Mark: 14.5/20 (total score: 18/25)

+191/1/20+

Examen d'algorithmique

Janvier 2017, ING1. Durée : 1 heure 30

Consignes :

- Aucun document ni appareil électronique autorisé.
- Dans les question à choix multiples, noircissez les cases au stylo (pas de crayon à papier) et sans déborder sur les voisines car la correction est automatisée.
- **En cas d'erreur**, recouvrir la case entièrement de blanc correcteur (inutile de chercher à préserver son contour).
- Le barème, indicatif, correspond à une note sur 29.
- Certaines réponses incorrectes apportent des points négatifs. Dans le doute, s'abstenir.
- Lorsqu'une réponse numérique demande plusieurs chiffres, les chiffres sont lus de haut en bas.
- Vous ne devez pas remplir le cadre sur fond gris de la dernière page.

Prénom, NOM

UID : ☐0 ☐1 ☒2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☒0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☒2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☒5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☒7 ☐8 ☐9

1 Divers (5 pts)

Question 1 On considère le tas « max » correspondant au tableau suivant :

9	8	5	7	6	4	3	1	2

.

Donnez l'état du tas après les *suppressions* successives de ses trois plus grandes valeurs. (Indiquez, de haut en bas, les valeurs du tas dans l'ordre où elle apparaissent dans le tableau résultant.)

0/2

Question 2 L'algorithme de Karatsuba permet...

- ● de trier un tableau de n valeurs en $\Theta(n \log n)$ opérations.
- ● de trier un tableau de n valeurs en $\Theta(n)$ opérations.
- ● de multiplier deux polynômes de degré n en $\Theta(n^{\log_2(3)})$ opérations.
- ● de multiplier deux matrices $n \times n$ en $\Theta(n^{\log_2(7)})$ opérations.
- ● de trouver le parenthésage optimal d'une chaîne de multiplication de n matrices en $\Theta(n^2)$ opérations.

1/1

Question 3 Avec les deux lettres $\{a, b\}$, on peut construire quatre mots de deux lettres : $\{aa, ab, ba, bb\}$. Avec un alphabet de k lettres, combien de mots de n lettres peut-on construire ?

- ● k^n
- ● $k + n$
- ● $\binom{n}{k}$
- ● $\binom{k}{n}$
- ● kn
- ● n^k
- ● $\log_k n$
- ● $\log_n k$

2/2

2 Bouclettes (4 pts)

Question 4 Combien de fois le programme suivant affiche-t-il `"x"` ?

```
for (int i = 4; i < 24; ++i)
  for (int j = i + 2; j - 1 > 0; --j)
    puts("x");
```

[redacted]

2/2

Question 5 Combien de fois le programme suivant affiche-t-il `"x"` ?

```
for (int i = 4; i <= 22; i += 2)
  for (int j = i; j > 0; --j)
    puts("x");
```

[redacted]

2/2

3 Programmation dynamique (6 pts)

Un arbre binaire est dit *"de recherche"* s'il est étiqueté par des valeurs et que l'étiquette de chaque nœud est supérieure ou égale à toutes les étiquettes de son sous-arbre gauche, et inférieure ou égale à toutes les étiquettes de son sous-arbre droit. Cet ordre permet de chercher des valeurs en sachant toujours quel sous-arbre inspecter.

On considère un arbre binaire de recherche étiqueté par n entiers distincts : $m_1 < m_2 < \cdots < m_n$. Chaque entier m_i est associé à un poids w_i représentant la fréquence avec laquelle il sera recherché dans l'arbre (plus w_i est grand, plus m_i sera recherché souvent). Notre objectif est de construire l'arbre de recherche le plus efficace en fonction de ces poids, connus à l'avance.

Pour formaliser la notion d'arbre efficace, définissons le coût d'accès pondéré C d'un arbre binaire de recherche pour n entiers comme

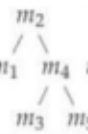
$$C = \sum_{i=1}^{n} (d_i + 1) w_i$$

où d_i représente la profondeur du nœud représentant la valeur m_i dans l'arbre (la racine étant à la profondeur 0). Intuitivement $d_i + 1$ correspond au nombre de comparaisons à faire avant de trouver m_i dans l'arbre.

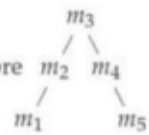
Par exemple considérons deux arbres binaires de recherches pour les valeurs

i	1	2	3	4	5
m_i	2	5	6	8	9
w_i	8	4	14	3	6

.



Par exemple l'arbre a un coût d'accès pondéré de $C = 2w_1 + 1w_2 + 3w_3 + 2w_4 + 3w_5 = 86$.



Tandis que le coût d'accès pondéré de l'arbre est $C = 3w_1 + 2w_2 + 1w_3 + 2w_4 + 3w_5 = 70$.

Si l'on devait choisir entre ces deux arbres, on garderait le second, de coût inférieur. On veut évidement trouver l'arbre de coût minimal parmi **tous** les arbres binaires de recherche possibles pour les valeurs $m_1, \ldots, m_n$.

Notons $C(i,j)$ le coût d'accès pondéré **minimal** des arbres binaires de recherche pour les valeurs $m_i, \ldots, m_j$. On a

$$C(i,j) = \begin{cases} 0 & \text{si } i > j \\ w_i & \text{si } i = j \\ \min \left\{ C(i,k-1) + C(k+1,j) + \sum_{m=i}^{j} w_m \,\middle|\, k \in [\![i,j]\!] \right\} & \text{si } i < j \end{cases}$$

Pour notre exemple, cette formule nous permet de construire le tableau $C(i,j)$:

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	8	16	42	48	y
$i=2$	0	4	22	28	x
$i=3$	0	0	14	20	35
$i=4$	0	0	0	3	12
$i=5$	0	0	0	0	6

Question 6 Quelles sont les valeurs de x et y dans le tableau précédent ?

- [x] $x = 43, y = 62$
- [ ] $x = 43, y = 63$
- [x] $x = 42, y = 63$
- [ ] $x = 42, y = 62$

0/2

Question 7 Pour n valeurs, quelle est la complexité de l'algorithme qui remplit ce tableau ?

- [x] $\Theta(n^2 \log n)$
- [ ] $O(n^2)$
- [ ] $O(n^2 \log n)$
- [x] $\Theta(n^2)$
- [ ] $O(n^3)$

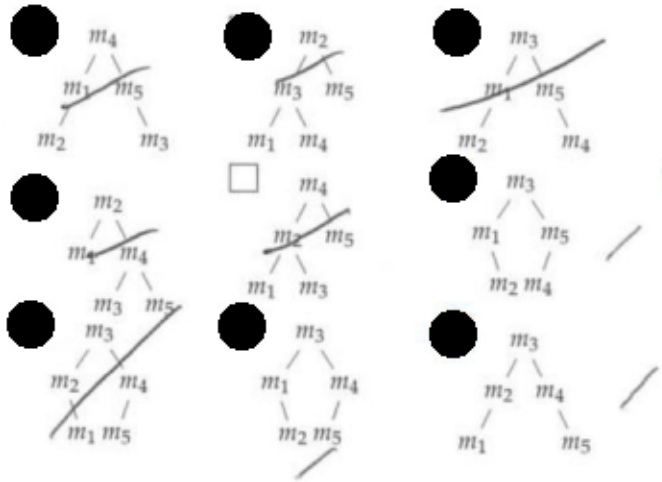
2/2

La personne qui a écrit l'algorithme avait suivi les cours d'algo, donc elle a aussi pris la peine de sauvegarder dans un tableau séparé la valeur du k qui donnait le coût minimal dans le calcul de $C(i,j)$.

Ce tableau des k est le suivant :

	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	1	3	3	3
$i=2$		3	3	3
$i=3$			3	3
$i=4$				5

Question 8 Quel est l'arbre binaire de recherche de coût minimal correspondant ?



0/2

4 Complexité récursive (5 pts)

Rappel du théorème général. Pour une récurrence du type $T(n) = aT(n/b + O(1)) + f(n)$ avec $a \geq 1$, $b > 1$:

- si $f(n) = O(n^{(\log_b a)-\varepsilon})$ pour un $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$;
- si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$;
- si $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ pour un $\varepsilon > 0$, **et** de plus $af(n/b) \leq cf(n)$ pour un $c < 1$ et toutes les grandes valeurs de n, alors $T(n) = \Theta(f(n))$.

Question 9 Pour une entrée de taille n, un algorithme a une complexité $T(n)$ qui vérifie

$$T(n) = 5T(n/4) + O(n).$$

Quelle est sa classe de complexité (la plus précise) ?

$T(n) = \Theta(n^2)$	$T(n) = O(n^2)$
$T(n) = \Theta(n^{\log_4 5})$	$T(n) = O(n^{\log_4 5})$
$T(n) = \Theta(n^{\log_5 4})$	$T(n) = O(n^{\log_5 4})$
$T(n) = \Theta(n^{\log_4 5} \log n)$	$T(n) = O(n^{\log_4 5} \log n)$
$T(n) = \Theta(n^{\log_5 4} \log n)$	$T(n) = O(n^{\log_5 4} \log n)$
$T(n) = \Theta(n \log n)$	$T(n) = O(n \log n)$
$T(n) = \Theta(n)$	$T(n) = O(n)$

Question 10 Pour une entrée de taille n, un algorithme a une complexité $T(n)$ qui vérifie

$$T(n) = 2T(n/2) + n^2 + \log n.$$

Quelle est sa classe de complexité (la plus précise) ?

$T(n) = \Theta(n^2 \log n)$	$T(n) = O(n^2 \log n)$
$T(n) = \Theta(n^2)$	$T(n) = O(n^2)$
$T(n) = \Theta(n(\log n)^2)$	$T(n) = O(n(\log n)^2)$
$T(n) = \Theta(n \log n)$	$T(n) = O(n \log n)$
$T(n) = \Theta(n)$	$T(n) = O(n)$
$T(n) = \Theta(\log n)$	$T(n) = O(\log n)$

Question 11 Pour une entrée de taille n, un algorithme a une complexité $T(n)$ qui vérifie

$$T(n) = 2T(n/2) + n \log_2 n.$$

Quelle est sa classe de complexité (la plus précise) ?

$T(n) = \Theta(n^2 \log n)$	$T(n) = O(n^2 \log n)$
$T(n) = \Theta(n^2)$	$T(n) = O(n^2)$
$T(n) = \Theta(n(\log n)^2)$	$T(n) = O(n(\log n)^2)$
$T(n) = \Theta(n \log n)$	$T(n) = O(n \log n)$
$T(n) = \Theta(n)$	$T(n) = O(n)$
$T(n) = \Theta(\log n)$	$T(n) = O(\log n)$

5 Recherche récalcitrante (6 pts)

En 1984, A. Broder et J. Stolfi (Dec System Research Center) ont publié un article satirique intitulé *"Pessimal Algorithms and Simplexity Analysis"* où ils proposent de construire des algorithmes "pessimaux", c'est-à-dire qui résolvent un problème en prenant un *maximum* de temps. Évidemment, on peut toujours ralentir un algorithme en y ajoutant des boucles inutiles, mais ce n'est pas élégant : pendant les itérations superflues il est clair que l'algorithme ne progresse pas. Leur objectif est donc de construire des algorithmes qui progressent strictement vers la solution, mais qui y arrivent avec très peu d'enthousiasme, voire une aversion manifeste.

Voici leur procédure RELUCTANTSEARCH, pour la recherche de la position d'un entier x dans un tableau $A[i..j]$ qui est trié dans l'ordre croissant. L'algorithme retourne $k = -1$ si x n'a pas été trouvé, ou $k \in \{i, ..., j\}$ si $A[k] = x$.

Dans la suite on note $n = j - i + 1$ le nombre de cases dans lesquelles la recherche est effectuée.

```
RELUCTANTSEARCH(A, i, j, x)
 1  if i > j then return −1
 2  if i = j then
 3      if x = A[i] then return i else return −1
 4  m ← ⌊(i+j)/2⌋
 5  if x ≤ A[m] then
 6      k ←RELUCTANTSEARCH(A, m + 1, j, x)
 7      if k = −1 then
 8          return RELUCTANTSEARCH(A, i, m, x)
 9      else
10          return k
11  else
12      k ←RELUCTANTSEARCH(A, i, m, x)
13      if k = −1 then
14          return RELUCTANTSEARCH(A, m + 1, j, x)
15      else
16          return k
```

Question 12 Quelle équation récursive décrit précisément la complexité de l'algorithme dans le **meilleur** cas ?

- $T(n) = T(\lfloor n/2 \rfloor) + \Theta(1)$
- $T(n) = T(\lceil n/2 \rceil) + \Theta(1)$
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1)$
- $T(n) = T(\lfloor n/2 \rfloor) + \Theta(n)$
- $T(n) = T(\lceil n/2 \rceil) + \Theta(n)$
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$

Question 13 Quelle équation récursive décrit précisément la complexité de cet algorithme dans le **pire** cas ?

- $T(n) = T(\lfloor n/2 \rfloor) + \Theta(1)$
- $T(n) = T(\lceil n/2 \rceil) + \Theta(1)$
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1)$
- $T(n) = T(\lfloor n/2 \rfloor) + \Theta(n)$
- $T(n) = T(\lceil n/2 \rceil) + \Theta(n)$
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$

Question 14 La complexité **moyenne** de l'algorithme satisfait $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1)$. À quelle classe cela correspond-t-il ? (Indiquez la classe la plus précise.)

- $T(n) = \Theta(1)$
- $T(n) = \Theta(\log n)$
- $T(n) = \Theta(n)$
- $T(n) = \Theta(n \log n)$
- $T(n) = \Theta(n^2)$
- $T(n) = O(1)$
- $T(n) = O(\log n)$
- $T(n) = O(n)$
- $T(n) = O(n \log n)$
- $T(n) = O(n^2)$

Question 15 Pour $n > 0$, combien de fois le test "$x = A[i]$" de la ligne 3 est-il exécuté ?

- 1
- $\lfloor n/2 \rfloor$
- $\lceil n/2 \rceil$
- $n - 1$
- n
- $n + 1$
- $2n - 1$
- $2n$
- $2n + 1$
- $n^2 - 1$
- n^2
- $n^2 + 1$

6 QUICKSORT et MERGESORT (3 pts)

Question 16 Rappelez les complexités de MERGESORT et de QUICKSORT, puis citez un avantage que MERGESORT ne partage ni avec QUICKSORT ni avec HEAPSORT.