EA3
Partiel du jeudi 2 novembre 2017

Durée : 1 heures 30
Aucun document autorisé
Appareils électroniques éteints et rangés

Préliminaires : *Ce sujet est constitué de 3 exercices qui peuvent être traités dans l'ordre de votre choix. Le barème est donné à titre indicatif. Il est conseillé de lire l'intégralité du sujet avant de commencer. Il est bien entendu préférable de ne faire qu'une partie du sujet correctement plutôt que de tout bâcler.*

Exercice 1 : un peu de lecture (4 points)

```
1 whatRUlooking4(T : tableau de n entiers, l : entier, r: entier){
2     if l=r { return T[l] }
3     m <- whatRUlooking4(T, l+1, r)
4     if T[l] > m { return T[l] }
5     return m
6 }
```

1. Exécuter l'algorithme `whatRUlooking4` sur le tableau `T={4, 22, 7, 9}`, `l=0` et `r=3` en déroulant la pile d'exécution. Il faut donc que toutes les valeurs et appels empilés apparaissent.
2. La récursion de l'algorithme `whatRUlooking4` est-elle terminale ? Justifier.
3. Écrire un algorithme itératif qui produit le même résultat que `whatRUlooking4`.

Exercice 2 : multiplicité (6 points)

1. Ecrire un algorithme `ppdiv` qui étant donné un nombre n strictement supérieur à 1, retourne sont plus petit diviseur strictement supérieur à 1. Par exemple `ppdiv(15)` retourne 3 et `ppdiv(17)` retourne 17.
2. On souhaite maintenant trier les éléments d'un tableau de nombres strictement supérieurs à 1 suivant leur plus petit diviseur, puis par ordre croissant. On appelle ce tri le *tri multiple*. Par exemple le tableau `T={4, 9, 12, 3, 21, 6, 8, 14, 10, 7, 18}` après un tri multiple deviendra `T={4,6,8,10,12,14,18,3,9,21,7}`.

   En vous appuyant sur le tri par insertion, écrire un algorithme `tri_mult` qui étant donné un tableau de nombres strictement supérieurs à 1, effectue le tri multiple en place.

Exercice 3 : sous-somme (10 points)

1. Etant donné un tableau de `n` entiers `T` et un entier `x`, écrire un algorithme quadratique (complexité en temps en n^2) `estSousSomme` qui retourne un couple `(a,b)` où `a` et `b` sont des éléments de `T` tels que `a+b=x`. Si deux tels entiers n'existent pas, l'algorithme retourne « - ».

2. Donner le nombre de comparaisons effectuées pour deux tableaux de tailles respectives 3 et 12 dans le cas pire, en précisant quel est le cas pire.

3. On donne maintenant l'algorithme suivant :

```
1  quoi(T:tableau de n entiers, x: entier){
2      trier(T)
3      a <- 0
4      b <- n-1
5      while( a < b ){
6          if T[a] + T[b] = x { return (T[a],T[b]) }
7          if T[a] + T[b] < x { a <- a+1 }
8          else{ b <- b-1 }
9      }
10     return -
11 }
```

a. En supposant que la fonction `trier(T)` trie le tableau `T` dans l'ordre croissant, exécuter l'algorithme sur le tableau `{15, 8, 17, 4}` pour `x = 20`, puis `x = 23`. Pour cela vous devez remplir à chaque fois un tableau de la forme :

a	b	T[a]+T[b]
⋮	⋮	⋮

, puis expliciter la valeur de retour.

b. On va montrer que l'algorithme `quoi` résout le problème de la question 1.

. Montrer que l'algorithme `quoi` termine.

On donne la propriété suivante :

$$(\mathcal{P}_{a,b}) \quad \begin{cases} \text{T est trié} \\ T[c] + T[d] \neq x \quad \forall c \in [0, a[\cup]b, n-1] \text{ et } 0 \leqslant d \leqslant n-1 \end{cases}$$

. Montrer que $(\mathcal{P}_{0,n-1})$ est vraie en entrant dans la boucle while.

. Montrer que si on suppose que $(\mathcal{P}_{a,b})$ est vraie au début d'un tour de boucle, elle reste vraie après une exécution de la boucle.

. Conclure.

c. Compter le nombre de comparaisons effectuées par la boucle `while` dans le cas pire pour deux tableaux de tailles respectives 3 et 12.

d. Quel tri faut-il choisir pour trier le tableau afin que la complexité en nombre de comparaisons soit la meilleure possible ? Rappeler sa complexité.

e. Avec le choix de tri fait à la question précédente, l'algorithme `quoi` est-il plus efficace que votre algorithme donné à la question 1 ? Justifier.