Tri, correction, terminaison

On s'intéresse aux deux problème algorithmiques suivant :

Problème du tri
Entrées : Une liste L d'entiers deux à deux distincts
Sortie : Une liste L' triée par ordre croissant contenant les mêmes éléments que L.

Problème du tri en place
Entrées : Une liste L d'entiers deux à deux distincts
Effet : L est triée par ordre croissant et contient les mêmes éléments qu'au départ.

1 Tri par sélection en place

On étudie dans cette section un algorithme de tri nommé *tri par sélection*, qui résout le problème du tri en place. Le principe de l'algorithme ayant en entrée un liste L de taille n est le suivant :

- prendre le minimum de L et échanger sa place avec $L[0]$.
- prendre le minimum de $L[1:]$ et échanger sa place avec $L[1]$.
- ... Répéter l'opération $n-1$ fois.

Voici l'algorithme décrit en pseudo-code.

Algorithme Algorithme `echanger`
ENTRÉES: L une liste de taille n, $0 \leq i, j < n$
SORTIES/EFFETS: ...

$t \leftarrow L[i]$
$L[i] \leftarrow L[j]$
$L[j] \leftarrow t$
renvoyer `None`

Algorithme `deplacer_minimum`
ENTRÉES: L une liste de taille n d'entiers deux à deux distincts et $0 \leq i < n$.
SORTIES/EFFETS: L est modifiée de telle manière à ce que :

- Les éléments de L aux indices $0 \leq j < i$ n'ont pas été modifiés,
- Les éléments de L aux indices $i \leq j < n$ sont les mêmes (à permutation près),
- Les éléments de L aux indices $i < j < n$ sont plus grands strictement que $L[i]$.

$j \leftarrow i+1$
tant que $j < n$ **faire** /* *B1* */
  si $L[j] < L[i]$ **alors**
    `echanger`(L,i,j)
  fin si
  $j \leftarrow j+1$
fin tant que
renvoyer `None`

Algorithme Algorithme `tri_selection`
ENTRÉES: L une liste de taille n d'entiers deux à deux distincts.
SORTIES/EFFETS: L est modifiée de telle manière qu'elle soit triée par ordre croissant et contiennent exactement les mêmes éléments qu'au départ.

$i \leftarrow 0$
tant que $i < n-1$ **faire** /* *B2* */
  `deplacer_minimum(L,i)`
  $i \leftarrow i+1$
fin tant que
renvoyer `None`

Exercice 1. *Échanger*

Écrire la post-condition de l'algorithme `echanger`.

Implémentez l'algorithme en Python.

Exercice 2. *Terminaison*

Montrez la terminaison de l'algorithme `deplacer_minimum`.

Pour ce faire, Déterminer un variant pour la boucle $B1$.

Montrez ensuite la terminaison de l'algorithme de tri sélection.

Pour ce faire, déterminez un variant pour la boucle $B2$.

Exercice 3. *Correction `deplacer_minimum`*

On définie l'invariant $I1$ pour la boucles $B1$ par :

1. i < j $\leq nLeslmentsde$L$auxindices 0 \leq x < i$ n'ont pas été modifiés,
3. Les éléments de L aux indices $i \leq x < n$ sont les mêmes (à permutation près),
4. Les éléments de L aux indices $i < x < j$ sont plus grands strictement que $L[i]$.

Montrez que $I1$ est bien un invariant de $B1$ et montrez la correction de l'algorithme.

Exercice 4. *Correction `tri_selection`*

On définie l'invariant $I2$ pour la boucles $B2$ par :

1. $n = i = 0$ ou $0 < i < n$
2. L contient toujours les mêmes éléments qu'on départ.
3. $L[:i]$ est triée par ordre croissant.
4. Si $i > 0$, les éléments de L aux indices $i \leq x < n$ sont plus grands strictement que $L[i-1]$.

Montrez que $I2$ est bien un invariant de $B2$ et montrez la correction de l'algorithme.

Exercice 5. *Implémentation et jeu de test*

Implémentez l'algorithme de tri sélection en Python.

Confectionnez un jeu de test.

Testez votre implémentation.

2 Tri fusion

On étudie dans cette section un algorithme de tri nommé *tri fusion* (ou *tri dichotomique*), qui résout le problème du tri. Le principe de l'algorithme ayant en entrée un liste de taille n est le suivant :

1. Si la liste possède un élément ou moins, il est trié.
2. Sinon couper la liste en 2.
3. Trier récursivement la partie gauche et droite de la liste.
4. Fusionner les deux liste triée en une seule liste triée.

Voici l'algorithme décrit en pseudo-code.

Algorithme Algorithme `fusion`

ENTRÉES: L_1 et L_2 deux listes de taille n_1 et n_2 d'entiers triés par ordre croissant. La liste $L_1 + L_2$ contient des entiers deux à deux distincts.

SORTIES/EFFETS: Une liste L triée par ordre croissant et contenant exactement les éléments de L_1 et L_2

```
i1 ← 0
i2 ← 0
L ← []
tant que i1 < n1 et i2 < n2 faire /* B3 */
   si L1[i1] < L2[i2] alors
      Ajoutez L1[i1] à la fin de L
      i1 ← i1 + 1
   sinon
      Ajoutez L2[i2] à la fin de L
      i2 ← i2 + 2
   fin si
fin tant que
si i1 = n1 alors
   L ← L + L2[i2 :]
sinon
   L ← L + L1[i1 :]
fin si
renvoyer L
```

Exercice 6. *Tri fusion*

Proposez un algorithme *récursif* de tri fusion résolvant le problème du tri et utilisant l'algorithme de fusion ci-dessus.

En supposant que l'algorithme de fusion termine et est correct, prouver la terminaison et la correction de votre algorithme.

Implémentez l'algorithme de fusion et votre algorithme de tri fusion.

Réalisez un jeux de test, puis testez votre implémentation.

Exercice 7. *Terminaison de Fusion*

Montrez que l'algorithme fusion termine en exposant un variant de la boucle $B3$.

Exercice 8. *Correction de Fusion*

Prouvez la correction de la boucle $B3$.

Pour trouvez l'invariant de $B1$ on pourra essayer de répondre aux questions suivantes à chaque itération de boucle :

1. Comparez $i_1 + i_2$ à 0 et $n_1 + n_2$, i_1 à 0 et n_1, et i_2 à 0 et n_2.
2. Est-ce que L_1, L_2 et L sont triées ?
3. Comparez le contenu de L et le contenu de $L_1[: i_1]$ et $L_2[: i_2]$
4. Que dire des éléments de $L[i_1 :]$ et $L[i_2 :]$ par rapport à ceux de L ?

Pour terminer la preuve de correction on pourra supposer $i_1 = n_1$, l'autre cas étant symétrique.

3 Exercices supplémentaires

Exercice 9. *Tri sélection récursif*

Proposez une version récursive du tri sélection pour résoudre le problème du tri (on renvoie une copie de la liste triée).

Prouvez sa terminaison et sa correction.

Exercice 10. *Tri à bulle*

Voici l'algorithme du tri à bulle qui résout le problème du tri en place.

Algorithme Algorithme `tri_bulle`

ENTRÉES: L une liste de taille n d'entiers deux à deux distincts.

SORTIES/EFFETS: L est modifiée de telle manière qu'elle soit triée par ordre croissant et contiennent exactement les mêmes éléments qu'au départ.

```
i ← 0
tant que i < n − 1 faire /* B1 */
    j ← i + 1
    tant que j < n faire /* B2 */
        si L[j − 1] > L[j] alors
            echanger(L,j-1,j)
        fin si
        j ← j + 1
    fin tant que
    i ← i + 1
fin tant que
renvoyer None
```

Prouvez la terminaison et la correction de l'algorithme.

Implémentez l'algorithme en Python.