Série III

BONNAZ Aymeric, MOUILLET Claude

13 mars 2021

Cette série d'exercices appartient au corpus testant vos connaissances et votre compréhension des concepts étudiés lors de l'initiation **C# INTM Strasbourg**.
Les réponses écrites doivent être données sous leurs questions.

Nom apprenant : _____

Date de réalisation : _____

Exercice I — Conseil de classe

Dans le cadre d'un conseil de classe, un professeur dispose de la moyenne des élèves de sa classe pour un certain nombre de matières. L'objectif de cet exercice est de calculer la moyenne pour toutes les matières et les écrire dans un fichier de sortie.

Fichier	Champ 1	Champ 2	Champ 3
Entrée	Nom	Matière	Note
Sortie	Matière	Moyenne	

Quelques indications pour la résolution du problème :

- Les deux fichiers sont au format csv et les différents champs sont séparés par un point virgule.
- Les notes sont des nombres réels avec un chiffre après la virgule.

1. (4 points) Implémenter la méthode : **SchoolMeans**(string input, string output) : void

Exemple :

Nom	Matière	Note
Marc	Histoire	12.0
Paul	Histoire	15.5
Yvonne	Maths	6.0
David	Histoire	2.5
Paul	Maths	15.5

Table 1 – Liste des notes par nom et matière.

Matière	Moyenne
Histoire	10.0
Maths	10.8

Table 2 – Liste des moyennes par matière.

Exercice II — Performances des tris

Lors de la résolution d'un problème donné, plusieurs solutions peuvent se présenter et permettre de le résoudre efficacement. L'appui de solides critères départage ces solutions. Le temps d'exécution est un de ceux-ci. L'objectif de cet exercice est de mettre en place un ensemble de fonctions mesurant le temps d'exécution de deux algorithmes apportant une solution au problème du tri d'un tableau de N entiers et de les comparer.

Tris étudiés :

Les algorithmes étudiés sont le tri par insertion et le tri rapide.

Le tri par insertion consiste à placer un élément (appelé **pivot**) dans une liste déjà triée. Après insertion, la liste déjà triée reste triée et contient un élément de plus. La stratégie consiste à l'appliquer du premier au dernier élément du tableau. La liste déjà triée contiendra l'ensemble des éléments du tableau. La figure (1) illustre l'un de ces pas.

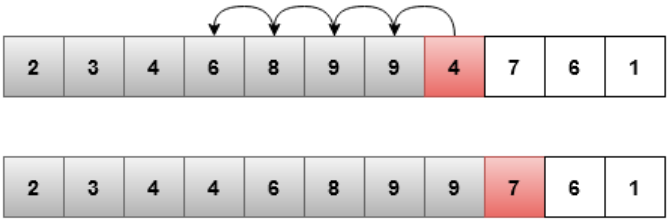

FIGURE 1 – Tri par insertion appliqué au tableau A = [6, 4, 8, 2, 9, 3, 9, 4, 7, 6, 1], les sept premiers éléments ont déjà été triés, le pivot est le huitième élément.

Le tri rapide consiste à ordonner relativement le tableau à la valeur d'un pivot. Après application d'un pas, le pivot est placé tel que tous les éléments à sa gauche lui sont inférieurs ou égaux et à droite tous les éléments lui sont supérieurs. On applique la même procédure dans les deux sous-tableaux formés de manière récursive. La figure (2) illustre l'un de ces pas.

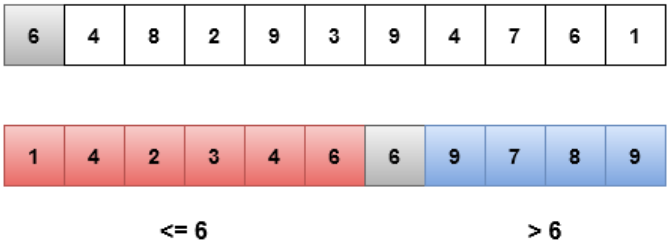

FIGURE 2 – Tri rapide appliqué au tableau A = [6, 4, 8, 2, 9, 3, 9, 4, 7, 6, 1], le pivot est **6**.

Les algorithmes sont implémentés par les fonctions suivantes :

InsertionSort(int[] array) : void ; **QuickSort**(int[] array) : void

Mesure du temps d'exécution :

Premièrement, il faut deux fonctions mesurant le temps d'exécution en millisecondes de chacun des deux algorithmes pour un tableau d'entrée donné.

1. (1 point) Implémenter les deux méthodes suivantes :
   UseInsertionSort(int[] array) : long ; **UseQuickSort**(int[] array) : long

Génération des données d'entrée :

Afin de tester efficacement les deux algorithmes, il est nécessaire de mesurer leur temps d'exécution un grand nombre de fois. Pour produire les données d'entrée, une méthode génératrice de tableaux est indispensable.

Quelques précisions sur ce générateur de tableaux :

- Le générateur doit produire une paire de tableaux identiques d'un **taille** donnée.
- Les entiers composant ces tableaux sont compris entre -1000 et 1000.
- **Random** permet de générer aléatoirement des entiers dans des bornes de valeurs données.

2. (1 point) Pour quelles raisons est-il important de réaliser plusieurs mesures du temps d'exécution différentes pour une même taille de tableau pour obtenir des données viables ?
   Justifier votre choix.

3. (2 points) Implémenter la méthode suivante : **ArraysGenerator**(int size) : List<int[]>

Mesures des performances :

Les performances des deux algorithmes de tris sont caractérisées par la moyenne du temps d'exécution $\bar{x}$ et de l'écart-type à cette moyenne σ dont les définitions sont données dans l'équation (1). **M** représente le nombre d'expériences réalisées pour obtenir ces deux grandeurs et correspond au paramètre **count** dans les fonctions à implémenter.

La structure **SortData** est fournie pour permettre de rendre compte des moyennes et écart-types des deux algorithmes.

Éléments de **SortData**	**Explications**
InsertionMean	Moyenne pour le tri par insertion.
InsertionStd	Écart-type pour le tri par insertion.
QuickMean	Moyenne pour le tri rapide.
QuickStd	Écart-type pour le tri rapide.

$$\bar{x} = \frac{1}{M}\sum_{i=1}^{M} x_i \qquad \sigma = \sqrt{\left(\frac{1}{M}\sum_{i=1}^{M} x_i^2\right) - \bar{x}^2} \tag{1}$$

4. Mesures des performances :
   (a) (2 points) Test de **count** entrées pour une taille donnée :
   PerformanceTest(int size, int count) : SortDatas
   (b) (2 points) Test de **count** entrées pour une liste de tailles données :
   PerformancesTest(List<int> sizes, int count) : List<SortDatas>

Affichage des performances :

Maintenant que les méthodes de mesures sont prêtes, la question de l'affichage de ces données se pose. On définit la méthode **DisplayPerformances**(List<int> sizes, int count) : void.
Quelques précisions pour le fonctionnement de cette fonction :

- Utilisation de **PerformancesTest** pour obtenir les grandeurs de mesures.
- L'en-tête est "n ;MeanInsertion ;StdInsertion ;MeanQuick ;StdQuick".
- Affichage de chacun des résultats pour une taille sur une ligne.

5. (2 points) Appliquer la méthode **DisplayPerformances** pour :
   [**sizes** = {2000, 5000, 10000, 20000, 50000, 100000} et **count** = 50].

```
n;MeanInsertion;StdInsertion;MeanQuick;StdQuick
1000;0;0;0;0
2000;4;1;0;0
5000;30;5;1;0
10000;101;12;2;0
20000;439;72;5;1
50000;2644;119;12;5
100000;10447;280;25;6
```

FIGURE 3 – Mesures de performance du tri par insertion et tri rapide pour des tableaux de tailles N = {1000, 2000, 5000, 10000, 20000, 50000, 100000}.

6. (1 point (bonus)) Déterminer la dépendance réelle en N des deux algorithmes.