10 Exercices avec les tris

$\mathcal{X}$10.1 Tri bébé

a. Quel est le minimum de comparaisons pour trier $n = 3$ éléments ? ▶ Donnez un algorithme pour trier trois éléments a, b, c avec ce nombre minimum de comparaisons.

b. Quel est le minimum de comparaisons pour trier $n = 4$ éléments ? ▶ Donnez un algorithme pour trier quatre éléments a, b, c, d avec ce nombre minimum de comparaisons.

$\mathcal{X}$10.2 Séquence bitonique

Le tableau $A[0 \ldots n-1]$ est appellé **bitonique** s'il existe un indice $i \in \{0, 1 \ldots, n-1\}$ tel que

$$A[0] \leq A[1] \leq \cdots \leq A[i]; \text{ et}$$
$$A[i] \geq A[i+1] \geq A[i+2] \geq \cdots \geq A[n-1].$$

▶ Donnez un algorithme pour trier A dans l'ordre croissant en temps $O(n)$ au pire.
Indice: Utiliser un tableau auxiliaire.

$\mathcal{X}$10.3 Tri local

On veut trier un tableau $A[0..n-1]$. Le tri correspond à une permutation π des indices $0, \ldots, n-1$ telle que $A[\pi(0)] \leq A[\pi(1)] \leq A[\pi(2)] \leq \cdots \leq A[\pi(n-1)]$.

Supposons qu'on sait que le tableau est «presque trié» dans le sens qu'il existe une constante Δ avec $|i - \pi(i)| \leq \Delta$ pour tout i. Dans d'autres mots, le tri déplace chaque élément par tout au plus Δ position. Il n'est pas difficile de voir que le tri par insertion prend $O(n\Delta)$ temps sur un tableau presque trié.

a. ► Modifiez le tri par sélection pour qu'il prenne $O(n\Delta)$ temps (Δ est fourni comme argument).

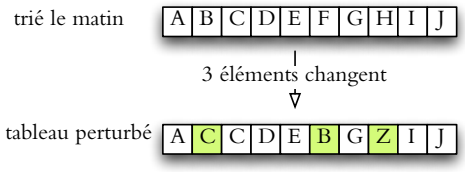
b. ► Développez un algorithme de tri qui prend le «désordre» Δ comme argument pour trier le tableau en $O(n \log \Delta)$ temps. L'algorithme ne doit utiliser que $O(\Delta)$ espace de travail.

Indice: employer une file à priorités de Δ éléments au plus, en liaison avec une fenêtre de taille Δ glissant au long du tableau.

$\mathcal{X}$10.4 Tri d'un tableau modifié

Supposons qu'on a un tableau $A[0..n-1]$, trié en ordre croissant le matin. On sait que $k > 0$ éléments ont changé de valeur pendant la journée, mais on ne sait pas lesquels. ► Donnez un algorithm pour trier $A[0..n-1]$ le soir, en $\Theta(n + k \log k)$ temps.

Indice: identifier d'abord $\Theta(k)$ éléments malplacés qu'on peut copier dans un petit tableau et trier séparamment.



$\mathcal{X}$10.5 Multi-fusion

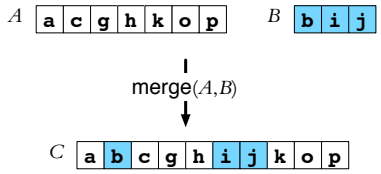
Dans le tri par fusion, on fusionne deux listes en un temps linéaire. ▶ Donnez un algorithme pour fusionner k listes triées dans un temps $O\big(\ell \log k\big)$ quand la longueur totale des listes est ℓ (donc ℓ est la longueur du résultat).

Indice: utiliser un tas de taille k, ou développer une solution récursive (par k).

$\mathcal{X}$10.6 Borne inférieure pour fusion

Supposons qu'on veut fusionner deux tableaux $A[0..n-1]$ et $B[0..m-1]$ triés dans l'ordre croissant des éléments. La fusion crée un troisième tableau qui contient tous les éléments de A et B, dans l'ordre croissant. L'algorithme a le droit de comparer un élément $A[i]$ à un autre élément $B[j]$ à la fois. Soit $C(n,m)$ le nombre de comparaisons au pire qu'un algorithme utilise pur fusionner des tableaux de tailles $m \leq n$.

Indice: examiner la hauteur minimale d'un arbre de décision pour le problème.



(a) Borne inférieure. ▶ Démontrez que $C(n,m) \geq \lg \binom{n+m}{n}$ pour tout algorithme de fusion.

(a) Tableaux de même taille. ▶ Démontez que la procédure du tri par fusion est très proche à la borne inférieure quand $m = n$.

Indice: utiliser la formule de Stirling pour $\binom{2n}{n} = \frac{(2n)!}{(n!)\cdot(n!)}$.

$\mathcal{X}$10.7 Vis-écrou

On a un ensemble de n vis et n écrous. On sait que les vis sont de tailles différentes, et qu'il existe exactement un écrou pour chaque vis. On ne peut comparer qu'un écrou E à une vis V à la fois (donc pas de comparaisons écrou-écrou ou vis-vis), pour vérifier que soit $E < V$, soit $E = V$, soit $E > V$.

a. ▶ Donnez un algorithme pour ranger $n = 2$, et un autre pour $n = 3$ paires de vis-écrou.

b. ▶ Donnez un algorithme pour trouver l'écrou à chaque vis avec $O(n \log n)$ comparaisons *en moyenne*.

Indice: adapter l'astuce de pivotage de Quicksort.

c. ▶ Quel est le nombre minimal de comparaisons vis-écrou qu'un algorithme de tri déterministe doit faire ?

REMARQUE. Il est très difficile de trouver un algorithme déterministe pour ce problème avec $O(n \log n)$ temps de calcul. En fait, c'est un problème ouvert de recherche.

$\mathcal{X}$10.8 Tri rapide sans récursion

On décrit le tri rapide souvent comme un algorithme récursif :

```
    Algo QUICKSORT(A, l, r)                    // sorts A[l..r]
1.  if r − l < 1 then return
2.  i ← PARTITION(A, l, r)    // partitioning with a pivot at i
3.  QUICKSORT(A, l, i − 1)
4.  QUICKSORT(A, i + 1, r)
```

(a) Tri rapide avec une pile. ▶ Donnez une implantation du tri rapide sans récursion, en utilisant une pile explicitement. Il n'est pas nécessaire d'implanter PARTITION().

Indice: Empiler/dépiler les indices (l, r) des sous-tableaux dans une boucle.

(b) Tri rapide avec une queue. Est-ce que l'algorithme de **(a)** s'éxécute correctement si on utilise une queue FIFO au lieu de la pile ? ▶ Justifiez votre réponse.

(c) Tri rapide avec une petite pile. ▶ Contrôlez la taille maximale de la pile : donnez un algorithme itératif en **(a)** qui utilise $O\big(\lg(n)\big)$ mémoire au pire.

Indice: On peut exécuter les lignes 3 et 4 dans n'importe quel ordre — l'un ou l'autre peut être en position terminale ce qu'on peut transformer en une boucle **while**.