Série 1: Solutions

1 Que font ces algorithmes?

a) La sortie de l'algorithme 1 est non dans ce cas, et celle de l'algorithme 2 vaut 9 ($= |22 - 13|$).

b) L'algorithme 1 indique si la liste L est ordonnée dans l'ordre croissant ou non.

La sortie de l'algorithme 2 donne quant à elle la plus grande différence en valeur absolue entre deux éléments consécutifs de la liste.

2 Quel est le bon algorithme?

Le bon algorithme est l'algorithme 2. L'algorithme 1 ne calcule que la somme des $n/2$ premiers nombres pairs; l'algorithme 3 calcule la somme des $2n$ premiers nombres entiers, et l'algorithme 4 produit une erreur à la première exécution de la ligne $s \longleftarrow s + i$, car la valeur de s n'est pas initialisée.

3 Réparez-moi ces algorithmes!

- Le problème de l'algorithme de Jeanne est le suivant: tout se passe bien jusqu'à ce que $i = 1$ et $j = 5$, moment auquel l'algorithme rajoute 5 à s et recommence la boucle, en passant à $i = 2$. A ce 2^{e} passage, comme la valeur de j n'a pas changé, elle est toujours égale à 5, donc l'algorithme n'entre pas dans la boucle "Tant que" et ajoute de nouveau 5 à s, et ainsi de suite. La sortie de l'algorithme sera donc $s = 5n$ et non le résultat escompté. Pour remédier à ce problème, il faut incrémenter j après avoir mis à jour la valeur de s, comme suit:

algorithme de Jeanne corrigé

entrée : *nombre entier positif n*
sortie : *nombre entier positif s*

```
s ⟵ 0
j ⟵ 1
Pour i allant de 1 à n
|   Tant que j n'est ni un multiple
|   de 5 ni un multiple de 7
|   |   j ⟵ j + 1
|   s ⟵ s + j
|   j ⟵ j + 1
Sortir : s
```

Ainsi, après avoir pris la valeur $j = 5$, j passe à 6, qui n'est plus ni un multiple de 5 ni de 7, et l'algorithme rentre dans la prochaine boucle "Tant que".

- Le problème de l'algorithme de Jean est le suivant: il existe des nombres qui sont à la fois des multiples de 5 et de 7 (35, par exemple). De tels nombres seront comptabiliés deux fois dans la somme s par l'algorithme. On peut réparer ce problème de deux manières:

algorithme de Jean corrigé, version 1

entrée : *nombre entier positif* n
sortie : *nombre entier positif* s

```
s ⟵ 0
i ⟵ 1
j ⟵ 1
Tant que i ≤ n
|   j ⟵ j + 1
|   Si j est un multiple de 5
|   |   s ⟵ s + j
|   |   i ⟵ i + 1
|   Sinon
|   |   Si j est un multiple de 7
|   |   |   s ⟵ s + j
|   |   |   i ⟵ i + 1
Sortir : s
```

algorithme de Jean corrigé, version 2

entrée : *nombre entier positif* n
sortie : *nombre entier positif* s

```
s ⟵ 0
i ⟵ 1
j ⟵ 1
Tant que i ≤ n
|   j ⟵ j + 1
|   Si j est un multiple de 5 ou de 7
|   |   s ⟵ s + j
|   |   i ⟵ i + 1
Sortir : s
```

4 Ecrivez un algorithme

a) Comme mentionné dans l'énoncé, il existe plusieurs façons de résoudre le problème. En voici deux:

algorithme 1

entrée : *liste* L *de nombres entiers, de taille* n*, nombre entier positif* x
sortie : *variable binaire* s *(oui/non)*

```
s ⟵ non
Pour i allant de 1 à n − 1
|   Pour j allant de i + 1 à n
|   |   Si |L(j) − L(i)| > x
|   |   |   s ⟵ oui
Sortir : s
```

algorithme 2

entrée : *liste* L *de nombres entiers, de taille* n*, nombre entier positif* x
sortie : *variable binaire* s *(oui/non)*

```
s ⟵ non
min ⟵ L(1)
max ⟵ L(1)
Pour i allant de 2 à n
|   Si L(i) < min
|   |   min ⟵ L(i)
|   Sinon, si L(i) > max
|   |   max ⟵ L(i)
Si max - min > x
|   s ⟵ oui
Sortir : s
```

Il en existe une troisième qui consiste à trier d'abord la liste L dans l'ordre croissant (nous verrons comment faire cela dans les semaines à venir), puis à comparer la différence $L(n) - L(1)$ (qui est égale à max - min ci-dessus à droite) avec la valeur x.

b) Même s'il est légèrement plus facile à écrire, l'algorithme 1 effectue considérablement plus d'opérations que l'algorithme 2. En effet, tandis que l'algorithme 2 ne parcourt qu'une seule fois la liste L, l'algorithme 1 parcourt toutes les paires (i, j) d'éléments de la liste, qui sont beaucoup plus nombreuses que les éléments eux-mêmes. En effet, pour n=1'000, le nombre d'élements de la liste vaut. . . 1'000, tandis que le nombre de paires d'éléments vaut approximativement 500'000. Nous y reviendrons la semaine prochaine.

Note: Si on utilise la troisième possibilité ci-dessus qui consiste à trier d'abord la liste L, alors l'efficacité de l'algorithme dépend directement de l'efficacité de l'algorithme de tri utilisé.

5 Pour le plaisir: un algorithme de tri inhabituel*

Quand il sort de la caverne, chaque lutin peut voir la couleur des bonnets de ses compagnons qui sont déjà sortis. Si tous les lutins déjà à l'extérieur de la caverne ont le bonnet de la même couleur, alors il va se placer à une extrémité de la ligne.

Sinon, supposons que les autres lutins soient déjà rangés dans le bon ordre, avec (par exemple) tous les bonnets rouges à gauche et les bonnets bleus à droite. Alors, il suffit que le lutin se place à l'endroit du changement de couleur de bonnet, entre le dernier lutin avec un bonnet rouge et le premier avec un bonnet bleu:

```
         x
         |
...RRRRRRR BBBBBBBBB ...
```