ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours d'Information, Calcul et Communication, sections MA et PH
Chappelier J.-C.

Information, Calcul et Communication (SMA/SPH) : Correction de l'Examen I

4 novembre 2022

SUJET 1

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez de deux heures quarante-cinq minutes pour faire cet examen (13h15 – 16h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée, **MAIS** ne mélangez pas les réponses de différentes questions !
   Ne joignez aucune feuilles supplémentaires ; **seul ce document sera corrigé**.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte 8 exercices indépendants sur 16 pages, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 150 points).
   Tous les exercices comptent pour la note finale.

Question 1 – Varia [23 points]

① **[1.5 points]** Soit x un nombre entier strictement positif dont la représentation binaire nécessite p bits. Combien de bits sont nécessaires pour représenter le nombre $y = 2^q \times x$ ($q \in \mathbb{N}$) ?
Justifiez *brièvement* votre réponse.

Réponse puis justification :

$p + q$: multiplier par 2^q revient à décaler la représentation binaire de q positions vers la gauche.

Commentaire : très peu réussi, beaucoup multipliant les nombres de bits ; aussi beaucoup disent que x vaut 2^p (alors que 2^p **n'**est **pas** représentable sur p bits).

② **[3 points]** Dans un système de représentation des nombres entiers sur 8 bits en complément à deux, combien vaut $30 \times (-7)$? Justifiez *brièvement* votre réponse.

Réponse et justification :

$30 \times (-7) = -210$ est plus petit que -128, il y a donc dépassement de capacité.

Par contre $256 + 30 \times (-7) = 46$ est représentable sur 8 bits. La réponse est donc 46.

Commentaire : très mal réussi ; certain(e)s disent que -210 n'est pas représentable et s'arrêtent, d'autres parlent du dépassement de capacité, sans réussir à donner la bonne valeur (ou sans donner de valeur du tout) ; d'autres enfin donnent uniquement le schéma binaire (ce qui est déjà bien) mais sans donner la **valeur** (il ne faut pas confondre valeur et représentation).

③ **[2.5 points]** Un moyen de représenter les couleurs consiste à représenter trois quantités : la proportion de rouge, de vert et de bleu. Sachant que la proportion de *chacune* des composantes (rouge, vert et bleu) est représentée avec trois octets, combien de couleurs distinctes est-il possible de définir avec cette représentation ?

Répondez tout d'abord sous forme d'une puissance de 2, puis donnez une valeur approchée sur la forme $a \times 10^b$ (p.ex. $8 \cdot 10^4$) ; justifiez ensuite *brièvement* vos réponses.

Réponses, puis justification : $2^{72} = 4 \cdot 2^{70} \simeq 4 \cdot 10^{21}$

Un octet = 8 bits = 2^8 possibilités, donc trois octets = 2^{24} possibilités, donc trois couleurs donnent ce nombre au cube.

Puis on utilise l'approximation $2^{10} \simeq 10^3$.

Commentaire : aussi assez mal réussi ; l'erreur principale était d'additionner les possibilités plutôt que des les multiplier ; aussi, pas mal se sont arrêté(e)s à la forme 2^a sans la convertir en base 10 ; aussi beaucoup de copies vides sur cette question.

④ **[6 points]** Vrai ou faux ? Répondez à droite, à la fin de chaque affirmation.

- Il existe des problèmes de la classe NP qui sont indécidables. **FAUX**
- Tout algorithme pouvant résoudre un problème de la classe P a une complexité forcément polynomiale. **FAUX**
- Si un problème est dans la classe NP, alors il n'est forcément pas dans la classe P. **FAUX**
- Décider si un nombre entier est un nombre premier est dans la classe NP. **VRAI**
- Décider si un nombre entier est un nombre premier est dans la classe P. **VRAI**
- Tout problème de la classe NP est aussi dans la classe P. **ON NE SAIT PAS**

— Tout problème indécidable est dans la classe NP. **FAUX**

— Si un problème est dans la classe P, alors il est aussi dans la classe NP. **VRAI**

Commentaire : Très bien réussi, *sauf* le fait qu'on ne sait pas si NP est dans P ou non-

⑤ **[5 points]** En utilisant une représentation à virgule flottante dans l'ordre signe, 4 bits d'exposant (codé simplement en binaire, sans décalage) et 5 bits de matisse, à quelle valeur décimale correspond `1001101101` ?

Justifiez *brièvement* votre réponse.

Réponse, puis justification : 1001101101 se décompose donc comme : `1 0011 01101`, ce qui se lit comme :

$$-2^3 \times 1,\texttt{01101} = -\left(8+2+1+\frac{1}{4}\right) = -11.25$$

Commentaire : Très bien réussi, sauf quelques fautes d'inattention.

⑥ **[5 points]** Par « colorier une carte », on veut dire « colorier tous les pays de cette carte de sorte qu'aucun pays n'ait la même couleur qu'un de ses voisins » ; et par « facile de décider », on veut dire que le problème de décision correspondant est dans la classe P.

Indiquez ci-dessous, à droite à la fin de chaque affirmation, si l'affirmation correspondante est vraie ou fausse.

— Toute carte peut être coloriée avec deux couleurs. **FAUX**

— Aucune carte peut être coloriée avec deux couleurs. **FAUX**

— Il est facile de vérifier si une carte a été correctement coloriée avec deux couleurs. **VRAI**

— Il est facile de décider si une carte peut être coloriée avec deux couleurs. **VRAI**

— On ne sait pas s'il est facile de décider si une carte peut être coloriée avec deux couleurs. **FAUX**

— Toute carte peut être coloriée avec trois couleurs. **FAUX**

— Aucune carte peut être coloriée avec trois couleurs. **FAUX**

— Il est facile de vérifier si une carte a été correctement coloriée avec trois couleurs. **VRAI**

— Il est facile de décider si une carte peut être coloriée avec trois couleurs. **ON NE SAIT PAS** (cf suivante ; on peut tolérer le FAUX ici, mais *uniquement* si on a répondu « vrai » à la question suivante)

— On ne sait pas s'il est facile de décider si une carte peut être coloriée avec trois couleurs. **VRAI**

— Toute carte peut être coloriée avec quatre couleurs. **VRAI**

— Aucune carte peut être coloriée avec quatre couleurs. **FAUX**

— Il est facile de vérifier si une carte a été correctement coloriée avec quatre couleurs. **VRAI**

— Il est facile de décider si une carte peut être coloriée avec quatre couleurs. **VRAI**

— On ne sait pas s'il est facile de décider si une carte peut être coloriée avec quatre couleurs. **FAUX**

Commentaire : Excellemment réussi.

Question 2 – Idées de suites [14 points]

① **[9 points]** Écrivez un algorithme *récursif* permettant, à partir de $u_0 \in \mathbb{Z}$, $u_1 \in \mathbb{Z}$ et $k \in \mathbb{N}$, de calculer la valeur u_k du $(k{+}1)$-ième terme de la suite :

$$u_{n+1} = 2\,u_n - 3\,u_{n-1} + 4 \qquad \text{pour } n \geq 1$$

Réponse :

Voici un premier algorithme, simple :

algoE
entrée : $a \in \mathbb{Z}$, $b \in \mathbb{Z}$, $k \in \mathbb{N}$ sortie : u_k

Si $k = 0$
  Sortir : a
Si $k = 1$
  Sortir : b
Sortir : $2\,$**algoE**$(a, b, k-1) - 3\,$**algoE**$(a, b, k-2) + 4$

Et voici une version plus élégante :

algoL
entrée : $a \in \mathbb{Z}$, $b \in \mathbb{Z}$, $k \in \mathbb{N}$ sortie : u_k
Si $k = 0$ **Sortir :** a **Sortir :** **algoL**$(b, 2\,b - 3\,a + 4, k-1)$

② **[5 points]** Quelle est la complexité de votre algorithme proposé en ① ? **Justifiez** votre réponse.

Réponse : La complexité de **algoE** est en $\Theta(2^k)$ comme vu en cours avec le calcul des coefficients du binôme (l'arbre des appels contient un arbre binaire de profondeur $\lfloor k/2 \rfloor$).

La complexité de **algoL** est en $\Theta(k)$ (l'arbre des appels est un arbre unaire de profondeur k).

Commentaire : Exercice peu réussi en moyenne. Il est dommage que plusieur(e)s ne suivent pas les conseils du cours, comme par exemple ne pas préciser les entrées (ou en oublier), ni la sortie de l'algorithme, ou ne pas penser à vérifier les conditions d'arrêt (arrêt à tous les coups ? trop oublient l'arrêt sur la « deuxième » valeur ($k = 0$ ou $k = 1$)). Manque de rigueur aussi dans les écritures : trop utilisent u_k et u_{k-1} sans les avoir définies ; ou utilisent u_0 ou u_1 comme si c'était des variables globales (qui sortent d'où ?).

Question 3 – Un peu de C++ [9 points]

Considèrez le code C++ suivant :

```
double f(int i, int j)
{
  if (i == 0) return 0.0;
  if (j == 0) return 1.0;
  if (j < 0)  return 1 / f(i, -j);
  return i * f(i, j-1);
}
```

① **[1.5 points]** `f(5,3)` renvoie la valeur $125 = 5 \times 5 \times 5 = 5^3$

② **[1.5 points]** `f(4,-5)` renvoie la valeur $4^{-5} = 1/4^5 = 1/1024 \simeq 10^{-3}$

③ **[2 points]** (Mis à part peut être un cas très particulier,) Quelle expression mathématique est réalisée par cette fonction ?

Réponse : i^j calculé récursivement ($i^j = i \cdot i^{j-1}$)

④ **[4 points]** Pour `i` fixé, quelle est, par rapport à `j`, la complexité de l'algorithme implémenté par `f()` ? Justifiez votre réponse.

Réponse et justification : $\Theta(j)$. En effet, toutes les opérations sont en $\Theta(1)$, seul compte donc le nombre d'appels ; et on en fait exactement $j + 1$ (pour $j > 0$, les autres cas étant triviaux).

Commentaire : Exercice globalement bien réussi dans l'ensemble. Plusieurs manquent de justification cependant, en particulier pour la complexité.

suite au dos ☞

Question 4 – Encore un peu plus C++ [18 points]

On souhaiterait que le programme C++ donné ci-dessous produise des sorties comme par exemple :

```
1_(10) = 1_(2) = 01_(8) = 0x1_(16)
2_(10) = 10_(2) = 02_(8) = 0x2_(16)
11_(10) = 1011_(2) = 013_(8) = 0xB_(16)
255_(10) = 11111111_(2) = 0377_(8) = 0xFF_(16)
```

Programme :

```
1   #include <iostream>
2   using namespace std;
3
4   char to_char(unsigned int n)
5   {
6     if (n < 10)  return n;
7     if (n <= 15) return 'A' + (n - 10);
8     return '?';
9   }
10
11  void base_print(unsigned int n, unsigned int base = 2)
12  {
13    if (n > 1) base_print(n / base, base);
14    return to_char(n % base);
15  }
16
17  unsigned int ask()
18  {
19    cout << "Entrez un entier (>= 0) : ";
20    unsigned int n(0);
21    cin >> n;
22  }
23
24  void print(unsigned int n)
25  {
26    cout << n << "_(10) = ";
27    cout << base_print(n);
28    cout << "_(2) = 0";
29    cout << base_print(n, 8);
30    cout << "_(8) = 0x";
31    cout << base_print(n, 16);
32    cout << "_(16)" << endl;
33  }
34
35  int main()
36  {
37    print(ask());
38    return 0;
39  }
```

Mais ce programme comporte plusieurs erreurs de programmation, possiblement de différente nature (syntaxe, déroulement, conception, méthodologie, ...).
Indiquez et corrigez toutes les erreurs (directement sur le code de la page d'en face).
Expliquez *brièvement* les erreurs/corrections à droite du code ou sur cette page ci.
On ôtera 1 point pour toute indication d'une erreur qui n'en est pas une.

Les cinq erreurs sont :

- ligne 6 : retour incorrect ; il manque le `'0'+` ;
- ligne 13 : la condition `n > 1` n'est pas la bonne ; ce devrait être `n >= base` (ajout d'une « colonne » de chiffres de plus) ;
- ligne 14 : ça ne doit pas être `return` (le type de retour est `void`), mais `cout <<` ;
- ligne 21.5 : il manque le `return n;` ;
- lignes 27, 29 et 31 : il ne faut pas de `cout <<` devant les `base_print()`.

Bien que ce ne soit pas strictement une erreur dans le code donné, on peut aussi signaler que tester si `base` n'est pas nul avant la division dans `base_print()` est une bonne pratique.

Commentaire : Exercice peu réussi. Parmi celles/ceux qui trouvent les erreurs, beaucoup perdent des points parce qu'ils/elles ne les corrigent pas. Le passage de valeur par défaut est trop souvent indiqué comme une erreur. Si l'on change le type de retour d'une fonction, il faut aussi faire attention à tous ses appels. Presque personne n'a vu l'erreur ligne 13 sur la condition de la base.

suite au dos ☞

Question 5 – Et toujours du C++ [15 points]

On dit qu'un nombre entier n est « *abondant* » s'il est strictement inférieur à la moitié de la somme de ses diviseurs.

Par exemple, 12 est abondant car ses diviseurs sont 1, 2, 3, 4, 6 et 12, dont la somme est 28, qui est plus grande strictement que 2×12.

① **[7.5 points]** Écrivez une fonction C++ `divisors_sum()` qui prend en paramètre un `int` et qui retourne la somme de ses diviseurs. Par exemple, `divisors_sum(12)` retourne `28`.

Pour rappel, `j` est diviseur de `i` si le modulo `i % j` est nul.

Réponse :

```
int divisors_sum(int n)
{
  int result(1+n); // 1 and n are always divisors
  for (int d(2); d < n; ++d) {
    if (n % d == 0) result += d;
  }
  return result;
}
```

Notes :

- il ferait aussi tout à fait sens d'utiliser des `unsigned int` ici (et dans la suite) ;
- on peut aussi optimiser la condition ci-dessus, par « `d <= n/2` » (on pourrait en plus mettre `n/2` dans un `const int`).

② **[3.5 points]** Écrivez une fonction C++ `is_abundant()` qui prend en paramètre un nombre entier et qui retourne `true` s'il est abondant et `false` sinon. Par exemple, `is_abundant(12)` retourne `true`.

Réponse :

```
bool is_abundant(int n)
{
  return divisors_sum(n) > 2*n;
}
```

③ **[4 points]** Écrivez une fonction C++ `print_all_abundant()` qui prend en paramètre un nombre entier et qui liste tous les nombres abondants inférieurs ou égaux à ce nombre.

Sachant que les premiers nombres abondants sont 12, 18, 20 et 24, l'appel
`print_all_abundant(11)` n'affichera rien,
`print_all_abundant(15)` affichera « `12,` »,
et `print_all_abundant(20)` affichera « `12, 18, 20,` ».

Réponse :

```
void print_all_abundant(int n)
{
  for (int i(1); i <= n; ++i) {
    if (is_abundant(i))
      cout << i << ", " ;
  }
  cout << endl; // facultatif
}
```

Commentaire : Exercice bien réussi. Les erreurs les plus courantes sont :

— attention à la porté des variables
— *return* sur des fonctions `void`
— encore diverses erreurs de syntaxe
— ce n'est pas vraiment une erreur, mais : on peut utiliser (ou retourner) directement une expression booléenne, sans obligatoirement la passer dans un `if` : `return X;` est bien plus simple que `if (X) return true; else return false;`.
De même (toujours pour un `X` booléen), on peut écrire `if(X)` au lieu de `if (X == true)` (même si `X` est un appel à une fonction qui retourne un booléen `;-)`)

suite au dos ☞

Question 6 – Que se passe-t-il ? [14 points]

On considère la machine de Turing ayant pour table de transition :

	0	1	ε
1	(2, ε, $-$)	(6, ε, $-$)	(12, ε, $+$)
2	(12, 0, $+$)	(12, 1, $+$)	(3, 0 , $+$)
3	(12, 0, $+$)	(12, 1, $+$)	(4, ε, $+$)
4	(4, 0, $+$)	(4, 1, $+$)	(5, ε, $+$)
5	(5, 0, $-$)	(5, 0, $-$)	(10, 0 , $-$)
6	(12, 0, $+$)	(12, 1, $+$)	(7, 1 , $+$)
7	(12, 0, $+$)	(12, 1, $+$)	(8, ε, $+$)
8	(8, 0, $+$)	(8, 1, $+$)	(9, ε, $+$)
9	(9, 1, $-$)	(9, 1, $-$)	(10, 1 , $-$)
10	(10, 0, $-$)	(10, 1, $-$)	(11, ε, $-$)
11	(11, 0, $-$)	(11, 1, $-$)	(1, ε, $+$)

① **[6 points]** Quel est l'état de la bande et la position de la tête de lecture lorsque la machine s'arrête, si elle a démarré avec sa tête de lecture positionnée comme suit :

$\cdots\varepsilon$	0	1	0	$\varepsilon\cdots$
	↑			

② **[8 points]** Justifiez votre réponse en deux ou trois phrase(s), puis dites en une phrase ce que fait cette machine.

Réponses : Cette machine ne termine pas (elle boucle à l'infini sur la séquence d'états $1, 2, 3, 4, 5, 5, 10, 11, 11$) ajoutant sans cesse des 0 devant le 11 qu'elle a écrit au début :

$\cdots\varepsilon$	0	ε	0	$\cdots$	0	1	1	$\varepsilon\cdots$
	↑							

L'état 1 « mémorise » le bit lu (choix du bloc à exécuter), que l'on recopie en le décalant à gauche (états (2 et 3) ou (6 et 7)), puis on va écrire un 0 tout en fin à droite (en sautant deux ε, états 4 et 5, ou (8 et 9) si l'on a lu un 1 au début), puis l'on revient au début de l'entrée de départ et on recommence si ce n'est pas ε (états 10 et 11). On arrive donc dans l'état suivant du ruban lors du premier recommencement :

$\cdots\varepsilon$	0	ε	1	0	ε	0	$\varepsilon\cdots$
			↑				

Mais en lisant maintenant un 1 au début, le passage par l'état 9 décrit ci-dessus va *écraser* l'ε qui séparait l'entrée initiale de la dernière recopie, ce qui fait que le retour (états 10 et 11) va se faire jusqu'au caractère recopié à gauche au début (états (2 et 3) ou (6 et 7)) :

$\cdots\varepsilon$	0	1	ε	0	1	1	$\varepsilon\cdots$
	↑						

sur laquelle la machine va redémarrer, entrant dans la séquence cyclique d'états $1, 2, 3, 4(, 4), 5, 5, 10, 11, 11$.

Cette machine ne fait donc rien d'intéressant car elle ne termine pas.

Commentaire : Évidemment beaucoup trop difficile en l'état. J'ai adapté mon barème.

Question 7 – Variations algorithmiques [36 points]

Note : lisez la sous-question ③ (au dos) avant de répondre à ①.

Soit L une liste de n nombres entiers strictement positifs tous différents les uns des autres et *triés* par ordre croissant, et soit a un nombre entier supérieur ou égal à 3.

On cherche à écrire un algorithme dont l'entrée soit la liste L et le nombre a et dont la sortie soit le nombre de paires (i, j) telles que $i < j$ et $L(i) + L(j) = a$.

Par exemple, pour $L = (2, 3, 5, 6, 7, 8, 10, 22)$ et $a = 10$, un tel algorithme sortira 2, puisque $2 + 8 = 3 + 7 = 10$ et qu'aucune autre paire de valeurs ne somme à 10.

① **[8 points]** Écrivez un algorithme pour résoudre ce problème.

Réponse :

Voici un premier algorithme, simple :

algo1
entrée : *une liste L triée [...] et un entier a [...]*
sortie : *nombre de paires [...]*

```
n ⟵ taille(L)
Si n ≤ 1
    Sortir : 0
s ⟵ 0
Pour i de 1 à n − 1
    Pour j de i + 1 à n
        Si L(i) + L(j) = a
            s ⟵ s + 1
Sortir : s
```

que l'on peut même écrire comme ceci avec les conventions du cours (mais faites quand même bien attention de vérifier que ce que vous écrivez est valide dans tous les cas) :

algo1
entrée : *une liste L triée [...] et un entier a [...]*
sortie : *nombre de paires [...]*

```
n ⟵ taille(L)
s ⟵ 0
Pour i de 1 à n
    Pour j de i + 1 à n
        Si L(i) + L(j) = a
            s ⟵ s + 1
Sortir : s
```

Commentaire : Assez bien réussi. Cependant encore trop de manque de rigueur, ou même ne serait-ce que suivre les conseils du cours : trop ne définissent ni entrée, ni sortie, ou ne définissent pas leurs

variables utilisées par la suite. Attention aussi à vos indices dans les boucles de parcourt. Enfin sur cet exercice spécifiquement : attention au cas $L(i) = a/2$, amenant certain(e)s à faussement compter la paire (i, i).

② **[4 points]** Quelle est la complexité de votre algorithme proposé en ① ? Justifiez votre réponse.

Réponse : La complexité de l'algorithme précédent est en $\Theta(n^2)$, avec n la taille de la liste. On parcourt en effet la liste en entier et, pour chaque position de ce parcourt (i dans l'algorithme) on reparcourt toute la liste (j) pour calculer la somme.

Une démonstration plus rigoureuse passerait par le calcul de $\sum_{i=1}^{n-1}(n-i)$.

Commentaire : Bien réussi aussi. Cependant, plusieurs justifications manquent de rigueur (arguments incomplets).

③ **[15 points]** Écrivez un algorithme de complexité temporelle $\Theta(n)$ pour résoudre le problème proposé.

Si votre réponse à ① est déjà en $\Theta(n)$, vous n'avez rien à faire ici (et serez, bien entendu, noté(e) sur la somme des points des deux sous-questions).

Réponse : Voici un algorithme en $\Theta(n)$ pour résoudre cette tâche :

```
algo2
entrée : une liste L triée [...] et un entier a [...]
sortie : nombre de paires [...]
    s ⟵ 0
    i ⟵ 1
    j ⟵ taille(L)
    Tant que i < j
        Tant que (i < j) et (L(i) + L(j) > a)
            j ⟵ j − 1
        Tant que (i < j) et (L(i) + L(j) < a)
            i ⟵ i + 1
        Tant que (i < j) et (L(i) + L(j) == a)
            j ⟵ j − 1
            i ⟵ i + 1
            s ⟵ s + 1
    Sortir : s
```

Commentaire : Très peu réussie, mais c'était la difficulté majeure de l'exercice.

④ **[5 points]** On considère maintenant un problème un peu différent en ne supposant plus que la liste L soit triée (ordre quelconque, donc) ; par contre, ses valeurs restent toutes différentes les unes des autres.

Proposez un nouvel algorithme pour ce nouveau problème. Ce nouvel algorithme doit être différent et ne pas utiliser celui proposé en ① sauf si cet algorithme proposé en ① était de complexité linéaire (vous pouvez alors l'utiliser). Vous pouvez aussi utiliser l'algorithme proposé en ③.

Réponse :

algo3
entrée : *une liste L [...] et un entier a [...]* sortie : *nombre de paires [...]*
$L' \longleftarrow$ **trier**(L) **Sortir : algo2**(L', a)

Commentaire : Moins bien réussie que je n'aurais pensé. Beaucoup trop peu pensent à utiliser un algorithme de tri.

⑤ **[4 points]** Quelle est la complexité de votre nouvel algorithme proposé en ④ ? Justifiez votre réponse.

Réponse : $\Theta(n \log n)$: le tri est en $\Theta(n \log n)$ et **algo2** est linéaire : $\Theta(n \log n + n) = \Theta(n \log n)$

Commentaire : Très bien réussie par celles/ceux qui ont fait la sous-question précédente.

suite au dos ☞

Question 8 – Quelques manipulations de listes [21 points]

Pour une liste L et une valeur e, on notera « $e \oplus L$ » la liste constituée de e (en premier), puis des éléments de L : $e \oplus L = (e, L(1), L(2), L(3), \cdots)$. Par ailleurs, on notera $L(i : j)$ la sous-liste de L composée de $L(i), \cdots, L(j)$ (avec $i \leq j$; la liste vide sinon).

On commence par considérer l'algorithme suivant :

bidule

entrée : L_1, L_2, *deux listes de nombres triés*
sortie : *? ? ?*

$n_1 \longleftarrow$ **taille**(L_1)
$n_2 \longleftarrow$ **taille**(L_2)
Si $n_1 = 0$
  Sortir : L_2
Si $n_2 = 0$
  Sortir : L_1
Si $L_1(1) < L_2(1)$
  Sortir : $L_1(1) \oplus$ **bidule**$\big(L_1(2 : n_1),\, L_2\big)$
Sortir : $L_2(1) \oplus$ **bidule**$\big(L_1,\, L_2(2 : n_2)\big)$

① **[1.5 points]** Que vaut **bidule**$\big((13, 15), (3, 10, 17)\big)$?

Réponse : $(3, 10, 13, 15, 17)$

② **[2 points]** Dites, en une courte phrase, ce que fait **bidule** et justifiez *brièvement* votre réponse.

Réponse : Il fait la fusion triée de deux listes triées. Il le fait en ajoutant devant, le plus petit des premiers éléments des deux listes, à la fusion du reste.

③ **[5 points]** En supposant **taille** et $\oplus$ en $\Theta(1)$, quelle est la complexité de **bidule** ? Justifiez votre réponse.

Réponse : Soit n la taille de l'entrée, somme des tailles des deux listes.

Tout l'algorithme est en $\Theta(1)$, sauf l'appel récursif. Cet appel récursif ne se produit qu'une seule fois à chaque appel et est lancé sur une entrée de taille $n - 1$. Si l'on note $C(n)$ la complexité recherchée et a le $\Theta(1)$ de la partie non récursive de l'algorithme [1], alors on a :

$$C(n) = a + C(n - 1)$$

et donc $C(n)$ est en $\Theta(n)$.

En notant $\lfloor x \rfloor$ la « partie entière par défaut » de x, c.-à-d. le plus grand entier inférieur ou égal à x,

1. c'est bien la même constante pour chaque appel.

on considère maintenant l'algorithme suivant :

machin
entrée : L, *une liste de nombres* sortie : *? ? ?*
$n \longleftarrow$ **taille**(L) **Si** $n \leq 1$ **Sortir :** L **Sortir : bidule**$\Big($**machin**$\big(L(1 : \lfloor \frac{n}{2} \rfloor)\big)$, **machin**$\big(L(\lfloor \frac{n}{2} \rfloor + 1 : n)\big)\Big)$

④ **[2 points]** Que vaut **machin**$\big(15, 13, 10, 17, 3\big)$?

Réponse : $(3, 10, 13, 15, 17)$

⑤ **[2.5 points]** Dites, en une courte phrase, ce que fait **machin** et justifiez *brièvement* votre réponse.

Réponse : Il s'agit d'un *tri* (en fait, c'est le tri fusion). Il trie la liste en faisant récursivement la fusion (*triée*) des deux moitiés de la liste.

⑥ **[8 points]** En supposant toujours **taille** en $\Theta(1)$, quelle est la complexité de **machin** ?
Justifiez votre réponse.

Soit $F(n)$ la complexité de **machin** pour une liste de taille n. On a donc :

$$F(n) = b + 2\,F(n/2) + C(n)$$

avec $C(n) \in \Theta(n)$, la complexitée de **bidule** : la taille totale des deux sous-listes traitées étant bien n.

En notant $C(n) = c\,n + D(n)$ pour mettre en évidence son terme dominant, ce nous donne :

$$\begin{aligned} F(n) &= a + 2\,F(n/2) + C(n) \\ &= 3a + 4\,F(n/4) + 2\,c\,n/2 + c\,n + 2\,D(n/2) + D(n) \\ &= 3a + 4\,F(n/4) + 2\,c\,n + ... \\ &= 7a + 8\,F(n/8) + 3\,c\,n + ... \\ &= 15a + 16\,F(n/16) + 4\,c\,n + ... \\ &... \\ &\simeq (n-1)\cdot a + n\,F(1) + \lceil \log_2(n) \rceil\ c\,n + ... \end{aligned}$$

où les « +... » sont négligeables par rapport au reste de la somme.

(Note : la dernière ligne est exacte si n est une puissance de 2 et sinon il faut remplacer n par 2^k dans les deux premiers termes, avec $k = \lceil \log_2(n) \rceil$ la partie entière par excès de $\log_2(n)$)

machin est donc en $\Theta\left(n \log(n)\right)$.

Une justification moins rigoureuse consiste à dire :
- à chaque fois, on appelle q fois **bidule** sur une liste de taille n/q, on a donc à chaque fois *au moins* un $\Theta(n)$;
- on doit faire $\log_2(n)$ appels (divise à chaque fois la liste en deux).

On a dont *au moins* $\Theta(n \log(n))$. Mais pour vraiment conclure, il me semble nécessaire de faire le calcul ci-dessus.

Commentaire : Exercice mitigé : bien réussi sur le début, beaucoup moins sur la fin, surtout le calcul et encore plus la justification de la complexité. Sur le départ : trop croient que **bidule** trie et confondent de ce fait **bidule** et **machin**. Sur la justification de la complexité : manques de rigueur : pas d'équation ou au moins un arbre d'appel, ou même dans le cas d'un arbre : quelle profondeur ? quelle largeur ? Et trop souvent la mention du reste en temps constant est oubliée.