Nom : **Prénom :** **Groupe :**

Exercice 1 (9p)

Soit l'algorithme FCT :

a) **(1p) Soit un tableau T = [3, 6, 8, 2]. Dérouler l'appel FCT(T, 1, 4) en traçant l'arbre des appels récursifs.**
b) **(2p) Que fait l'algorithme FCT ?**
c) **(2p) Formuler l'équation récurrente caractérisant la complexité de cet algorithme.**
d) **(2p) Calculer la complexité de FCT en exploitant le théorème 2.**
e) **(2p) Démontrer si on peut trouver un algorithme meilleur que FCT au niveau complexité ?**

```
Algorithme FCT(T[1..n], d, f)
if (d >= f) then return d
else
        m := PartieEntière( (d + f)/2 )
        i := FCT(T, d, m)
        j := FCT(T, m+1, f)
        if (T[i] > T[j]) then return i ;
        else return j ;
        endif
endif
```

Nom : **Prénom :** **Groupe :**

Exercice 2 (6p)

Soit le tableau *T* = [40, 20, 18, 15, 8, 3].

a) **(4p) Dérouler le tri par tas sur le tableau *T*.**
b) **(1p) Ecrire l'algorithme *ConstruireTas.***
c) **(1p) Etudier la complexité de l'algorithme *ConstruireTas*.**

Nom : **Prénom :** **Groupe :**

Exercice 3 (5p)

a) **(1p) Dresser un arbre de recherche AVL des données [6, 11, 26, 28, 2, 3].**
b) **(1p) Justifier l'intérêt qu'un arbre soit de type AVL.**
c) **(3p) Donner l'algorithme récursif de recherche d'un successeur dans un arbre AVL.**

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*

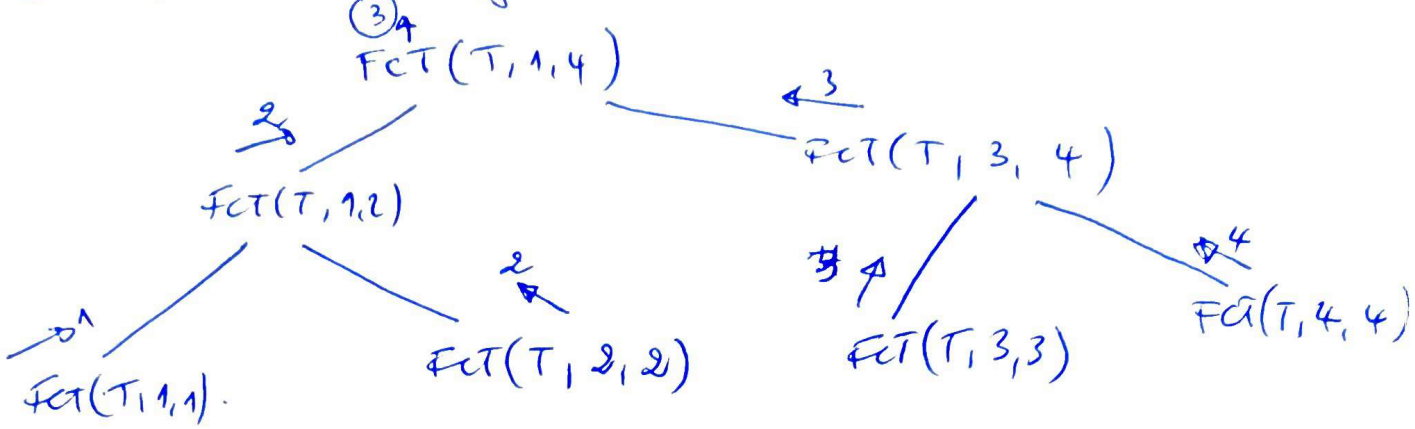
$$T(n) = aT(n/b) + f(n),$$

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 $T(n) = \Theta(f(n))$.*

Exo 2

2a) Déroulement de l'algorithme FCT(T,1,4).



L'algorithme FCT retourne à la fin l'indice 3.

2b) L'algorithme FCT retourne l'indice du plus grand élément dans le tableau T[1..n].

2c) L'équation récurrente de FCT :

Soit la fonction T la complexité de FCT sur un tableau de données de taille n.

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ T(n/2) + T(n/2) + \Theta(1) & \text{si } n > 1. \\ = 2T(n/2) + \Theta(1). \end{cases}$$

2d) Résolution de l'équation récurrente :

Dans le théorème 2, nous considérons l'équation $T(n) = aT(n/b) + f(n)$

où $a = 2$, $b = 2$, $f(n) = 1$.

Soit le cas 1 : est-ce que $1 = O(n^{\log_2^2 - \epsilon})$?

$1 = O(n^{1-\epsilon})$ avec $\epsilon > 0$

on peut prendre $\epsilon = \frac{1}{2}$, ainsi $1 = O(n^{1/2})$ qui est vérifié si $\exists n_0, c$, avec $1 \leq c \cdot n^{1/2}$, avec $n \geq n_0$.

on peut prendre $c = 1$ et $n_0 = 1$.

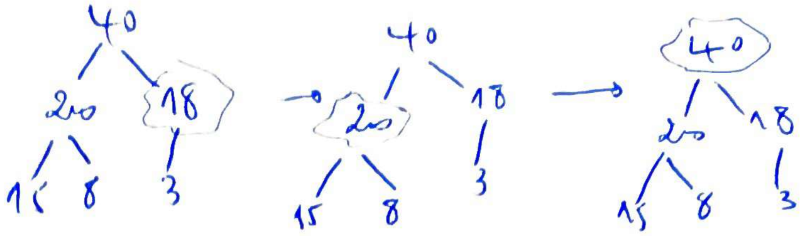
On peut donc poser $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

AAC — Février 2022.

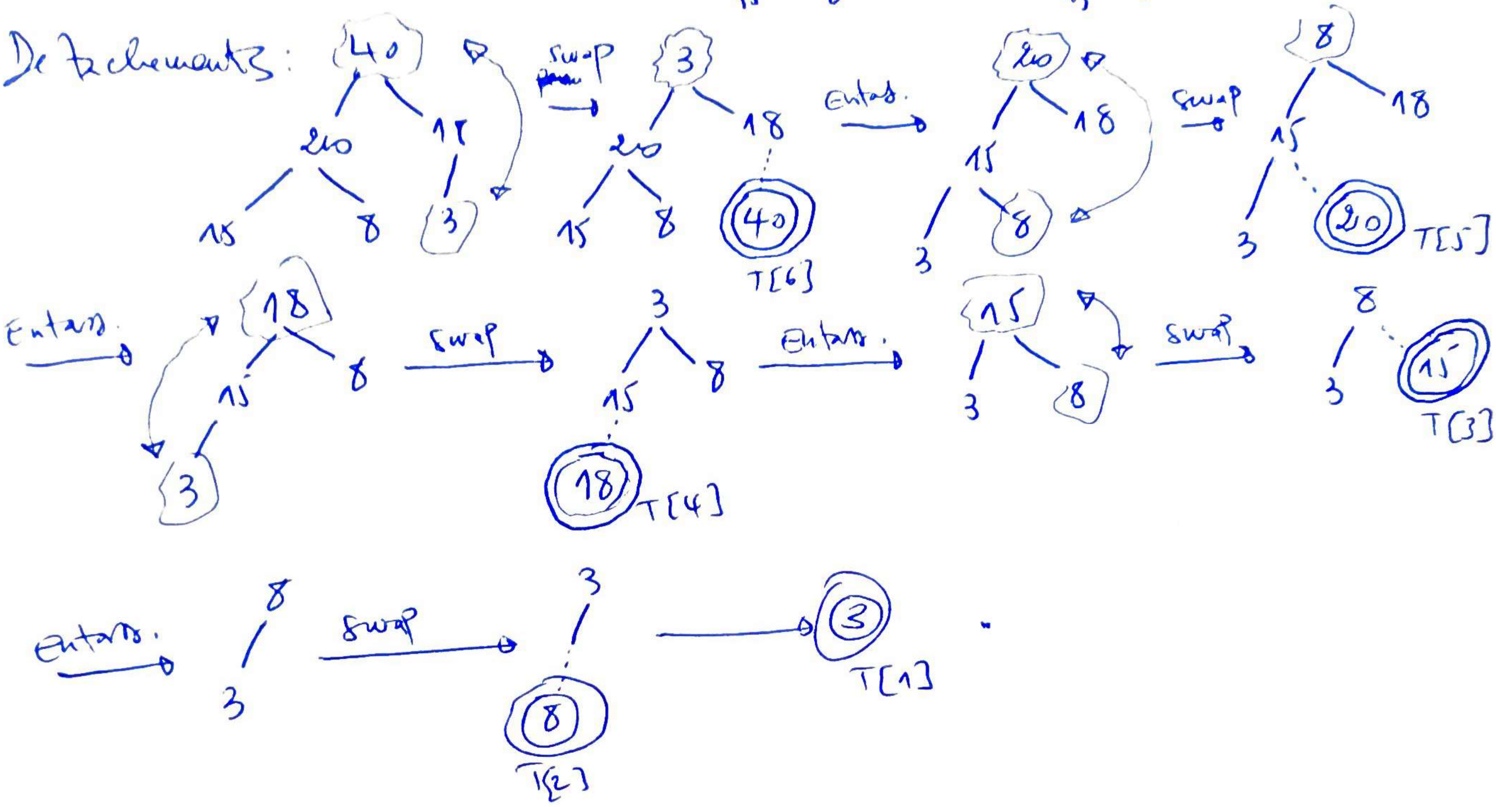
Exercice 1)

Q a) Déroulement.

ConstruireTas :



Détachements :



Q b) Algorithme ConstruireTas :

```
Algorithme ConstruireTas( A[1..n] ).
  for i ← ⌊n/2⌋ to 1 do
     Entasser(A, i)
  end for
```

Q c) L'algorithme Entasser est en $O(\log(n))$.

Au pire ConstruireTas est en $O(n/2 \cdot \log(n))$

c-à-d $O(n \log(n))$.

On peut démontrer que ConstruireTas est en $\Theta(n)$.

Exercice 2 (Suite)

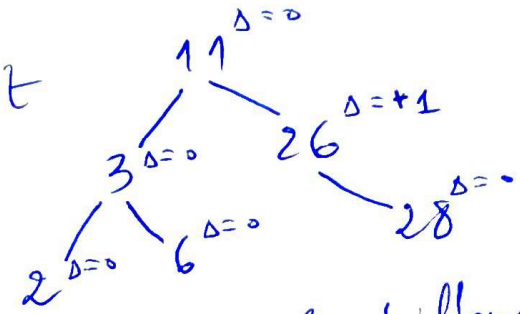
2e) Preuve si on peut mieux faire que FCT?

On ne peut pas, car si on peut faire moins que n comparaisons, il y aura au moins un élément du tableau qui ne sera pas comparé aux autres, rendant l'algorithme Correct.

D'où la Contradiction.

Exercice 3

Q a) Soit



L'arbre est un arbre AVL car la différence Δ de hauteur entre le sous-arbre gauche et le sous-arbre droit est au plus 1.

Q b) L'intérêt d'un arbre AVL : La structure d'un AVL garantie une hauteur de l'arbre de recherche en $h = \Theta(\lg(n))$, c'est-à-dire un arbre équilibré Comme tous les 7 algorithmes sont en $O(h)$, alors la structure AVL garantit une performance optimale.

Q c)

```
AVL-successeur(x)
    if droite(x) ≠ nil then
        return arbre-min(droite(x))
    end if
    y ← père(x)
    While (y ≠ nil et x = droite(y)) do
        x ← y
        y ← père(y)
    end While
    return y
```