SortSimulation

Documentation

– Français –

Peter Folta

Version 1.2.1

Site Web : http ://www.peterfolta.de/software/sortsimulation

Table des matières

1 Introduction

SortSimulation est un programme java que visualise des méthodes de tri. D'un coté on peut mieux comprendre mieux le mode de fonctionnement des algorithme de tri et de l'autre le programme explique les différence en ce qui se concerne les périodes d'action sans comparer des chiffres.

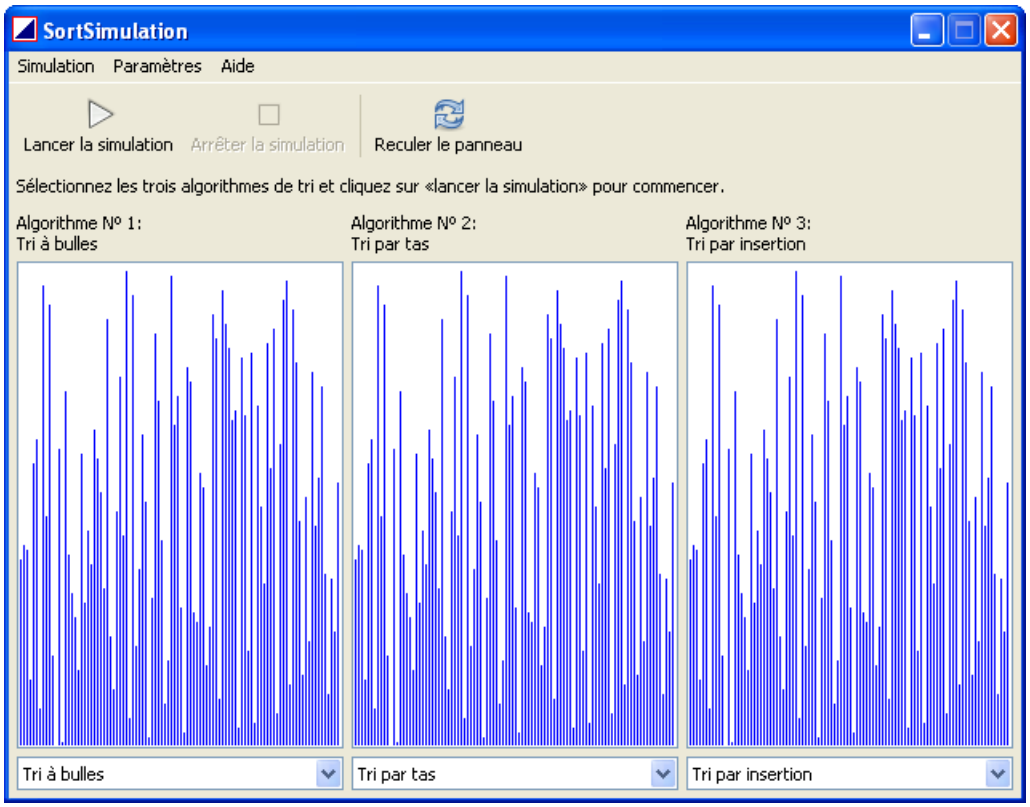

Fig. 1: *La fenêtre principale de SortSimulation*

Ce document consiste d'une introduction courte concernant la commande et une présentation des méthodes de tri soutenu, le mise en œuvre de Java inclus.

2 Commande

La commande de SortSimulation est simple : Après le lancement du programme les panneau de tri sont de déjà remplies par hasard. Dans touts les panneau il y a la même situation de base. En outre trois méthodes de tri sont déjà ajustés, on peut voir cela á cause de liste déroulante au dessous des panneaux.

Alors pour comparer les trois algorithmes de tri, il faut choisir les méthodes dans les listes déroulantes. La simulation lance dés que vous avez cliqué sur le bouton de commande

Lancer la simulation au barre d'outils, après vous avez choisi l'entrée correspondant dans le menu ou pressé la touche d'entrée. Pour arrêter une simulation courante il faut presser la touche Échappement ou on peut clique sur le bouton **Arrêter la simulation**. Pour obtenir après une simulation arrêtée ou finie des panneaux non assorti, cliquez sur le bouton **Reculer le panneau** ou pressez `Ctrl+N`.

SortSimulation vous offre plusieurs paramètres pour adapter la simulation. Dans le paragraphe suivant les paramètres sont expliqués détaillés.

2.1 Paramètres

Plusieurs paramètres sont á votre disposition chez SortSimulation pour ajuster la simulation de tri comme vous voulez. Vous trouvez les possibilités d'ajuster dans le menu **Paramètres** :

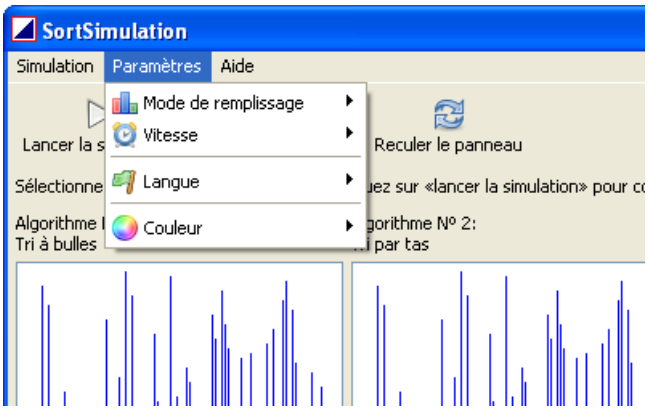

Fig. 2: *Menu « Paramètres »*

L'option **Mode de remplissage** vous offre la possibilité de choisir la sorte de remplissage quand l'action **Reculer le panneau** s'est déroule. Il y a deux modes pour cet action : par hasard (présélectionné ; `Ctrl+R`) et l'inverse (`Ctrl+I`). Dans le mode **Par hasard** les éléments sont arrangés par hasard á chaque remplissage du panneau de sorte qu'on puisse essayer réalistement des méthodes de tri avec des dates non sortie.

Dans le mode **Inverse** au contrarie les panneau sont déjà rempliés mais à l'envers (décroissant au lieu de ascendant) pour savoir l'efficience les méthodes de tri concernant les séquences qui sont sorties à l'envers.

Le sous-menu **Vitesse** vous donnez la possibilité de choisir la vitesse. Vous avez le choix entre cinq degrés. Pendant que les algorithmes comme tri à bulles sont très lente et ainsi une grande vitesse s'offre, il est profitable de regarder par exemple tri rapide lentement pour pouvoir comprendre le fonction. La vitesse se laisse aussi controler par la combinaison de ouches `Ctrl+Shift+(1-5)`.

SortSimulation n'écrit pas des données sur votre ordinateur ou les modifies, à cause de cela le programme commence chaque fois avec une interface anglais. Vous pouvez changer le langue dans le menu (anglais : **Settings** > **Language**).

Dans le sous-menu **Couleur** vous pouvez ajuster le couleur des barres. Le standard est bleu mais vous avez le choix entre huit couleur.

3 Méthodes de tri

3.1 Tri à bulles

Tri à bulles est un algorithme de tri simple que sort les éléments en comparant **par étapes**. Tri à bulles n'est pas efficace ainsi on utilise l'algorithme pour démontrer un mauvais méthode de tri.

3.1.1 L'idée de tri à bulles

Tri à bulles compare deux élément voisin et les change si l'ordre n'est pas correct. Cet action se repasse jusqu'à toutes les éléments sont en bon ordre c'est-à-dire qu'il sont sortis.

3.1.2 Mise en œuvre de Java

```
1  public class Bubblesort {
2      public void sort(int[] a) {
3          int tmp;
4
5          for(int i = a.length-1; i >= 0; i--) {
6              for(int j = 0; j <= i-1; j++) {
7                  if(a[j] > a[j+1]) {
8                      tmp = a[j];
9                      a[j] = a[j+1];
10                     a[j+1] = tmp;
11                 }
12             }
13         }
14     }
15 }
```

Listing 1: *Mise en oeuvre de tri à bulles*

3.2 Tri par tas

Tri par tas est un méthode de tri vite que c'était inventé par **Robert W Floyd** et **J. W. J. Williams** en 1964 sur la base de **tri par sélection**.

3.2.1 L'idée de tri par tas

Tri par tas utilise pour le triage une structure des données particulière ; le **tas** qui est fondé sur un arbre qui est presque totalement binaire. Un arbre binaire est (presque) complet quand toutes les plans sauf éventuellement la dernière sont complètes.

Quand on a une séquence tas, le plus grand élément de la **racine** d'arbre peut être échantillonné et distribué. Pour avoir accès au élément supérieur, il faut **ordre le tas à nouveau**.

3.2.2 Mise en œuvre de Java

```
1  public class Heapsort {
2      private void sift(int[] a, int l, int r) {
3          int i;
4          int j;
5          int x;
6
7          i = l;
8          x = a[l];
9          j = 2 * i + 1;
10
11         if((j < r) && (a[j+1] > a[j])) j++;
12
13         while((j <= r) && (a[j] > x)) {
14             a[i] = a[j];
15             i = j;
16             j = 2 * j + 1;
17
18             if((j < r) && (a[j+1] > a[j])) j++;
19         }
20
21         a[i] = x;
22     }
23
24     public void sort(int[] a) {
```

```
25          int l;
26          int r;
27          int tmp;
28
29          for(l = (a.length - 2) / 2; l >= 0; l--) {
30              sift(a, l, a.length-1);
31          }
32
33          for(r = a.length - 1; r > 0; r--) {
34              tmp = a[0];
35              a[0] = a[r];
36              a[r] = tmp;
37              sift(a, 0, r-1);
38          }
39      }
40  }
```

Listing 2: *Mise en oeuvre de tri par tas*

3.3 Tri par insertion

Tri par insertion est une méthode de tri simple qui n'est pas si efficace comme des algorithmes plus complexes, mais on peut la mettre plus **facilement en œuvre** et elle a besoin d'une distance électrique courte quand on a peu des données ou quand ils sont déjà tris.

3.3.1 L'idée de tri par insertion

Tri par insertion échantillonne une élement d'une ensemble non tri et l'ajoute à la vraie position dans la séquence de distribution. Si la séquence est encore vide, l'élément est collée à la première position.

Tri par insertion n'est pas efficace car cette méthode de tri doit délocaliser les éléments souvent par des distances vaste.

3.3.2 Mise en œuvre de Java

```
1  public class Insertionsort {
2      public void sort(int[] a) {
3          int tmp;
4          int j;
5
```

```
6           for(int i = 1; i < a.length; i++) {
7               tmp = a[i];
8               j = i;
9
10              while(j > 0 && a[j-1] > tmp) {
11                  a[j] = a[j-1];
12                  j--;
13              }
14
15              a[j] = tmp;
16          }
17      }
18  }
```

Listing 3: *Mise en oeuvre de tri par insertion*

3.4 Tri fusion

Tri fusion est un algorithme de tri récursif et stable qui base sur le principe **Divide-and-Conquer**. Tri fusion était présenté en 1945 par John von Neumann.

3.4.1 L'idée de tri fusion

Tri fusion décompose la séquence en plusieures petites séquences qui sont triés isolements. Puis les petites séquences sont assemblées selon le principe de la fermeture éclair jusqu'au moment quand toutes les éléments sont triés pour avoir une grande séquence.

3.4.2 Mise en œuvre de Java

```
1  public class Mergesort {
2      private int[] a;
3      private int[] b;
4      private int n;
5
6      public void sort(int[] a) {
7          this.a = a;
8          n = a.length;
9          b = new int[n];
10         mergesort(0, n-1);
11     }
12
```

```
13      private void mergesort(int lo, int hi) {
14          if(lo < hi) {
15              int m = (lo + hi) / 2;
16              mergesort(lo, m);
17              mergesort(m+1, hi);
18              merge(lo, m, hi);
19          }
20      }
21
22      private void merge(int lo, int m, int hi) {
23          int i = lo;
24          int j = hi;
25          int k = lo;
26
27          while(i <= m) b[k++] = a[i++];
28          while(j > m) b[k++] = a[j--];
29
30          i = lo;
31          j = hi;
32          k = lo;
33
34          while(i <= j) {
35              if(b[i] <= b[j]) a[k++] = b[i++];
36              else a[k++] = b[j--];
37          }
38      }
39  }
```

Listing 4: *Mise en oeuvre de tri fusion*

3.5 Tri rapide

Tri rapide est une des méthodes de tri **les plus rapides** qui base sur le principe **Divide-and-Conquer**. En 1960 la version initial de l'algorithme récursif de tri rapide était développé par **C. Antony R. Hoare**.

3.5.1 L'idée de tri rapide

La séquence qu'on veut trier est d'abord fragmenté en deux parts afin que les éléments dans la première fragment sont plus petites ou égaux que les élément dans la deuxième fragment (**divide**). Puis les deux fragment sont triés isolements dans une manière récursive

en utilisant la même méthode (**conquer**). Dans le dernier étape les fragments assemblées donnent la séquences triée (**combine**).

La partition est réalisé à l'aide d'un **élément pivot** qui est choisi pendant la première étape de la séquence. Toutes les éléments de la séquence qui sont **plus petites** que l'élément pivot, viennent au première tronçon. Toutes les éléments qui sont **plus grandes** que l'élément pivot au deuxième tronçon. En ce qui se concerne les éléments qui sont autant grandes que l'élément pivot il ne joue aucun rôle d'où ils viennet.

3.5.2 Mise en œuvre de Java

```
1  public class Quicksort {
2      private void quicksort(int[] a, int bottom, int top) {
3          int tmp;
4          int i = bottom;
5          int j = top;
6          int middle = (bottom + top) / 2;
7          int x = a[middle];
8
9          do {
10             while(a[i] < x) i++;
11             while(a[j] > x) j--;
12
13             if(i <= j) {
14                 tmp = a[i];
15                 a[i] = a[j];
16                 a[j] = tmp;
17                 i++;
18                 j--;
19             }
20         } while(i <= j);
21
22         if(bottom < j) quicksort(a, bottom, j);
23         if(i < top) quicksort(a, i, top);
24     }
25
26     public void sort(int[] a) {
27         quicksort(a, 0, a.length-1);
28     }
```

```
29 }
```

Listing 5: *Mise en oeuvre de tri rapide*

3.6 Tri par sélection

Tri par sélection est un algorithme de tri naïve qui travaille sur place.

3.6.1 L'idée de tri par sélection

Tri par sélection fragmente la séquence de tri en un part **trié** et un part **non-trié**. Au début le part trié est vide. Tri par sélection cherche le plus petit élément dans le part non-trié et le remplace avec le première élément. Après cela la séquence est triée jusqu'à cette position. Le part trié a grandi par un élément, le part non-trié a diminué par un élément. La méthode est répété jusqu'au moment quand la séquences est triée complètement.

3.6.2 Mise en œuvre de Java

```
1  public class Selectionsort {
2      public void sort(int[] a) {
3          int tmp;
4
5          for(int i = 0; i < a.length; i++) {
6              for(int j = i+1; j < a.length; j++) {
7                  if(a[j] < a[i]) {
8                      tmp = a[i];
9                      a[i] = a[j];
10                     a[j] = tmp;
11                 }
12             }
13         }
14     }
15 }
```

Listing 6: *Mise en oeuvre de tri par sélection*

3.7 Tri de Shell

Tri de Shell est une méthodes de tri qui est basé sur **tri par insertion**. Tri de Shell était développé par **Donald L. Shell** en 1959.

3.7.1 L'idée de tri de Shell

Tri de Shell compense le désavantage d'tri de insertion, les éléments ne doivent plus être déplacés par des distances longues. Pour réaliser cela, tri de Shell produit une **matrice k-délitescente**, les colonnes du matrice sont triés isolements. Après cela la séquence est déjà triée bourrue. Cette démarche est répétée, le nombre des colonnes se réduit pendant chaque réalisation jusqu'au moment quand il y a seulement une colonne.

3.7.2 Mise en œuvre de Java

```
1  public class Shellsort {
2      public void sort(int[] a) {
3          int[] cols = {
4              1391376, 463792, 198768, 86961, 33936, 13776,
5              4592, 1968, 861, 336, 112, 48, 21, 7, 3, 1
6          };
7
8          for(int i = 0; i < cols.length; i++) {
9              int h = cols[i];
10
11             for(int j = h; j < a.length; j++) {
12                 int k = j;
13                 int tmp = a[k];
14
15                 while(k >= h && a[k-h] > tmp) {
16                     a[k] = a[k-h];
17                     k = k - h;
18                 }
19
20                 a[k] = tmp;
21             }
22         }
23     }
24 }
```

Listing 7: *Mise en oeuvre de tri de Shell*

4 Coopérateurs

Je remerie les personnes suivantes qui m'a soutenu allantement de développer SortSimulation. Des nouveaux coopérateurs (traducteurs, designer, auteurs des documentations,

etc.) sont cherchés continuellement. Si vous avez envie de soutenir les projets de Peter Folta, contactez -le.

4.1 Traducteurs

- Folta, Lucia Sonja – Russe
- Folta, Peter – Anglais, Allemand
- Müllner, Jan Sebastian – Français, Espagnol

5 Informations de contact

Peter Folta
Humboldtstrasse 9
34497 Korbach
Allemagne

E-Mail : mail@peterfolta.de
Site Web : http ://www.peterfolta.de/

Références

[1] Lang, Prof. Dr. Hans Werner : **Algorithmen in Java**. 2. Édition 2006. Munich : Oldenbourg Wissenschaftsverlag GmbH 2006. ISBN 978-3-486-57938-3, p. 5–52

Table des figures

Listings