

CSI2110/CSI2510 Automne 2018, Prof. L. Moura et Prof. R. Laganière
Examen de mi-session
4 Novembre, 10h00 - 120 minutes
30 points (30%)
Aucune documentation permise – Pas de calculatrice

SOLUTIONS

Nom: _____

Numéro d'étudiante: _____

Signature _____

9 pages

Pour la complexité Grand O, toujours spécifier la meilleure valeur.

Tous les logarithmes sont en base 2.

QUESTION	POINTAGE
Questions 1-7	/7
Question 8	/3
Question 9	/3
Question 10	/2
Question 11	/2
Question 12	/5
Question 13	/3
Question 14	/5
TOTAL	/30

Question 1 [1 point] Le programme suivant utilise une pile réalisée avec une liste chaînée simple. Donner la complexité Grand O de cet algorithme en fonction de N.

```
Stack<Integer> myStack = new Stack<Integer>();  
  
for (int i=0; i< N; i++) {  
    myStack.push(i);  
}  
int tot=0;  
for (int i=0; i< N; i++) {  
    tot += myStack.pop();  
}
```

- A) $O(1)$ B) $O(\log N)$ C) $O(N)$ D) $O(N \log N)$ E) $O(N^2)$

Question 2 [1 point] Le programme suivant utilise un tableau trié `arr` contenant N éléments et utilise une recherche binaire retournant l'index de l'élément du tableau contenant la clé cherchée (`key`) ou `-1` si la clé est introuvable. Donner la complexité Grand O de cet algorithme en fonction de N.

```
int j; int count=0;  
int N = arr.length;  
  
for (int i=0; i< N/2; i++) {  
    int key=2*i;  
    j=binarySearch(arr, key, 0, N-1);  
    if (j != -1) count++;  
}
```

- A) $O(1)$ B) $O(\log N)$ C) $O(N)$ D) $O(N \log N)$ E) $O(N^2)$

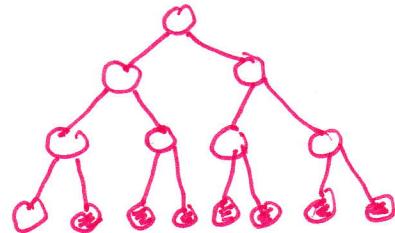
Question 3 [1 point] Donner la complexité Grand O de l'algorithme ci-dessous en fonction de N.

```
int i=1;  
while (i< N) {  
    i = i*2;  
    print(i);  
}
```

- A) $O(1)$ B) $O(\log N)$ C) $O(N)$ D) $O(N \log N)$ E) $O(N^2)$

Question 4 [1 point] Quel est le nombre minimum et maximum de noeuds dans un arbre binaire complete de hauteur 3 ?

- A) min= 4, max= 8
- B) min= 4, max= 15
- C)** min= 8, max= 15
- D) min= 8, max= 16
- E) Aucune de ces réponses.



Question 5 [1 point]

Soit un arbre binaire contenant des lettres et tel que son parcours **in-ordre** donne G,C,A,D,B,F,E et que son parcours **pré-ordre** donne B, G, A, C, D, F, E. Que donnera alors son parcours **post-ordre**?

- A) G, A, C, B, E, F, D
- B) D, C, F, G, A, B, E
- C) E, F, D, C, A, G, B
- D)** C, D, A, G, E, F, B
- E) Aucune de ces réponses.

Question 6 [1 point]

Soit une file à priorité composée de n éléments, réalisée avec un **monceau-min (min-heap)**. Laquelle des réponses ci-dessous donne la complexité respective des trois opérations suivantes:
`insert` (insertion d'un élément quelconque dans la file)
`removeMin` (retrait de l'élément de valeur minimum de la file)
`min` (obtenir la valeur de l'élément minimum dans la file, sans modifications de celle-ci):

- A) $\Theta(1)$, $\Theta(l)$, $\Theta(1)$
- B)** $\Theta(\log n)$, $\Theta(\log n)$, $\Theta(1)$
- C) $\Theta(\log n)$, $\Theta(\log n)$, $\Theta(\log n)$
- D) $\Theta(n)$, $\Theta(n)$, $\Theta(n)$
- E) Aucune de ces réponses.

Question 7 [1 point]

Dans laquelle des structures de données suivantes contenant n éléments le retrait de l'élément de valeur minimum a une complexité au **pire cas** de $\Theta(n)$?

- A) Un tableau trié en ordre croissant
- B) Une liste doubllement chainée non triée
- C) Un arbre binaire de recherche
- D) Une file
- E)** Aucune de ces réponses.

{ pire cas $\Theta(n)$

Question 8 [3 points] Soit un TAD dictionnaire pouvant être réalisé avec différentes structures de données.

Pour chacune des opérations ci-dessous, donner la complexité Grand O au pire cas en fonction de n , le nombre d'éléments dans la structure de données (en supposant que dans chaque cas, l'algorithme le plus efficace est utilisé).

- `search(k)` : retourne l'élément ayant la clé spécifiée (null si il n'existe pas).
- `insert(k, v)` : insert l'élément v ayant la clé k dans le dictionnaire.
- `searchAndRemove(k)` : retire l'élément dont la clé est k ; cet élément doit être trouvé d'abord.

	<code>search(k)</code>	<code>insert(k, v)</code>	<code>searchAndRemove(k)</code>
unsorted linked list	$\mathcal{O}(n)$	$\mathcal{O}(1)$ à la fin	$\mathcal{O}(n)$
sorted array	$\mathcal{O}(\log n)$ binnaire	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Binary Search Tree	$\mathcal{O}(n)$ arbre linéaire	$\mathcal{O}(n)$	$\mathcal{O}(n)$

Question 9 [3 points] Quelle est la complexité Grand O (big-Oh) des fonctions ci-dessous?
Justifier vos réponses par de courtes démonstrations.

a) $f(n) = 1 + n + 3n^2 + 3n^3 \leq n^3 + n^3 + 3n^3 + 3n^3 = 8n^3$

$\mathcal{O}(n^3)$

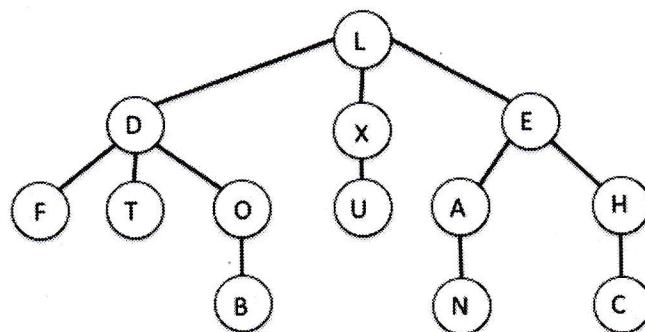
b) $f(n) = \log_2(2^n \cdot n^3) \quad \log 2^n + \log n^3 = n + 3 \log n$

$\mathcal{O}(n)$

c) $f(n) = \sum_{i=0}^n 2^i = 1 + 2 + 4 + \dots + 2^n = 2^{n+1} - 1$

$\mathcal{O}(2^n)$

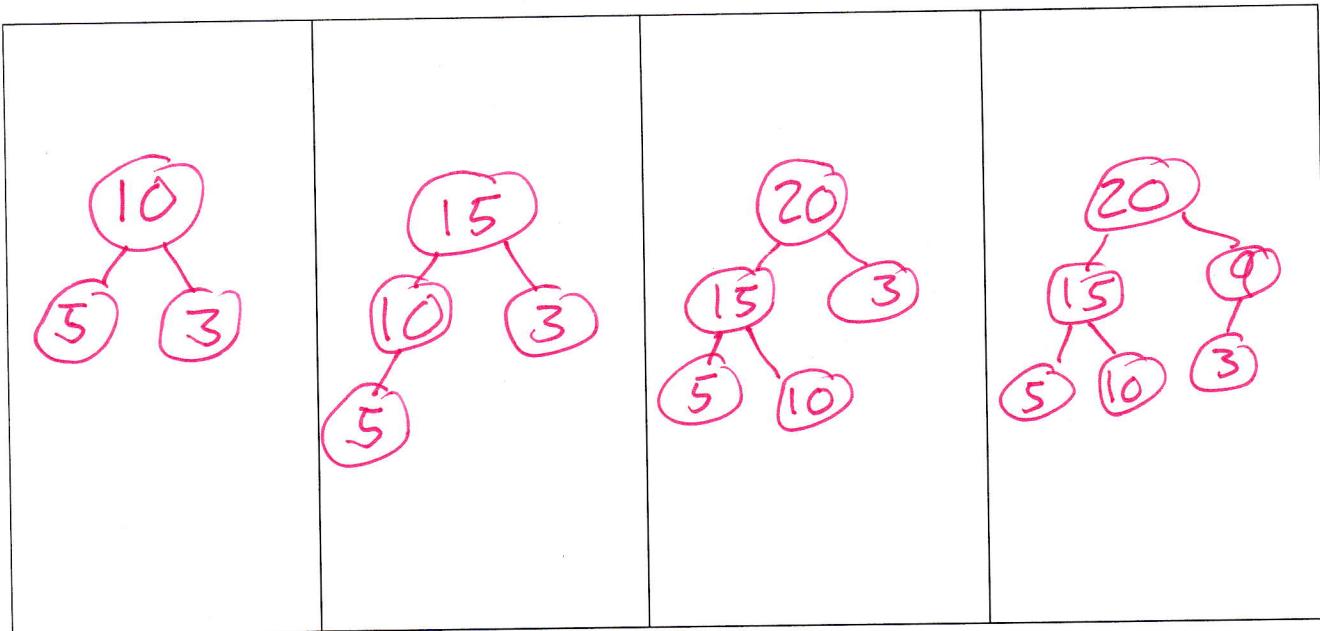
Question 10 [2 points] Donner le parcours des noeuds de cet arbre pour les stratégies suivantes :



pré-ordre : LDFTOBXUEANHC
 post-ordre: FTBODUXNACHEL

Question 11 [2 points]

- (a) Dessiner le **monceau-max (max-heap)** qui sera obtenu après l'insertion, dans l'ordre, des éléments ayant les clés suivantes (partant d'un monceau vide): 5, 10, **3**, **15**, **20**, **9** (dessiner le monceau après l'insertion de chacun des éléments en caractères gras)



Question 12 [4 points] Soit le tableau suivant:

0	1	2	3	4	5	6
10	40	50	20	70	30	60

Trier ce tableau sur-place en ordre croissant en utilisant l'algorithme du **tri par monceau (HeapSort)**.

Phase 1: Construction ascendante: montrer l'état du tableau après chaque opération de *downheap*.

Phase 2: Retrait des éléments du monceau afin d'obtenir le tableau trié. Montrer l'état du tableau chaque fois qu'un élément est retiré du monceau.

Bien indiquer le moment où vous passer de la phase 1 à la phase 2.

Répondre sur la page suivante →

Phase 1

0	1	2	3	4	5	6
10	40	<u>60</u>	20	70	30	<u>50</u>

10	<u>70</u>	60	20	<u>40</u>	30	50
----	-----------	----	----	-----------	----	----

<u>70</u>	<u>40</u>	60	20	10	30	50
-----------	-----------	----	----	----	----	----

Phase 2

60	40	50	20	10	30	70
----	----	----	----	----	----	----

50	40	30	20	10	60	70
----	----	----	----	----	----	----

40	20	30	10	50	60	70
----	----	----	----	----	----	----

30	20	10	40	50	60	70
----	----	----	----	----	----	----

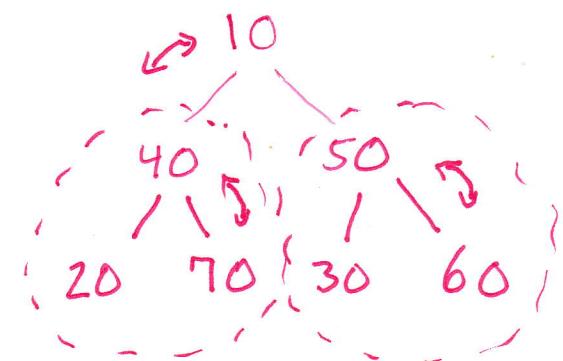
20	10	30	40	50	60	70
----	----	----	----	----	----	----

10	20	30	40	50	60	70
----	----	----	----	----	----	----

--	--	--	--	--	--	--

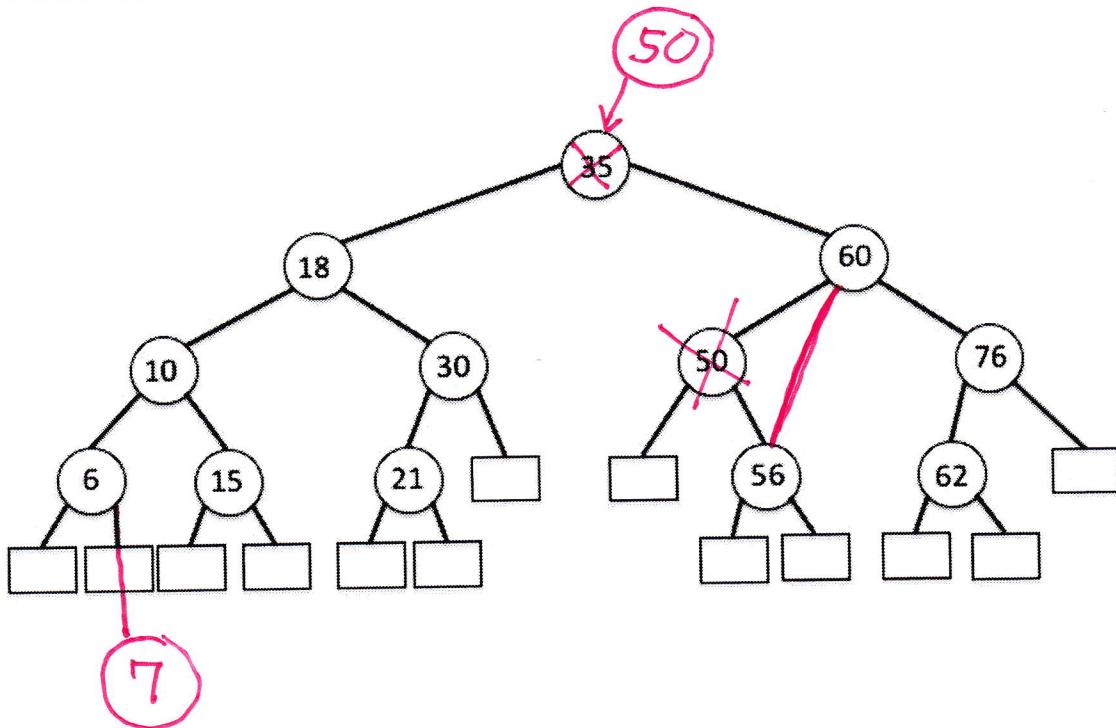
--	--	--	--	--	--	--

--	--	--	--	--	--	--



Question 13 (3 points =1+2)

Soit l'arbre binaire de recherche ci-dessous :



- Modifier l'arbre ci-dessus afin de montrer son état après l'insertion de l'élément 7. Dessiner directement sur l'arbre montré afin de clairement montrer la structure du nouvel arbre.
- Modifier l'arbre ci-dessus afin de montrer son état après le retrait de l'élément 35. Votre solution ne doit pas modifier la structure du sous-arbre de gauche (celui dont la racine est 18). Dessiner directement sur l'arbre montré afin de clairement montrer la structure du nouvel arbre.

Question 14. (5 points) Soit un arbre binaire de recherche avec les opérations suivantes:

root()	Retourne le noeud à la racine de l'arbre ou null si l'arbre est vide
parent(p)	Retourne le noeud parent du noeud p ou null si p est la racine de l'arbre
leftChild(p)	Retourne la position de l'enfant de gauche de p ou null si ce noeud n'a pas d'enfant de gauche.
rightChild(p)	Retourne la position de l'enfant de droite de p ou null si ce noeud n'a pas d'enfant de droite.
key(p)	Retourne la clé du noeud p
isInternal(p)	Retourne vrai si le noeud p est interne
isExternal(p)	Retourne vrai si le noeud p est externe (une feuille)
size()	Retourne le nombre de noeuds dans l'arbre

En utilisant les opérations ci-dessus, on vous demande une description par pseudo-code (vous pouvez faire votre description en Java).

- a) (3.5 points) Donner l'algorithme permettant d'obtenir la valeur de la clé maximum contenue dans l'arbre. Par exemple, cet algorithme retournerait 76 pour l'arbre de la page précédente.

int maxKey() { // complete the code here.

$p = \text{root}();$
 $\text{if } (\text{right}(p) == \text{null})$
 $\quad \text{return } \text{EMPTY_TREE}$
 $\text{while } (\text{right}(p) != \text{null})$
 $\quad p = \text{right}(p);$
 $\text{return } \text{key}(p);$

3

- b) (1.5 point) Si cet arbre contient n noeuds et a toujours une hauteur de $h = 2 \log n$, quell sera alors la complexité Grand O de la fonction maxKey() ? O(log n)