



EA4 – Éléments d'algorithmique
Contrôle du 16 mars 2023 – Sujet A
Durée : 1 heure 15

*Aucun document autorisé
Appareils électroniques éteints et rangés*

Nom :

Prénom :

Numéro :

Groupe :

Exercice 1 :

On considère l'algorithme `foo` ci-contre.

Dessiner ci-dessous l'arbre des appels récursifs provoqués par l'appel `foo([4, 3, 2, 1], 0, 4)` : pour chaque appel, indiquer le contenu de `T` au début *et* à la fin de l'appel (sauf s'il est inchangé), en encadrant le sous-tableau concerné.
En cas d'appels équivalents, vous pouvez ne dérouler que le premier.

```
def foo(T, deb, fin) :  
    if fin-deb <=1 : return  
    m = (deb+fin)//2  
    foo(T, deb, m)  
    foo(T, m, fin)  
    if T[m-1] > T[fin-1] :  
        T[m-1], T[fin-1] = T[fin-1], T[m-1]  
    foo(T, deb, fin-1)
```

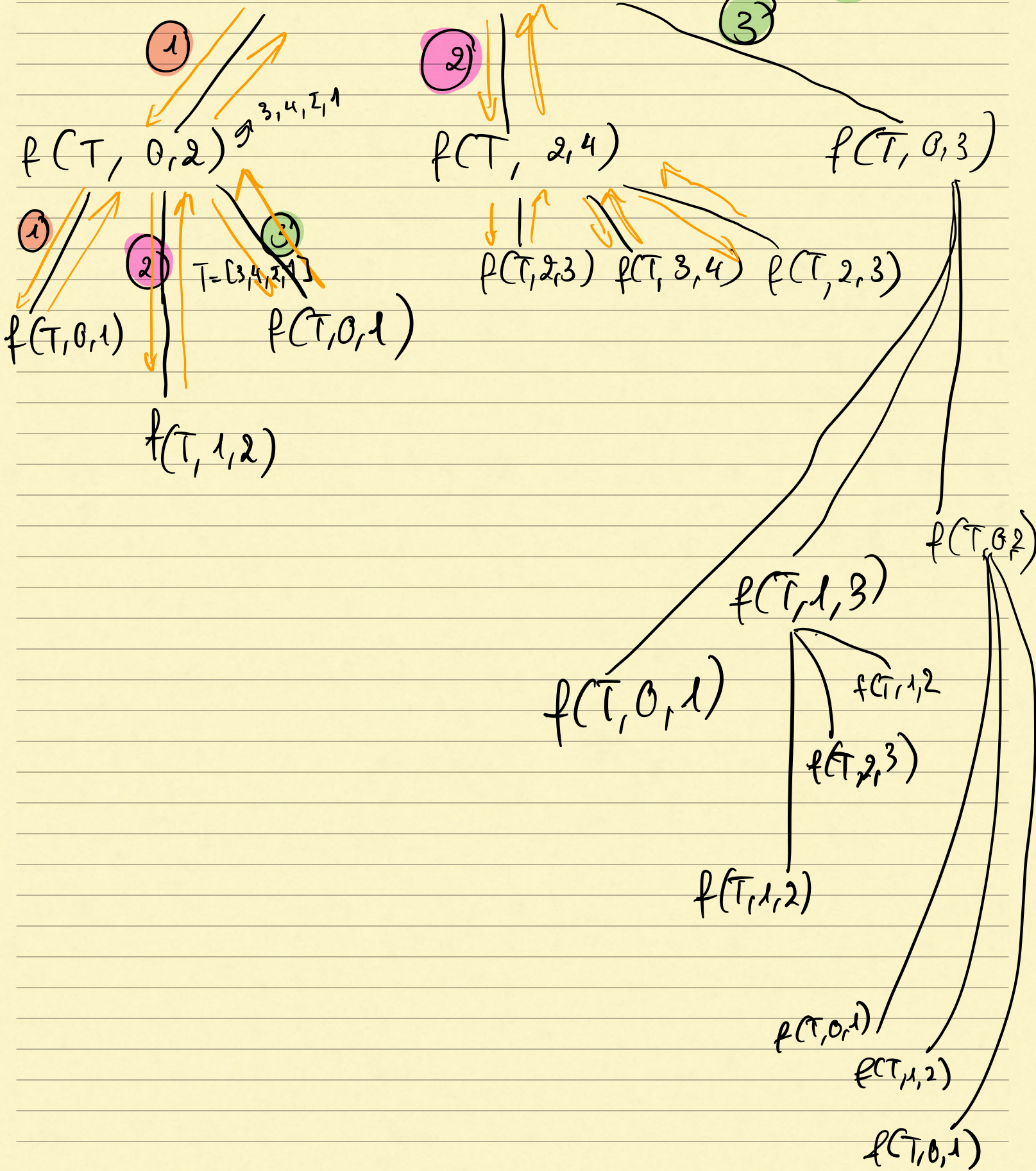
Que fait `foo` ? Justifier. _____

Soit $C(n)$ le nombre de comparaisons effectuées par `foo(T)` si `T` est un tableau de longueur n .
Quelle relation de récurrence $C(n)$ satisfait-elle ?

Ex 1

$$f([4, 3, 2, 1], 0, 4)$$

```
def foo(T, deb, fin) :
    if fin-deb <=1 : return
    m = (deb+fin)//2
    foo(T, deb, m) ①
    foo(T, m, fin) ②
    if T[m-1] > T[fin-1] :
        T[m-1], T[fin-1] = T[fin-1], T[m-1]
    foo(T, deb, fin-1) ③
```



$P(n)$: f trie les tableaux si $\text{fin} - \text{deb} \leq n$

Montrons par récurrence que $P(n)$ est vraie pour tout $n \in \mathbb{N}$

Initialisation $n = 0$ ok

Hérédité: Supposons que $P(n)$ est vraie.

Montrons que $P(n+1)$ est vraie

$$\begin{aligned} \text{fin} - \text{deb} &= n + 1 & 1. \text{ } m - \text{deb} &\leq \frac{\text{fin} + \text{deb}}{2} - \text{deb} \\ m & & &= \frac{\text{fin} - \text{deb}}{2} \leq \frac{n+1}{2} \\ & & &\leq n \end{aligned}$$

$$\begin{aligned} 2. \text{ } \text{fin} - m &\leq \text{fin} - \frac{\text{fin} + \text{deb}}{2} \\ &= \frac{\text{fin} - \text{deb}}{2} + \frac{1}{2} \\ &\leq \frac{n}{2} + 1 \leq n \end{aligned}$$

$$3. \text{ } \text{fin} - 1 - \text{deb} \leq n$$

Après les 2 premiers appels récursif, le maximum de $T[\text{deb} : \text{fin}]$ est soit $\bar{e} T[m-2]$

Soit $\bar{a} T[\text{fin}-1]$

Donc après le "j", il est dans la case $T[\text{fin}-1]$. Après le 3^e appel, on a

$T[\text{deb} : \text{fin}-1]$ est traité, et $T[\text{fin}-2] \leq T[\text{fin}-1]$

$$C(n) = \begin{cases} 0 & \text{si } n \leq 1 \\ 2 C\left(\frac{n}{2}\right) + 1 + C(n-1) \end{cases}$$

Cette relation implique que, $\forall n \geq 2, C(n) \geq \sum_{i=0}^{\lfloor n/2 \rfloor} C(i)$. (inégalité très large, mais suffisante)

En déduire par récurrence sur k la propriété suivante :

(hors-barème) « pour tout entier $k \geq 0$, il existe $\alpha_k > 0$ tel que $\forall n \geq 2, C(n) \geq \alpha_k n^k$. »

Que pensez-vous de l'efficacité de foo? $\forall n, C(n) = \Omega(n^k)$

Exercice 2 :

Soit T un tableau de n éléments comparables, par exemple des entiers positifs. On dit que T possède un *minimum local en position i* si $T[i] \leq T[i-1]$ et $T[i] \leq T[i+1]$ (donc en particulier, il faut que $n \geq 3$ et $0 < i < n-1$).

Entourer les 5 minima locaux du tableau $T = [9, 7, 7, 2, 1, 3, 7, 5, 4, 7, 3, 3, 6]$.

Soit $n \geq 3$. Tout tableau T de longueur n possède-t-il un minimum local? Justifier.

Non : $[1, 2, 3]$

On suppose dorénavant que T satisfait la propriété : $T[0] \geq T[1]$ et $T[n-2] \leq T[n-1]$. Justifier que sous cette hypothèse, T possède au moins un minimum local.

$2 \times T[0] \geq T[1]$
 $T[2] \geq T[1]$
 Initialisation $n=3 \rightarrow T[1]$ min local
 Hérité T de taille $n+1$ avec $T[0] \geq T[1]$
 $T[n] \geq T[n-1]$

$P(n)$: Si $T[0] \geq T[1]$
 et $T[n-1] \geq T[n-2]$
 alors T admet 2 un min local

Décrire un algorithme *le plus efficace possible* dans le pire cas pour déterminer un minimum local d'un tel tableau.

Quelle est sa complexité ?

Exercice 3 :

Dans cet exercice, on représente des ensembles par des tableaux *sans doublon*, et on considère la fonction foo suivante :

```
def foo(E, F) :
    res = E[:]
    for f in F :
        if f not in E : res.append(f)
    return res
```

Que calcule foo(E, F) ?

union E, F

Quelle est sa complexité dans le pire cas si E et F sont supposés de même longueur n ? Justifier.

$\Theta(n^2)$

Supposons maintenant que les ensembles sont représentés par des tableaux *triés* (et toujours sans doublon). Proposer un algorithme le plus efficace possible pour effectuer le même calcul.

def union_triee(E, F)

$n = \text{len}(E)$ $m = \text{len}(F)$ $res = []$

$i, j = 0, 0$

while $i < n$ and $j < m$

if $E[i] < F[j]$:

res.append(E[i])

$i += 1$

else if $E[i] == F[j]$

res.append(E[i])

else : $E[i] > F[j]$

res.append(F[j])

$j += 1$

res += E[i:]

res += F[j:]

return res

Quelle est sa complexité ?

Exercice 4 :

On s'intéresse au problème suivant : étant donné une liste L de nombres (non nécessairement entiers) de longueur n , déterminer le *vainqueur* de L , *i.e.* l'élément de L qui y apparaît le plus de fois (ou l'un quelconque d'entre eux, en cas d'égalité).

Décrire un algorithme naïf permettant de résoudre ce problème sans modifier la liste L , et avec mémoire auxiliaire constante.

Quel est la complexité (en temps) de cet algorithme ? Justifier.

Comment résoudre ce problème avec une complexité (dans le pire cas) strictement meilleure ?

Quelle est la complexité de cette méthode ?