

TP n°4

Algo de tri

Exercice 1 : Tri à bulles

Il doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface.

L'algorithme parcourt le tableau et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.

Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, à sa position définitive, il n'a plus à être traité.

Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément. Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive. Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive.

Exemple avec (5 4 1 8 2) :

Étape 1 :

```
(5 4 1 8 2) -> (4 5 1 8 2). Échange, car 5 > 4.  
(4 5 1 8 2) -> (4 1 5 8 2). Échange, car 5 > 1.  
(4 1 5 8 2) -> (4 1 5 8 2). Pas d'échange, car 5 < 8.  
(4 1 5 8 2) -> (4 1 5 2 8). Échange, car 8 > 2.
```

Étape 2 :

```
(4 1 5 2 8) -> (1 4 5 2 8). Échange, car 4 > 1.  
(1 4 5 2 8) -> (1 4 5 2 8). Pas d'échange.  
(1 4 5 2 8) -> (1 4 2 5 8). Échange, car 5 > 2.
```

Étape 3 :

```
(1 4 2 5 8) -> (1 4 2 5 8). Pas d'échange.  
(1 4 2 5 8) -> (1 2 4 5 8). Échange, car 4 > 2.
```

Étape 4 :

```
(1 2 4 5 8) -> (1 2 4 5 8). Pas d'échange.
```

Exercice 2 : Tri comptage

Méthode de tri relativement simple et efficace par dénombrement. Cet algorithme de tri convient exclusivement aux données constituées de nombres entiers compris entre une borne min et une borne max connues.

Le principe repose sur la construction de l'histogramme des données en quantifiant chacune des possibilités. Une fois cet histogramme construit, il suffit de le parcourir pour construire le tableau trié.

Exemple :

Tableau initial :

i	0	1	2	3	4	5	6
tab[i]	9	4	6	4	7	2	7

Tableau qui compte le nombre d'occurrences :

i	0	1	2	3	4	5	6	7	8	9
tabComptage[i]	0	0	1	0	2	0	1	2	0	1

Tableau trié :

i	0	1	2	3	4	5	6
tabTrié[i]	2	4	4	6	7	7	9

Exercice 3 : Quick Sort

Voici un déroulé pour trier ce tableau : quickSort(0, 6);

6	3	7	9	1	5	4
---	---	---	---	---	---	---

Le dernier élément est le pivot 4, on va trouver sa position exacte dans le tableau de [0 .. 6].

6	3	7	9	1	5	4
---	---	---	---	---	---	---

On compare le pivot au premier élément $6 > 4$. Cet élément est plus grand, il vient se ranger après le pivot, et le premier élément devant le pivot (5 ici) vient prendre la place de 6.

5	3	7	9	1	4	6
---	---	---	---	---	---	---

Du fait qu'il y a eu une permutation, on ne change pas la case à comparer. On compare toujours le pivot à cette case : $5 > 4$. Cet élément est plus grand, il vient se ranger après le pivot, et le premier élément devant le pivot (1 ici) vient prendre sa place.

1	3	7	9	4	5	6
---	---	---	---	---	---	---

On compare 1 à 4. 1 n'est pas plus grand que 4 alors on décale la case à comparer.

1	3	7	9	4	5	6
---	---	---	---	---	---	---

On compare 3 à 4. 3 n'est pas plus grand que 4 alors on décale la case à comparer.

1	3	7	9	4	5	6
---	---	---	---	---	---	---

On compare 7 à 4. 7 est plus grand, on le place après et on permute.

1	3	9	4	7	5	6
---	---	---	---	---	---	---

On compare 9 à 4. 9 est plus grand, il est devant le pivot, on permute les deux seulement.

1	3	9	4	7	5	6
---	---	---	---	---	---	---

On a le pivot qui est passé avant la case testée, on s'arrête.

1	3	4	9	7	5	6
---	---	---	---	---	---	---

On arrive à la fin :

Le pivot est exactement au bon endroit. On y touchera plus, on sait que tout ce qui est à gauche est plus petit et ce qui est à droite est plus grand, c'était l'objectif.

Il faut maintenant :

- répéter cette opération pour la partie du tableau qui est à gauche (seulement si besoin),
- répéter cette opération pour la partie du tableau qui est à droite (seulement si besoin),

Dans le cas de l'exemple :

On lance la même méthode mais pour les deux "sous-tableaux", ici dans notre exemple le pivot finit à la position 2, on lance `quickSort(0, 1)`; et `quickSort(3, 6)` car on ne sait pas si ces deux sous parties contenant au moins deux éléments sont triés.

Sur la première partie l'algorithme sera relancée sur :

1	3					
---	---	--	--	--	--	--

Sur la seconde partie l'algorithme sera relancée sur :

			9	7	5	6
--	--	--	---	---	---	---

Le traitement est le même, et l'algorithme sera relancée sur ou nouvelle sous partie du tableau tant qu'il y en aura besoin.

Écrire le programme correspondant.

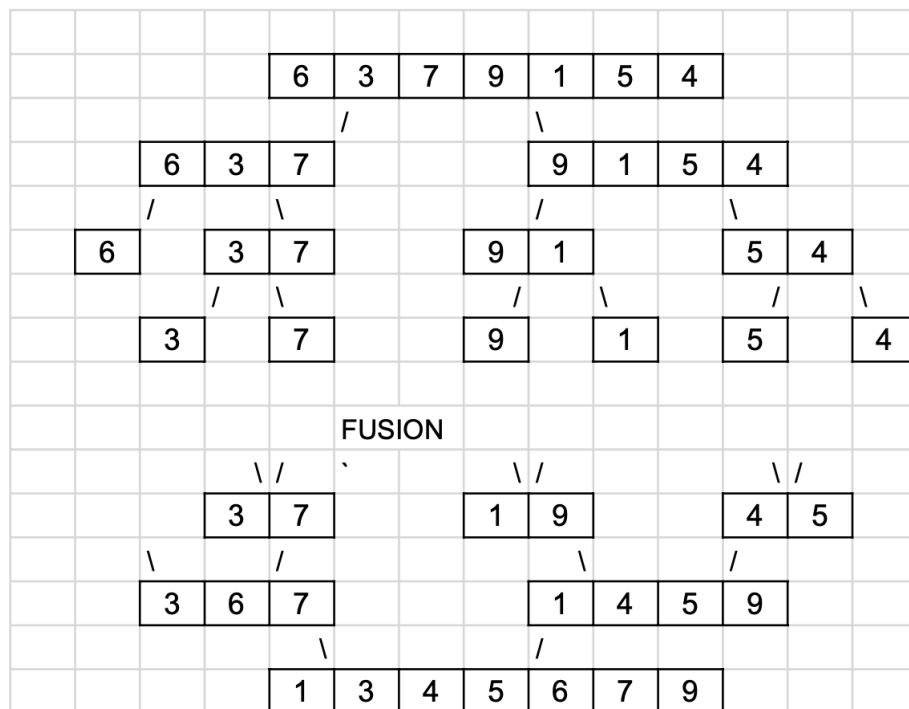
Exercice 3 : Merge Sort

Cet algorithme se distingue par deux grandes deux étapes : **diviser**, puis **fusionner les sous-tableaux triés**.

Le principe du **Merge Sort** est donc :

1. **Diviser** le tableau en deux.
2. Continuer à diviser chaque partie jusqu'à obtenir des tableaux d'un seul élément. Un tableau contenant un seul élément est considéré comme **déjà trié**.
3. **Fusionner** les petits tableaux deux par deux en conservant l'ordre.
4. Continuer les fusions jusqu'à reconstituer le tableau complet.

On peut représenter notre exemple ainsi :



Première étape : mergeSort(0, 6);

Le tableau contient 7 éléments.

[6, 3, 7, 9, 1, 5, 4]

On le coupe en deux parties.

On obtient donc :

- une partie de gauche avec [6, 3, 7]
- Et une partie de droite avec [9, 1, 5, 4]

Ces deux sous-tableaux seront à fusionner plus tard.

On va maintenant appliquer exactement la même méthode à chacune des deux parties.

On lance :

mergeSort(0, 2);

mergeSort(3, 6);

Découpage de la partie gauche : mergeSort(0, 2);

La partie gauche contient :

[6, 3, 7]

On la découpe à nouveau en deux.

On obtient donc :

- une partie de gauche avec [6]
- Et une partie de droite avec [3, 7]

Ces deux sous-tableaux seront à fusionner plus tard.

La partie de gauche avec 6 est seule, on ne relance pas, on considère qu'elle est triée.

On relance la partie de droite :

mergeSort(1, 2);

Découpage de la droite de la partie gauche : mergeSort(1, 2);

La partie droite de la partie gauche contient :

[3, 7]

On la découpe à nouveau en deux.

On obtient donc :

- une partie de gauche avec [3]
- Et une partie de droite avec [7]

Ces deux sous-tableaux seront à fusionner plus tard.

Les deux parties n'ont qu'un seul élément, on ne relance pas, on considère qu'elles sont triées.

On peut donc les fusionner.

On Fusionne(1, 2)

La partie contient :

[3] et [7]

On réécrit le tableau en fusionnant les deux parties de manière triée. Ici 3 est déjà à gauche de 7. On ne change rien.

On remonte pour fusionner maintenant [6] à [3, 7], On réécrit le tableau en plaçant le 3 en premier (parce qu'il est plus petit que le 6) puis le 6 (parce qu'il est plus petit que le 7) et le 7 car il n'y a plus rien dans la partie de gauche (6 a déjà été placé).

On obtient : [3, 6, 7]

On remonte mais on ne peut pas encore faire la fusion avec la partie (3,6) car la division n'a pas encore été faite.

C'est le moment et on lance :

Découpage de la partie droite : mergeSort(3, 6);

La partie contient :

[9, 1, 5, 4]

De la même manière, cette partie va se diviser pour se fusionner triée. Et obtenir : [1, 4, 5, 9]

On peut maintenant fusionner les deux parties initiales [3, 6, 7] et [1, 4, 5, 9].

Indice à la mise en place

- écrire la méthode *mergeSort(int[] tableau, int debut, int fin)* ;
- déterminer le milieu du tableau ;
- appeler récursivement *mergeSort* sur la partie gauche ;
- appeler récursivement *mergeSort* sur la partie droite ;
- écrire une méthode permettant de **fusionner deux parties déjà triées** ;
- utiliser cette méthode de fusion pour construire progressivement le tableau trié.

Dans notre exemple, le premier appel est *mergeSort(tableau, 0, 6);*

Puis l'algorithme devra automatiquement effectuer les appels récursifs nécessaires jusqu'à obtenir des sous-tableaux d'un seul élément.

Écrire le programme correspondant.