?
2) (1.5p) Illustrer toutes les étapes pour supprimer la clé 17.
3) (2p) L'arbre obtenu après suppression, est-il un AVL ? Si « non AVL », expliciter toutes les étapes pour rétablir la structure AVL ?



Exercice 3 (4 pts) : programmation dynamique

On considère le problème où l'on doit rendre la monnaie pour x DA avec le minimum possible de pièces de k types de pièces $[c_1, c_2, \dots, c_k]$.
Expliciter l'algorithme par programmation dynamique qui donne le nombre optimal de chacune des pièces $[c_1, c_2, \dots, c_k]$. (Exemple illustratif : soit 10 DA avec des pièces [1, 2, 5, 7]. L'algorithme doit afficher 2 pièces de 5 DA.)

Exercice 4 : cours / (1p) Expliquer les classes de complexité P, NP, NPC ?

RAPPEL :

Théorème 2 (Théorème général). *Soient $a \geq 1$, et $b > 1$ deux constantes, soit $f(n)$ une fonction, et soit $T(n)$ définie pour les entiers positifs par la récurrence*

$$T(n) = aT(n/b) + f(n),$$

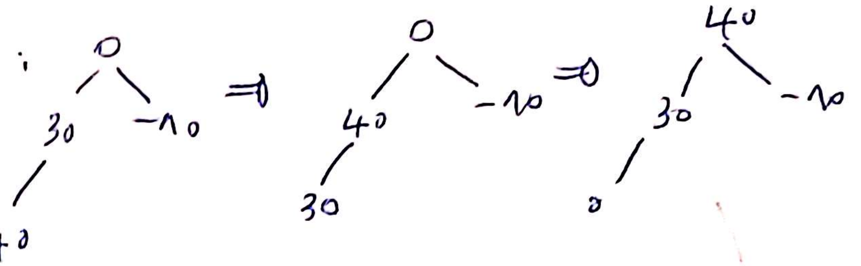
où l'on interprète n/b soit comme $\lfloor n/b \rfloor$, soit comme $\lceil n/b \rceil$. $T(n)$ peut alors être bornée asymptotiquement comme suit :

1. *Si $f(n) = O(n^{\log_b a - \epsilon})$ pour une certaine constante $\epsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$.*
2. *Si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$.*
3. *Si $f(n) = \Omega(n^{\log_b a + \epsilon})$ pour une constante $\epsilon > 0$, et si $af(n/b) \leq cf(n)$ pour une constante $c < 1$ et tous les n suffisamment grands, alors $T(n) = \Theta(f(n))$.*

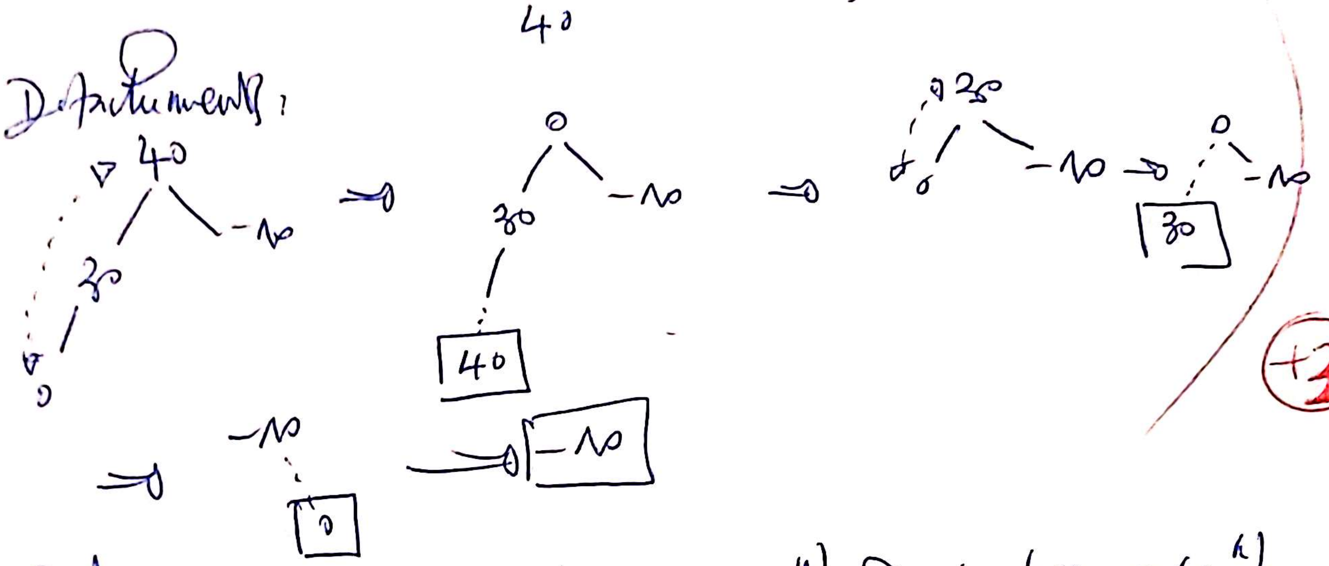
AAC — EF - 2024/2025

Exercice 1

Q1) Tri par tas : construction tas



Détachements :



+3

Q2) a) Est-ce que $2^{k\cdot n} = \Theta(2^n)$? soit $2^{k\cdot n} = O(2^n)$ et $2^{k\cdot n} = \Omega(2^n)$.

Soit $2^{k\cdot n} = O(2^n)$?

$\exists\, c, n_0 > 0,\ 2^{k\cdot n} \leq c\cdot 2^n,\ \forall n \geq n_0$.

$\Rightarrow 2^n \cdot 2^{(k-1)n} \leq c\cdot 2^n \Rightarrow 2^{(k-1)\cdot n} \leq c$.

$\Rightarrow \begin{cases} k = 1 \Rightarrow 2^n = \Theta(2^n). \\ k > 1 \Rightarrow 2^{(k-1)n} \leq c \Rightarrow \text{impossible car} \end{cases}$

aucune constante c ne peut être supérieure à $2^{(k-1)n}$ qui tend vers l'infini.

+2

D'où $2^{k\cdot n} \neq \Theta(2^n)$.

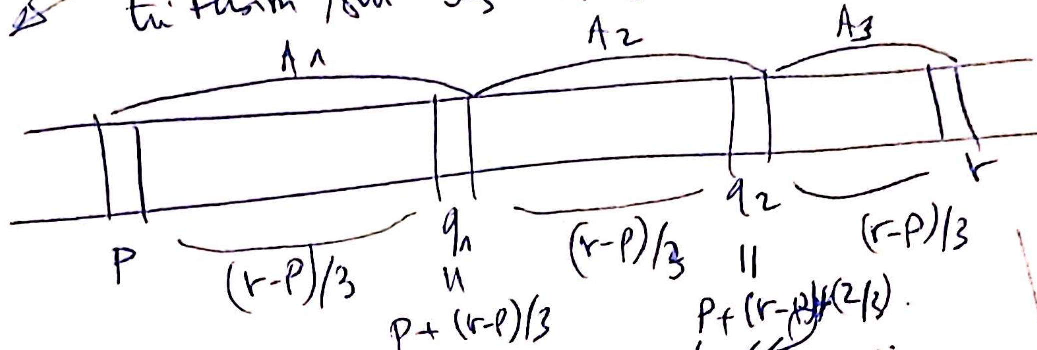
b) $2^{k\cdot n} = \Omega(2^n)$ car $\forall k, n \geq 0$, $2^{kn} \geq 2^n$

D'où $2^{kn} = \Omega(2^n)$.

+1

Ex 1) Q3) TripleTri-fusion fait appel trois fois à tri-fusion sur les trois tiers du tableau

a)



si Tri-fusion = TripleTri fusion

On partitionne A en trois sous tableaux :
$A_1 = A[p \, ... \, p+(r-p)/3]$, $A_2 = A[p+(r-p)/3+1, \, ..., \, p+(r-p)*2/3]$,
$A_3 = A[p+(r-p)*\frac{2}{3}+1, \, ..., \, r]$.
Chacun des trois sous tableaux A_1, A_2, A_3 est de taille $(r-p)/3$.

Après division :
— On fait appel à tri-fusion trois fois sur chacun des tiers.
— Fusionner deux fois.

D'où $T(n) = 3 \cdot T(n/3) + \frac{2}{3} \cdot n + n$

$$T(n) = 3 \cdot T(n/3) + \frac{5}{3} \cdot n .$$

+2

+1,5 m

b) Résolution de l'équation récurrente :

$$T(n) = a \cdot T(n/4) + b \cdot n.$$

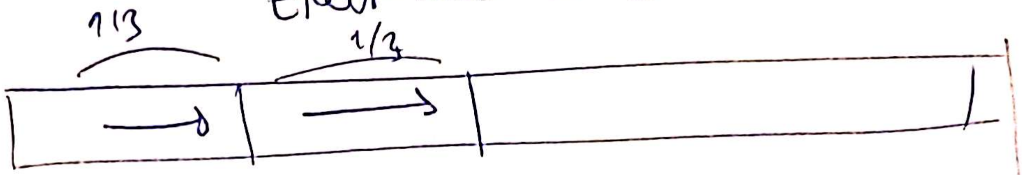
Cas 1) $b \cdot n = O(n^{\log_a a - \epsilon})$ $\Rightarrow b.n = O(n^{-\epsilon})$
$\Rightarrow$ Impossible.

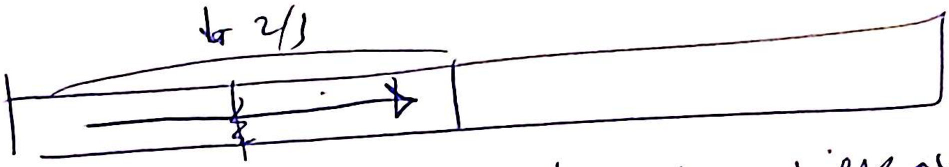
Cas 2) $b \cdot n = \Theta(n^{\log_a a}) = \Theta(n) \longrightarrow$ Evident
$\Rightarrow T(n) = \Theta(n \cdot \log(n))$

EX1/ Q3.c)

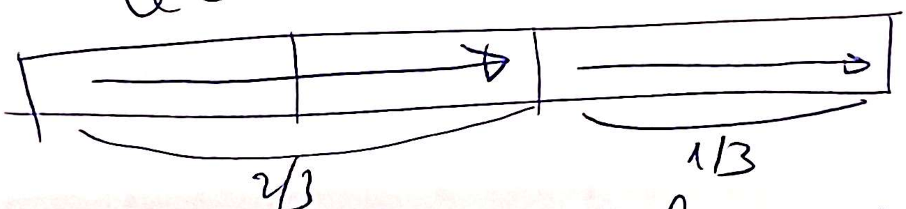
Si Tri-Fusion ≡ TripleTriFusion

↳ l'algorithme fusionne le premier tiers avec le $2^{ième}$ tiers.





↳ puis fusion les deux tiers avec le dernier tiers.



+1

TripleTriFusion est aussi performant que TriFusion, car d'après b) l'algorithme est toujours en $\Theta(n \cdot \log(n))$.

Q3 a) Une deuxième réponse si Tri-fusion = tri fusion. Unique ~~ment~~ au premier appel, on partitionne en trois tiers

$$\Rightarrow T(n) = 3 * T'(n/3)$$

$$\rightarrow \text{où } T'(n) = 2 . T'(n/2) + n.$$

$$a = 2, \ b = 1.$$

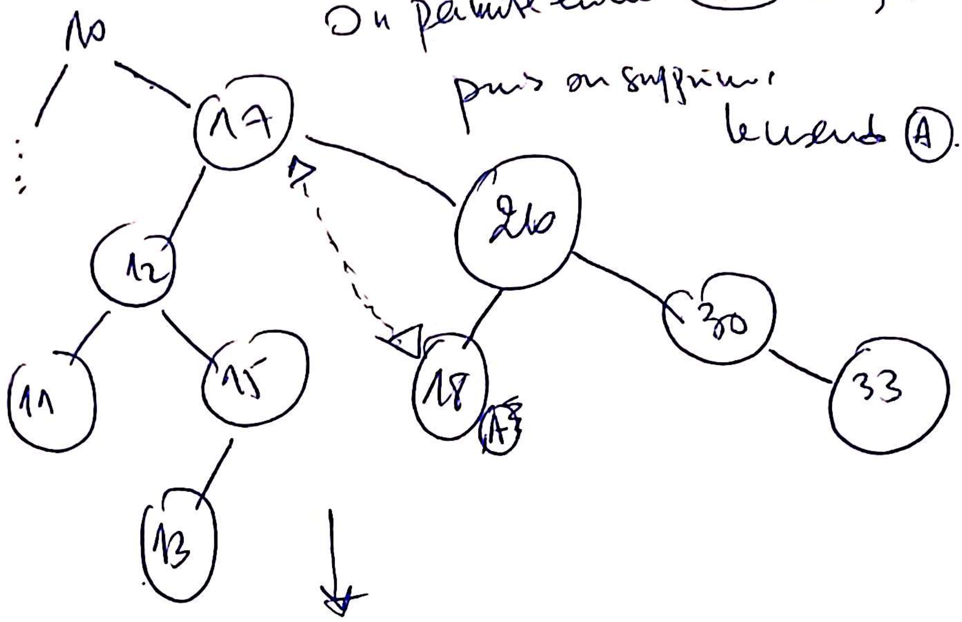
$$\Rightarrow T'(n) = \Theta(n \cdot \log(n)).$$

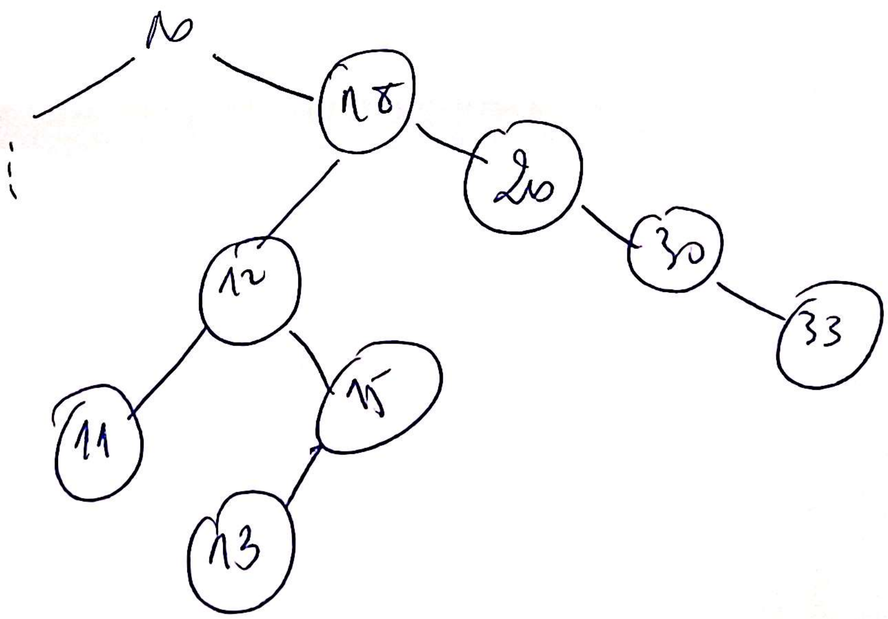
Exercice 2

Q1) C'est un AVL car la différence de profondeur de tous les noeuds est au plus 1 en valeur absolue.

+0,5

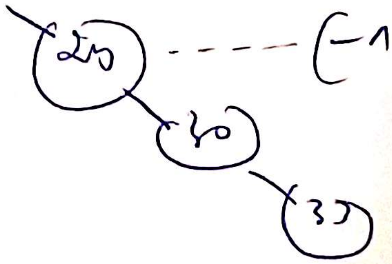
Q2) On permute entre 17 et son successeur 18 puis on supprime le noeud A.





+1,5

Q3) l'arbre n'est pas AVL au niveau du noeud 20

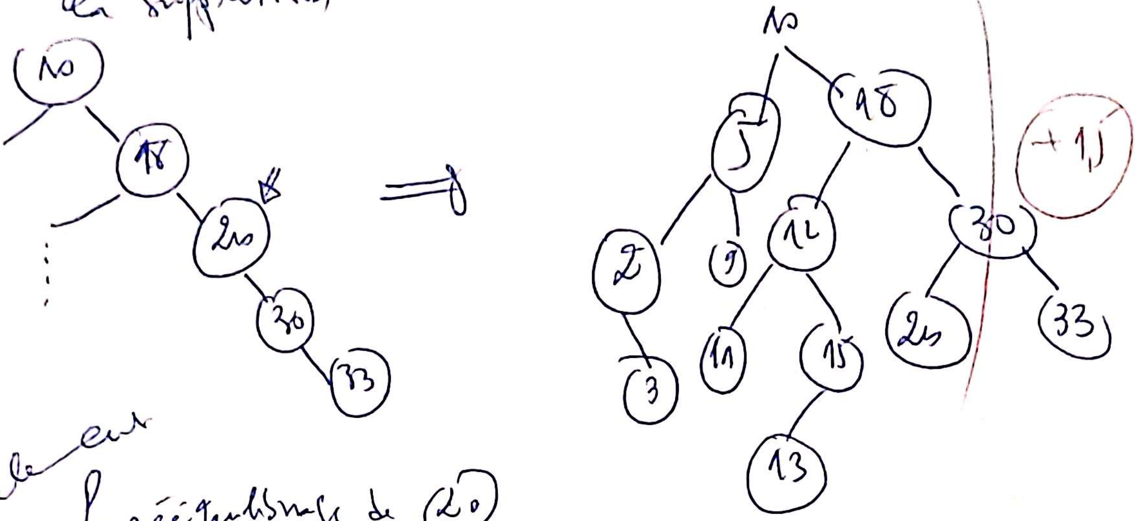


$(-1) - 1 = -2$ plus grande que 1 en valeur absolue.

+0,5

Q2 | Q3) suite.

Pour rétablir la propriété AVL, il suffit d'équilibrer les nœuds sur le chemin de retour dans la suppression



Finalement

Le rééquilibrage de 20 a été suffisant pour rééquilibrer tout l'arbre.

Exercice 3 | Même réponse du TD avec ajout des instructions pour garder trace des pièces optimales

---

```
⋮

Comb = ["", ..., ""] (X+1 fois).

Algorithm rendre_Monnaie_mem (X, mem, Comb):
  ⋮
  if nb < maxi then
      maxi ← nb
      mem[X] ← nb
      Comb[X] ← "Comb[X-P[i]]"
                 || "P[i]";
  end if
  ⋮
```