IFT2015 H19 :: Examen intra[1]

[1] The English translation follows.

L'EXAMEN vaut 100 points, et vous pouvez avoir jusqu'à 10 points de boni additionnels.

- ⋆ Aucune documentation n'est permise.
- ⋆ Décrivez vos algorithmes en pseudocode ou en Java(-esque).
- ⋆ Répondez à toutes les questions dans les cahiers d'examen.

Exo	points	boni
F0	1	
F1	$9+5=14$	
F2	15	
F3	20	
F4	25	
F5	$10+15=25$	10
Σ	100	10

F0 Votre nom (1 point)

▶ Mettez votre nom et code permanent sur tous les cahiers soumis.

F1 Types abstraits (14 points)

a. Définition (5 points) ▶ Donner la définition de *type abstrait* [de données].

b. Types fameux (9 points) ▶ Décrire les opérations principales pour les trois types abstraits suivants, sans discuter l'implantation :
(1) pile, (2) file FIFO (=queue), et (3) file de priorité.

F2 Tris (15 points)

▶ Pour chacun des algorithmes suivants, donner l'ordre de croissance du du temps de calcul en moyen et au pire cas, ainsi que de l'espace de travail[2] utilisé au pire : tri rapide, tri par tas, tri par fusion, tri par insertion, tri par sélection. Il n'est pas nécessaire de justifier vos réponses.

[2] «espace de travail» : mémoire utilisé sans compter l'entrée de n éléments

tri	temps		mémoire
	moyen	pire	
rapide			
tas			
fusion			
insertion			
sélection			

F3 File concatenable (20 points)

▶ Donner une implémentation de file FIFO avec concaténation, basée sur une liste circulaire (simplement) chaînée. Opérations à implémenter sur une file Q :

- ⋆ Q.enqueue(x) enfiler x
- ⋆ Q.dequeue() supprimer et retourner le plus ancien élément sur la file
- ⋆ Q.concat(P) concaténer la file P après Q : équivalent d'enfiler tous les éléments de P sur Q. La file P est détruite par cette opération.

Toutes les opérations doivent s'exécuter en $O(1)$.

50

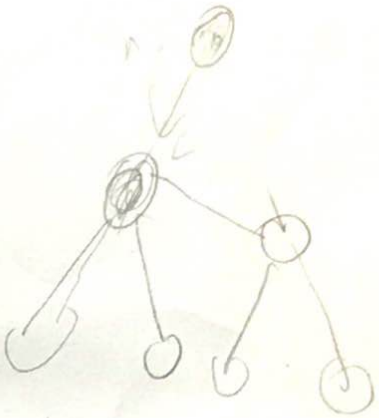
F4 File égalitaire (25 points)

Étant donné une collection d'éléments comparables $\mathcal{C} = \{x_1 x_2, \ldots, x_n\}$, un $x \in \mathcal{C}$ est un élément *médian* s'il permet de couper $\mathcal{C}$ en deux parties égales :

$$\underbrace{\left|\{i: x_i \leq x\}\right| \geq \left\lceil \frac{n}{2} \right\rceil}_{\text{au moins la moitié des éléments sont inférieurs ou égaux à } x} \quad \text{et, en même temps} \quad \underbrace{\left|\{i: x_i \geq x\}\right| \geq \left\lceil \frac{n}{2} \right\rceil}_{\text{au moins la moitié des éléments sont supérieurs ou égaux à } x}$$

▶ Concevoir une structure[3] de données qui implante les opérations insert(x) et deleteMedian(). L'opération insert insère un nouvel élément ; deleteMedian supprime et retourne un élément médian. Toutes les deux opérations doivent s'exécuter en $O(\log n)$.

[3] **Indice:** : utiliser un min-tas et un max-tas.



F5 Recherche trichotomique (25 points+10 points boni)

Professeure Boucles d'or propose une procédure récursive pour chercher un élément x dans un tableau trié $A[0..n-1]$. Une sous-tâche pour la récurrence est de chercher x aux indices $g \leq i < d$ (dans le «sous-tableau» $A[g..d-1]$) ; le premier appel est avec $g = 0$ et $d = n$. Si $d - g = 0$ alors retourner avec échec. Si $d - g = 1$, alors examiner si $x = A[g]$ et déclarer succès ou échec. Si $d - g > 1$, alors définir les pivots $y = A[p]$ et $z = A[q]$ aux indices

$$p = \left\lfloor \frac{2g+d}{3} \right\rfloor \quad \text{et} \quad q = \left\lfloor \frac{g+2d}{3} \right\rfloor.$$

Maintenant, si $x = y$ ou $x = z$, alors retourner avec succès. Si $x < y$, alors x est trop petit, et il faut continuer la recherche dans $A[g..p-1]$. Si $x > z$, alors x est trop grand, et il faut continuer la recherche dans $A[q..d-1]$. Autrement, si x n'est ni trop grand ni trop petit ($y < x < z$), il faut continuer la recherche dans $A[p..q-1]$.

a. Pseudocode (10 points) ▶ Décrire l'algorithme en pseudocode. (Retourner vrai/faux = si $A[\,]$ contient x.)

b. Analyse (15 points) ▶ Démontrer que le nombre d'accès[4] au tableau est $\sim 2\log_3 n$ au pire. Comparer avec la recherche dichotomique.

[4] «accès au tableau» : chaque fois qu'on accède à une case $A[j]$

c. Tri par tas ternaire (10 points boni) Prof. Boucles d'or maintient que le tri par tas ternaire[5] fait asymptotiquement moins de comparaisons au pire que le traditionnel tri par tas binaire. ▶ Confirmer ou infirmer sa proposition.

[5] Tas ternaire dans $A[0..n-1]$: enfants de $A[i]$ aux indices $3i+1, 3i+2, 3i+3$, parent à $\lfloor (i-1)/3 \rfloor$. On change la procédure de sink, mais autrement le tri procède de même façon :

```
for i ← n−2, n−1, ..., 0 do sink(A[i], i, A, n)
for i ← n−1, n−2, ..., 1 do
    échanger A[0] ↔ A[i]
    sink(A[0], 0, A, i)
```

BONNE CHANCE !

Mid-term examination

THE EXAMEN is worth 100 points, and you can collect up to 10 additional bonus points.

- ⋆ No documentation is allowed.
- ⋆ Write your algorithms in (Java-style) pseudocode.
- ⋆ You may write your answers in English or in French.
- ⋆ Answer each question in the exam booklet.

Exercise	points	bonus
E0	1	
E1	9 + 5 = 14	
E2	15	
E3	20	
E4	25	
E5	10 + 15 = 25	10
Σ	100	10

E0 Your name (1 point)

▶ Write your name and *code permanent* on each booklet that you submit.

E1 Abstract types (14 points)

(a) Definition (5 points) ▶ Define what is an *abstract data type*.

(b) Famous types (9 points) ▶ Describe the fundamental operations for the following three abstract data types, without discussing implementation :
(1) [LIFO] stack, (2) [FIFO] queue, (3) priority queue.

E2 Sorting (15 points)

▶ Give the growth order for the running time in the worst and average case, as well as the worst-case work space[6] requirements for the following sorting algorithms : quicksort, heapsort, mergesort, insertion sort, and selection sort. You do not need to justify your answers.

method	time		memory
	average	worst	
quicksort			
heapsort			
mergesort			
insertion sort			
selection sort			

[6] "work space" : memory usage excluding the n-element input array

E3 Concatenable queue (20 points)

▶ Give an implementation of a FIFO queue with concatenation, using a (single-)linked circular list. Operations to implement for a queue Q :

- ⋆ Q.enqueue(x) adds x to the queue
- ⋆ Q.dequeue() removes and returns the oldest element in the queue
- ⋆ Q.concat(P) concatenates the queue P after Q : equivalent to adding all elements of P to Q. (P is destroyed in this operation.)

All operations must take $O(1)$ time.

E4 Equality queue (25 points)

Given a collection of comparable elements $\mathcal{C} = \{x_1\, x_2, \ldots, x_n\}$, an $x \in \mathcal{C}$ is a *median* element if it cuts $\mathcal{C}$ in two near-equal parts :

$$\underbrace{\left|\{i\colon x_i \le x\}\right| \ge \left\lceil \frac{n}{2} \right\rceil}_{\text{at least half of the elements are less than or equal to } x} \quad \text{and, at the same time} \quad \underbrace{\left|\{i\colon x_i \ge x\}\right| \ge \left\lceil \frac{n}{2} \right\rceil}_{\text{at least half of the elements are greater than or equal to } x}.$$

▶ Give a data structure[7] that implements the operations insert(x) that inserts a new element x in the collection, and deleteMin() that removes and returns a median element. Both operations must take $O(\log n)$ time.

[7] **Hint:** : use a min-heap and a max-heap.

E5 Ternary search (25 points+10 bonus points)

Professor Goldilocks proposes a recursive procedure to search for an x in a sorted array $A[0..n-1]$. One subproblem for the recursion is searching for x at the indices $g \le i < d$ (in the «subarray» $A[g..d-1]$) ; the first call is with $g = 0$ and $d = n$. If $d - g = 0$ then return with failure. If $d - g = 1$, then examine if $x = A[g]$ and declare success or failure. If $d - g > 1$, then define the pivots $y = A[p]$ et $z = A[q]$ at the indices

$$p = \left\lfloor \frac{2g+d}{3} \right\rfloor \quad \text{and} \quad q = \left\lfloor \frac{g+2d}{3} \right\rfloor.$$

Now, if $x = y$ ou $x = z$, then return with success. If $x < y$, then x is too small, and the search continues in $A[g..p-1]$. If $x > z$, then x is too big, and the search continues in $A[q..d-1]$. Otherwise, if x is not too small and not too big ($y < x < z$), continue with $A[p..q-1]$.

a. Pseudocode (10 points) ▶ Describe the algorithm in pseudocode. (Return true/false — whether $A[\,]$ contains x.)

b. Analysis (15 points) ▶ Show that the number of array accesses[8] in the worst case is $\sim 2\log_3 n$. Compare to binary search.

[8] "array access" : every time a cell $A[j]$ is consulted

c. Ternary-heap sort (10 bonus points) Prof. Goldilocks maintains that heap-sort with a ternary[9] heap makes asymptotically fewer comparisons in the worst case than the traditional binary-heap sort. ▶ Prove or disprove Goldilocks' assertion.

[9] Ternary heap in $A[0..n-1]$: children of $A[i]$ with indices $3i+1, 3i+2, 3i+3$; parent at $\lfloor (i-1)/3 \rfloor$. One needs to change sink, but otherwise the sort proceeds the same way :

```
for i ← n−2, n−1, ..., 0 do sink(A[i], i, A, n)
for i ← n−1, n−2, ..., 1 do
    échanger A[0] ↔ A[i]
    sink(A[0], 0, A, i)
```

GOOD LUCK !