Tri, complexité

On s'intéresse à la complexité des deux algorithmes du TP précédent.

1 Tri par sélection en place

Algorithme Algorithme `echanger`

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

```
1. t ← L[i]
2. L[i] ← L[j]
3. L[j] ← t
4. 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 $L[i]$.

```
1. j ← i + 1
2. tant que j < n faire /* B1 */
3.     si L[j] < L[i] alors
4.         echanger(L,i,j)
5.     fin si
6.     j ← j + 1
7. fin tant que
8. 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.

```
1. i ← 0
2. tant que i < n − 1 faire /* B2 */
3.     deplacer_minimum(L,i)
4.     i ← i + 1
5. fin tant que
6. renvoyer None
```


Exercice 1. *echanger*

Quel est la complexité asymptotique de l'algorithme `echanger` en fonction de n la longueur de la liste en entrée ?

Exercice 2. *deplacer_minimum*

Quel est la complexité asymptotique de l'algorithme `deplacer_minimum` en fonction de n la longueur de la liste en entrée ?

Exercice 3. *tri_selection*

Quel est la complexité asymptotique de l'algorithme `tri_selection` en fonction de n la longueur de la liste en entrée ?

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

```
1. i1 ← 0
2. i2 ← 0
3. L ← []
4. tant que i1 < n1 et i2 < n2 faire /* B3 */
5.     si L1[i1] < L2[i2] alors
6.         Ajoutez L1[i1] à la fin de L
7.         i1 ← i1 + 1
8.     sinon
9.         Ajoutez L2[i2] à la fin de L
10.        i2 ← i2 + 2
11.    fin si
12. fin tant que
13. si i1 = n1 alors
14.     L ← L + L2[i2 :]
15. sinon
16.     L ← L + L1[i1 :]
17. fin si
18. renvoyer L
```

Exercice 4. *Tri fusion*

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

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

Exercice 5. *Complexité `fusion`*

Quel est la complexité asymptotique de l'algorithme `fusion` en fonction de n_1 et n_2 les longueurs des listes L_1 et L_2 prises en entrée ?

Exercice 6. *Complexité `tri_fusion`*

On note $C(n)$ la complexité asymptotique de l'algorithme `tri_fusion` en fonction de n, la taille de la liste en entrée.

1. Exprimez $C(n)$ en fonction de $C(\lfloor \frac{n}{2} \rfloor)$, $C(\lceil \frac{n}{2} \rceil)$ et F la complexité de l'algorithme `fusion`.
2. En supposant que $n = 2^k$, calculer $C(n)$.
3. Exprimez $C(n)$ dans le cas général.

3 Recherche Dichotomique

Exercice 7. *Algorithme récursif*

Proposez une version récursive de l'algorithme de recherche dichotomique dans une liste triée.

Exercice 8. *Analyse*

Prouvez la terminaison, la correction de l'algorithme. Vérifiez que la complexité asymptotique de l'algorithme est en $\mathrm{O}(\log(n))$, avec n la longueur de la liste en entrée.

Exercice 9. *Implémentation*

Implémentez l'algorithme.