MPSIs
2012/2013

Devoir Surveillé 4 d'Informatique - 16/03/2013

durée : 4 heures

Ce sujet est très largement inspiré du sujet du concours E3A 2006

Dans ce sujet, la complexité d'une fonction désigne l'ordre de grandeur du nombre d'opérations réalisées par la fonction.

1 Le tri par sélection

Le tri (croissant) par sélection d'un tableau t de n éléments consiste à rechercher le plus grand élément de t, le permuter avec l'élément situé en fin de tableau, et à itérer le traitement avec un élément de moins, jusqu'à ce qu'il n'y ait plus qu'un seul élément.

Question 1	Écrire la fonction `indiceMaxi : int vect -> int` telle que `indiceMaxi t` recherche dans le tableau t le plus grand élément et retourne son indice. S'il y a plusieurs maxima égaux, elle retourne l'indice du premier de ceux-ci.
Question 2	Écrire la fonction ITÉRATIVE `triSelecIter : int vect -> unit` qui trie par sélection le tableau passé en entrée.
Question 3	Écrire la fonction RÉCURSIVE `triSelecRec : int vect -> unit` qui trie par sélection le tableau passé en entrée.
Question 4	Donner les complexités des fonctions triSelecIter et triSelecRec en fonction de la longueur du tableau passé en entrée.

2 Tri par insertion

Le tri (croissant) par insertion d'un tableau t de n éléments consiste à rechercher itérativement la place d'insertion de l'élément d'indice i dans les éléments d'indice 0 à i sachant que les éléments d'indice 0 à $i-1$ sont triés. Une fois cette place déterminée, on procède par échanges successifs pour que le tableau des éléments d'indice 0 à i soient triés.

On commence en considérant que l'élément d'indice 0 est, à lui tout seul, un tableau trié, et on procède ainsi pour insérer les éléments d'indice 2 à n.

La recherche de la place d'insertion peut s'effectuer

- soit *séquentiellement* (vu en cours) : les éléments d'indice 0 à $i-1$ sont considérés séquentiellement
- soit *par dichotomie* : pour insérer l'élément d'indice i à sa place, on teste d'abord si il est inférieur ou égal à l'élément d'indice $\lfloor i/2 \rfloor$, et on continue ainsi récursivement.

Question 5	Expliquer brièvement les deux principes de recherche (séquentielle et dichotomique) et donner leurs complexités (en pire cas).

Question 6	Ecrire la fonction `posInser1 : int vect -> int -> int` où `posInser1 t x` recherche séquentiellement dans un tableau `t` trié la place d'insertion de `x`.
Question 7	Ecrire la fonction `posInser2 : int vect -> int -> int` où `posInser1 t x` recherche par dichotomie dans un tableau `t` trié la place d'insertion de `x`.
Question 8	Ecrire la fonction `decale : int vect -> int (i) -> int (j) -> unit` qui décale d'une position vers la droite (incrémente d'un la position) tout les éléments du tableau en entrée entre les deux indices `i` et $j-1$ en entrée et met l'élément d'indice `j` en position `i`.
Question 9	Écrire la fonction ITÉRATIVE `triInsertion : int vect -> unit` qui trie par insertion le tableau passé en entrée.

3 Tri rapide (quick sort)

Le principe du quick sort est de partitionner le tableau à trier (s'il a au moins deux éléments) en deux sous-tableaux, le premier comprenant tous les éléments inférieurs ou égaux à un élément (l'élément pivot), le deuxième ne contenant qu'un seul élément (l'élément pivot) et le troisième contenant tous les éléments supérieurs ou égaux à l'élément pivot. Puis on réapplique **récursivement** le quick sort sur les premier et troisième sous-tableau (l'élément pivot, lui, est à sa place définitive).

Soient les fonctions

`partition : int vect -> int -> int -> int`

où `partition t i j` opère sur le sous-tableau de `t` des éléments d'indice allant de `i` à `j` (inclus), prend `t.(i)` comme élément pivot, déplace les éléments supérieurs ou égaux au pivot en fin de sous-tableau, positionne le pivot à sa place définitive et retourne l'indice de cette place

et

`quicksort : int vect -> int -> int -> unit`

qui trie le tableau `t` entre les indices `i` et `j` (inclus).

On considère de plus le tableau suivant :

indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13
t	5	1	12	3	24	8	10	2	14	7	2	9	4	17

Question 10	Donner un contenu possible de `t` après l'appel à `partition t 0 13` et sa valeur de retour.
Question 11	Avec quels paramètres doit-on appeler `quicksort` pour continuer le tri après cette partition ?
Question 12	Écrire la fonction `quicksort` .
Question 13	Écrire la fonction `partition` .

On pourra utiliser deux parcours, l'un allant du début du sous-tableau vers la fin et l'autre allant de la fin vers le début. Le parcours *montant* sera suspendu quand un élément sera supérieur au pivot ; le parcours *descendant* sera

suspendu quand un élément sera inférieur au pivot. Les éléments seront alors échangés et les parcours reprendront jusqu'à ce qu'ils se rejoignent. La place du pivot sera alors déterminée, le pivot y sera mis, et sa place retournée.

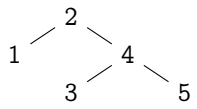
Question 14	Donner et expliquer la complexité de ce tri.

4 Le tri par tas (heap sort)

4.1 Définitions préliminaires

Déf.	Un *arbre binaire* est – soit un arbre vide appelé nil et noté ici $\emptyset$ – soit un nœud (père) (u, k, v) où u et v sont des arbres (appelés ses fils gauche et droit) et $k \in \mathbb{N}$ est sa valeur. Un nœud de la forme $(\emptyset, k, \emptyset)$ est appelé une feuille, on la note dans la suite k.

Par exemple $(1, 2, (3, 4, 5))$ est un arbre que l'on pourra représenter ainsi :



Déf.	Soit (g, r, d) un arbre binaire. g est le sous arbre gauche, d est le sous arbre droit et r est la *racine* de l'arbre, c'est-à-dire, le nœud situé en haut sur le dessin.

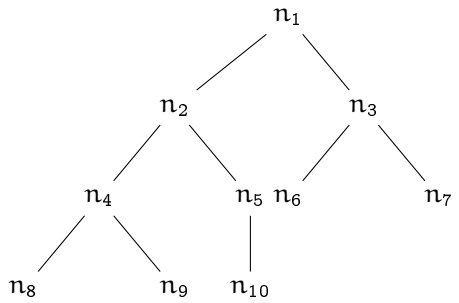
Dans l'exemple précédent, g est le sous-arbre réduit au nœud 1, la racine est le nœud 2 et $(3, 4, 5)$ est le sous arbre droit. Les feuilles sont les nœuds 1, 3 et 5.

Déf.	Si n est un nœud, le nombre de nœuds (n exclu) pour aller de n à la racine est appelé sa profondeur. La racine est le seul nœud de niveau 0. L'ensemble des nœuds de même profondeur p dans un arbre est appelé le niveau de profondeur p de l'arbre. Le nombre de nœuds d'un arbre est appelé sa taille.

Dans l'arbre précédente la racine est le nœud de valeur 2, la feuille de valeur 1 est de profondeur 1 et les autres feuilles sont de profondeur 2. Le niveau de profondeur 1 est constitué des nœuds de valeur $1, 3$ et 5. Cet arbre est de taille 5.

Déf.	Un arbre binaire presque complet (ABPC) est un arbre binaire dans lequel toutes les feuilles sont au même niveau de profondeur et où toutes les feuilles sont à gauche dans la représentation.

Par exemple l'arbre suivant est un ABPC :



Il dispose de quatre niveaux de profondeurs :
- Le niveau 0 qui contient n_1
- Le niveau 1 qui contient n_2 et n_3
- Le niveau 2 qui contient n_4, n_5, n_6 et n_7
- Le niveau 4 qui contient n_8, n_9 et n_{10}.

Question 15	On considère ici un ABPC dont les feuilles sont au niveau de profondeur n. Déterminer le nombre de nœuds au niveau de profondeur $k < n$. En déduire une majoration de la taille de l'arbre en considérant le cas où le niveau n serait entièrement rempli.

Il est possible de représenter un ABPC par un tableau en stockant successivement les nœuds du niveau 0, les nœuds du niveau 1, les nœuds du niveau n jusqu'à finir par les feuilles.

Dans le cas de l'ABPC précédent on obtient le tableau

```
let t = [| n1; n2; n3; n4; n5; n6; n7; n8; n9; n10 |]
```

Question 16	Dessiner l'arbre représenté par le tableau suivant : `let t = [| 9; 5; 8; 9; 7; 6; 7; 3; 6; 4 |]`

Déf.	Un nœud (u, k, v) est dit dominant si k est supérieure ou égale à la valeur de u et à la valeur de v (on considère que $\emptyset$ est de valeur $-\infty$). Un tas est un ABPC dont tous les nœuds sont dominants.

Question 17	Indiquer les nœuds dominants dans l'arbre obtenu à la question précédente.

4.2 Construction d'un tas

Question 18	Écrire les fonctions
	`indiceFilsG : int -> int`
	`indiceFilsD : int -> int`
	`indicePere : int -> int`
	telle que `indiceFilsG i` (resp. `indiceFilsD i`) renvoie l'indice du fils gauche (resp. droit) du nœud d'indice `i` dans un ABPC, et `indicePere i` renvoie l'indice du père du nœud d'indice `i`.

Question 19	Écrire les fonctions
	`estFeuille : int -> int -> bool`
	`estPere1 : int -> int -> bool`
	`estPere2 : int -> int -> bool`
	où `estFeuille n i` renvoie `vrai` si et seulement si `i` est l'indice d'une feuille d'un ABPC de taille n, et `estPere1 n i` (resp. `estPere2 n i`) renvoie `vrai` si et seulement si `i` est l'indice d'un nœud ayant un fils (resp. deux fils) dans un ABPC de taille n.

Question 20	Écrire la fonction `estDominant : int vect -> int -> bool` où `estDominant t i` retourne `vrai` si et seulement si `i` est l'indice d'un nœud dominant dans la représentation `t` sous forme de tableau d'un ABPC. Quelle est l'ordre de grandeur du nombre d'opérations par rapport à la taille de l'ABPC ?

Question 21	Écrire la fonction `retablirTas : int vect -> int -> unit` où `retablirTas t i` opère sur le sous-arbre de `t` (ABPC sous forme de tableau) à partir de l'indice `i` pour en faire un tas, **avec comme hypothèse que les fils de `i`, s'ils existent, sont des tas**. Par exemple, si `t = [| 9; 5; 8; 9; 7; 6; 7; 3; 6; 4 |]` l'appel à `retablirTas t 2` transformera l'arbre en `[|9; 9; 8; 6; 7; 6; 7; 3; 5; 4|]`. Décrire les opérations effectuées par la fonction. Quelle est l'ordre de grandeur du nombre d'opérations par rapport à la taille de l'ABPC ?
Question 22	Ecrire la fonction `construireTas : int vect -> unit` qui transforme l'ABPC donné en entrée pour en faire un tas. Par exemple si `t = [| 1; 2; 3; 4; 5; 6; 7; 8; 9; 10 |]` l'appel à `construireTas t` transformera l'arbre en `[| 10; 9; 7; 8; 5; 6; 3; 1; 4; 2 |]`. Quelle est l'ordre de grandeur du nombre d'opérations par rapport à la taille de l'ABPC ?

4.3 Tri par tas

Le tri par tas consiste à interpréter le tableau comme un arbre binaire presque complet, à le transformer en tas, puis itérativement, à permuter la racine de l'arbre avec la dernière feuille, puis, à reconstituer le tas avec un élément de moins et ce jusqu'à ce qu'il n'y ait plus qu'un élément à traiter.

Question 23	Écrire la fonction `heapsort : int vect -> unit` qui réalise cet algorithme et trie le tableau donné en entrée. Quelle est l'ordre de grandeur du nombre d'opérations par rapport à la taille du tableau ?