Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2021 – 2022
ATL

ATL – Ateliers logiciels

Mise en pratique des Threads

Comparaison de la complexité des algorithmes de tri

Consignes

Dans cet exercice vous allez implémenter plusieurs algorithmes de tri afin d'en comparer la complexité. Vous ajouterez au fur et à mesure du développement l'utilisation de `Threads` afin d'améliorer la vitesse d'exécution.

1 Présentation

Vous devez développer un logiciel qui permet de trier 10 listes d'entiers via un algorithme et de voir le nombre d'opérations effectuées pour chaque tri.

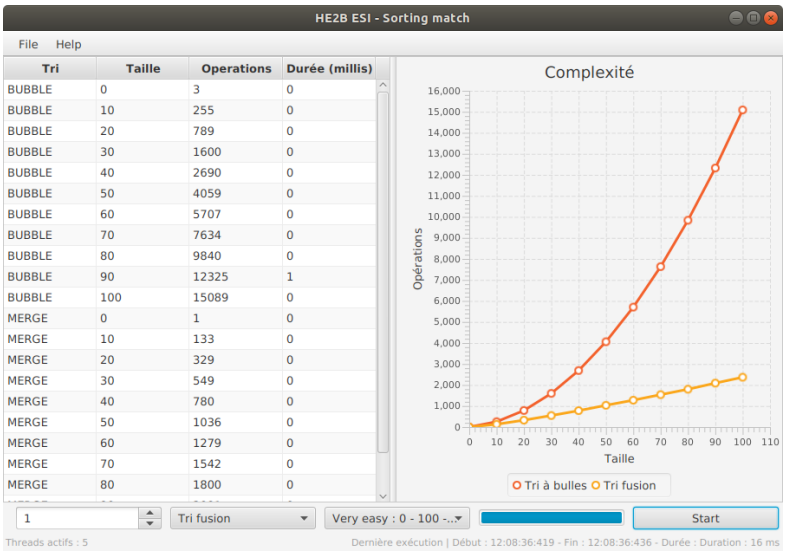


FIGURE 1 – Illustration de l'application

La vue permet de sélectionner :

- ▷ le nombre de `Threads` disponibles pour traiter les données (1 par défaut) ;
- ▷ l'algorithme de tri à exécuter : **Tri à bulles** ou **Tri fusion** ;
- ▷ la quantité maximale d'entiers à trier (Very easy : 100, Easy : 1 000,...).

Version du document
2 février 2022

Si on choisit le **Tri à bulles** pour **100 entiers** avec **1 Thread**, voici ce qui se produit lorsqu'on appuie sur le bouton `Start` :

- ▷ un tableau (ou une liste) ne contenant aucun élément est généré(e) ;
- ▷ aucun tri n'est nécessaire, le résultat est affiché sur la vue ;
- ▷ un tableau (ou une liste) de 10 entiers est généré(e) ;
  - ▷ le tableau est passé en paramètre à l'algorithme de tri en bulles ;
  - ▷ durant le tri, on compte le nombre d'opérations effectuées ;
  - ▷ lorsque les données sont triées, le résultat est affiché sur la vue
- ▷ un tableau (ou une liste) de 20 entiers est généré(e) ;
  - ▷ le tableau est passé en paramètre à l'algorithme de tri en bulles ;
  - ▷ durant le tri, on compte le nombre d'opérations effectuées ;
  - ▷ lorsque les données sont triées, le résultat est affiché sur la vue
- ▷ ...

2 La vue

Commencez par créer le projet `maven SortingRace`. Ajoutez ensuite les packages nécessaires à l'architecture `MVC`. Utilisez le fichier `sort.fxml` disponible sur `PoEsi` et implémentez la vue du logiciel.

Informations sur la vue

Vous trouverez sur cette vue, les composants suivants :

- ▷ un `TableView` qui affiche le résultat des tris ;
- ▷ un `LineChart` qui affiche la complexité des tris ;
- ▷ un `Spinner` du nombre de `Threads` autorisé pour traiter les données ;
- ▷ une `ChoiceBox` du type de tri ;
- ▷ une `ChoiceBox` de la quantité maximale d'entiers à trier ;
- ▷ une `ProgressBar` qui affiche la progression des tris ;
- ▷ un `Button Start` pour exécuter les tris ;
- ▷ un `Label` qui affiche le nombre de `Threads` actifs ;
- ▷ un `Label` qui affiche le début, la fin et la durée de la dernière exécution.

Vous êtes libre de modifier cette interface graphique si vous le souhaitez.

Threads actifs

Pour connaître le nombre de `Threads` actifs, vous pouvez utiliser la méthode `Thread.activeCount()`. Comment expliquez vous qu'une application `JavaFX` utilise plusieurs `Threads` ? [a]

a. Un indice est caché dans la documentation https://openjfx.io/javadoc/11/javafx.graphics/javafx/application/Application.html

3 Algorithme de tri

L'implémentation de la vue terminée, commencez par développer les algorithmes de **Tri à bulles** et **Tri fusion**. Ces algorithmes reçoivent un tableau (ou une liste) d'entiers et en trient les éléments.

Vous trouverez une explication de ces tris en consultant par exemple :

- ▷ https://www.baeldung.com/java-bubble-sort
- ▷ https://www.baeldung.com/java-merge-sort

Validez votre implémentation via des tests unitaires.

4 Complexité d'un algorithme

L'instant Wikipedia

En algorithmique, la complexité en temps est une mesure du temps utilisé par un algorithme, exprimé comme fonction de la taille de l'entrée. Le temps compte le nombre d'étapes de calcul avant d'arriver à un résultat.

Concrètement vous allez compter le nombre d'opérations élémentaires effectuées par vos algorithmes afin d'en comparer la performance.

Prenons l'exemple de la recherche du minimum dans un tableau et notons les opérations à compter :

```
1  public int min(int[] datas) {
2      int min = datas[0]; //une opération d'affectation
3      for (int i = 0; i < datas.length; i++) {
4          if (datas[i] < min) { //une opération de comparaison
5              min = datas[i]; //une opération d'affectation
6          }
7      }
8      return min;
9  }
```

Vous constatez qu'on ne compte que les opérations dites élémentaires. Une discussion sur le statut élémentaire des opérations peut être engagée, mais **dans le cadre de cet exercice**, nous considérerons élémentaires les affectations et les comparaisons entre éléments du tableau (de la liste).

La complexité de notre algorithme de recherche de minimum est dite linéaire, c'est à dire que le nombre d'opération grandit linéairement avec la taille du tableau `datas`. On note la complexité linaire par : $\mathcal{O}(n)$.

Nombre d'éléments	Nombre d'opérations
10	21
20	41
30	61

Remarquez que suivant l'implémentation de vos algorithmes le nombre exact d'opérations élémentaires est différent. Par exemple vous pouvez avoir moins d'affectations de variables qu'une autre implémentation. Ce qui importe c'est la valeur asymptotique (l'ordre de grandeur).

Complexité des tris

- **Tri à bulles** a une complexité Quadratique : $\mathcal{O}(n^2)$
- **Tri fusion** a une complexité Linéarithmique : $\mathcal{O}(n \log n)$

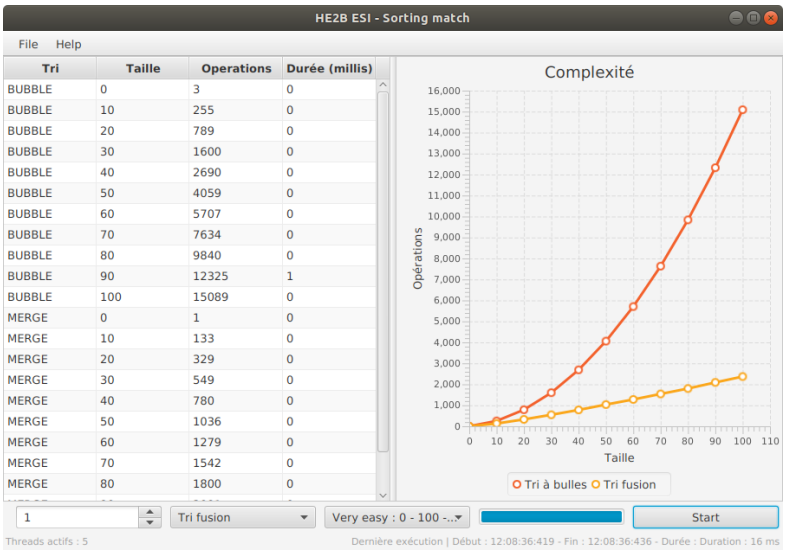
En connaissant la complexité des deux algorithmes de tris, qu'attendez-vous à observer sur le `LineChart` au cours de vos tests ?

5 Série de liste d'entiers

Pour l'instant vous pouvez exécuter un tri à la fois. Ce résultat ne permet pas de voir apparaître sur le `LineChart` la complexité des algorithmes.

Augmentons le nombre de données à présenter sur ce graphique. Pour ce faire, permettez à votre logiciel de lancer 10 tris successivement sur des listes contenant de plus en plus d'entiers.

Par exemple commencez par permettre l'exécution d'un tri sur des listes contenant 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 et 100 éléments. Avec ces résultats vous devriez visualiser la complexité sur le `LineChart`.



Augmentez au fur et à mesure la quantité maximale d'entiers à trier. Prêtez attention à l'utilisation de la mémoire de votre logiciel. L'exception `OutOfMemoryError` pourrait apparaître si vous augmentez la quantité d'entiers à trier au delà d'une certaine limite.

6 Durée d'une exécution

Pour mesurer le temps d'exécution de 10 tris il suffit avant le premier tri de récupérer via la classe `java.time.LocalDateTime` l'heure de début, et à la fin du dernier tri, de récupérer l'heure de fin. Le calcul de la durée entre ces deux moments s'effectuent via la classe `java.time.Duration`.

Mettez l'interface utilisateur à jour avec informations après chaque exécution d'une série de 10 tris.

Vous remarquerez que le temps d'exécution est variable pour un même algorithme. Par exemple, voici quelques résultats produit sur une machine de test :

Type de tri	Nombre de Threads	Nombre d'éléments	Temps (ms)
Tri à bulles	1	10 000	2 005
Tri à bulles	1	10 000	1 486
Tri à bulles	1	10 000	1 575

Les mêmes paramètres ne donnent pas exactement le même temps de calculs.

7 Utilisation d'un pool de threads

Lorsque l'utilisateur appuie sur le bouton `Start`, l'exécution de 10 tris se fait séquentiellement. Le tri d'une liste attend la fin du tri de la liste précédente avant de commencer.

Modifiez cette situation afin d'accélérer le traitement demandé. Permettez d'avoir en permanence deux `Threads` disponibles pour exécuter les tris.

> L'affichage du nombre de `Threads` actifs va permettre de garder un œil au nombre de `Threads` actifs durant l'exécution. Prenez garde que ces `Threads` s'ajoutent aux `Threads JavaFX` déjà actifs.

Lorsque l'utilisateur appuie sur le bouton `Start`, le tri des deux premières listes s'exécute en parallèle. Chaque tri est réalisé dans un `Thread` séparé. Quand un des deux tris à terminé, le troisième tri est exécuté dans le `Thread` libéré.

On continue cette logique jusqu'à ce qu'il ne reste plus rien à trier. On applique le principe du **thread pool** [1] [2].

Message « *Not on FX application thread* »

Durant votre implémentation vous risquez de rencontrer l'exception `java.lang.IllegalStateException: Not on FX application thread`. Vous pouvez résoudre cet incident via l'utilisation de la méthode `Platform.runLater` [a]. Comment expliquez-vous l'origine de cette exception ?

a. https://openjfx.io/javadoc/11/javafx.graphics/javafx/application/Platform.html#runLater(java.lang.Runnable)

N'oubliez pas que le partage des ressources entre vos `Threads` doit être réfléchi. Par exemple si l'un de vos deux `Threads` demande la prochaine liste à trier, le second `Thread` doit attendre avant d'effectuer la même demande [3]. L'utilisation du mot clé `synchronized` permet de résoudre cette situation.

Observateur-Observé

Un Thread peut signaler qu'il a terminé son travail en notifiant d'autres instances qui l'observent.

Une fois cette mise à jour apportée, vous pouvez la généraliser pour que le nombre de `Threads` disponibles soit choisi par l'utilisateur.

Quel est l'impact de ce changement sur le temps d'exécution des tris ?

1. https://en.wikipedia.org/wiki/Thread_pool
2. Il est possible d'implémenter « à la main » la *pool* de *threads* ou bien d'utiliser l'API Java. Pour ce faire, les classes `java.util.concurrent.ExecutorService` et `java.util.concurrent.Executors` sont redoutables d'efficacité.
3. Que se passe-t-il si le second `Thread` demande au même instant la prochaine liste à trier ?