L2 informatique - Année 2021–2022

G5, Contrôle Continu d'Éléments d'Algorithmique n° 2

Nom, Prénom : ____________________

étudiant : ____________________

Exercice 1. *Récursion (5 points).*
Un nombre entier $n > 0$ est un nombre *Hamming* si ses seuls diviseurs premiers sont les 2,3 et 5 (s'il peut s'écrire comme $n = 2^i \cdot 3^j \cdot 5^k$, $i, j, k \geq 0$ entiers). Par exemple, $60 = 2^2 \cdot 3 \cdot 5$ est un nombre Hamming ; par contre $14 = 2 \cdot 7$ n'en est pas.
Écrire une fonction récursive `isHamming(n)` qui prend en paramètre un nombre n (supposé entier strictement positif) et qui retourne `true` si n est un nombre Hamming et `false` si non.
Argumenter si votre fonction est récursive terminale.
Réponse :

fonction isHamming :
Entrée n entier supposé strictement positif
si (n = 1) retourne true
si (n % 2 = 0) retourne isHamming (n/2)
si (n % 3 = 0) retourne isHamming (n/3)
si (n % 5 = 0) retourne isHamming (n/5)
retourne false

La fonction est récursive terminale car chaque instruction d'un retourne ne contient qu'un simple appel récursif.

Exercice 2. *Listes chaînées (5 points).*
Écrire une fonction `apresX(L,x)` qui, étant donné liste L (non vide) et élément x de la liste (un pointeur vers une Cellule qui n'est ni null ni la tête de la liste), elle deplace le premier élément de L après l'élément x. Par exemple, si la liste contient les données `L={1,4,3,5,7}` où la donnée 1 est à la tête de L et `x.data = 5`, alors `apresX(L,x)` modifie L de façon que `L={4,3,5,1,7}` (la donnée 4 est à la tête). Votre fonction ne doit créer aucune nouvelle Cellule.
Réponse :

Rappel : Les structures de données

Cellule :
Objet data
Cellule next

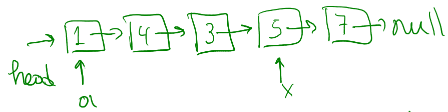
Liste :
Cellule head

---

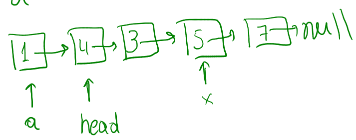
fonction apresX :

Entrée • Liste L supposée non-vide
• Cellule x supposée non-null et pas la tête de la liste L

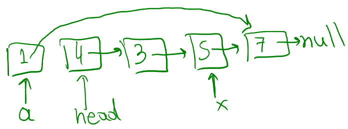
Cellule a = L.head



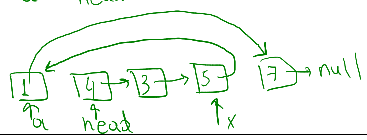
L.head = L.head.next



a.next = x.next



x.next = a



Exercice 3. *Tri par insertion dichotomique (10 points).*
Dans cet exercice, vous allez écrire une nouvelle version du tri par insertion (le code est dans l'Annexe ci-dessous). Vous allez remplacer la fonction `triPartiel(T, i)` avec une foction `triPartielDich(T, i)`. Cette fonction a le même effet que la fonction `triPartiel(T, i)` : étant donné un tableau T (ayant des éléments pas necessairement distincts) et un indice i, $1 \leq i \leq T.length - 1$, en supposant que le sous-tableau $T[0 \ldots (i-1)]$ est déjà trié, elle insère l'élément $T[i]$ dans le sous-tableau $T[0 \ldots (i-1)]$ à sa bonne place pour que $T[0 \ldots i]$ soit trié. Or, cette fois-ci, la recherche pour la bonne place de $T[i]$ se fait par une *recherche dichotomique*.

1. Écrire la fonction `triPartielDich(T, i)`.
2. Combien de comparaisons fait votre fonction `triPartielDich(T, i)` en fonction de i au meilleur et au pire de cas ? Et combien d'affectations au meilleur et au pire de cas ?

Réponse :

fonction triPartielDich :

Entrée • tableau T où T[0...i-1] est trié

• entier i supposé $1 \leq i < T.length$

$l \leftarrow 0$; $r \leftarrow i-1$

tant que $(l<r)$ faire

mid $\leftarrow (l+r)/2$

si (T[mid]=T[i]) placer (T, i, mid) // si T[i]==T[mid], on place T[i] à gauche de T[mid]

sinon

si (T[mid]<T[i] $l \leftarrow$ mid+1

sinon $r \leftarrow$ mid

si (T[l]<T[i]) placer (T, i, l+1) // quand on sort de la boucle : 1) $0 \leq l = r < i$; 2) soit T[i]=T[l]=T[r], soit T[i] ∉ T[0...i-1]. La bonne position pour emplacer T[i] est soit à gauche, soit à droite de T[l] (=T[r]).

sinon placer (T, i, l)

fonction placer :

Entrée • tableau T où T[0...i-1] est trié

• entier i supposé $1 \leq i < T.length$

• entier j supposé $0 \leq j \leq i$

si (j<i) // si j==i, T[0,...,i] est trié.

tmp $\leftarrow$ T[i]

$k \leftarrow i$

tant que $(k>j)$

T[k] $\leftarrow$ T[k-1]

T[j] $\leftarrow$ tmp

// la fonction place l'élément de de l'indice i dans la position j et déplace tous les éléments entre j et i-1 d'une position à droite :

T[.... ○ → ○ → ... → ○ → ○ ....]
j   j+1 ....   i-1   i

	Meilleur de cas	Pire de cas
Comparaisons :	1 (si T[i]==T[mid])	O(log i) (si T[i] ∉ T[0...i-1])
Affectations :	0 (si T[0...i] trié)	O(i) (si T[i] est minimum)

Annexe

Bien que le tri par insertion dichotomique ne fait que $O(n \log n)$ comparaisons, sa complexité est toujours $O(n^2)$ car il effectue $O(n^2)$ affectations.

Le tri par insertion :

Entrée : tableau T

fonction TRIPARTIEL(T, i)
    $j \leftarrow i$
    tant que $j > 0$ et $T[j] < T[j-1]$ **faire**
        échanger $T[j-1]$ et $T[j]$
        $j \leftarrow j-1$
fonction TRIPARINSERTION(T)
    pour $i \leftarrow 1$ à longueur de $T - 1$ **faire**
        TRIPARTIEL(T,i)