SortSimulation

Документация

– Русский –

Peter Folta

Версия 1.2.1

Веб: http://www.peterfolta.de/software/sortsimulation

Содержание

1 Введение

SortSimulation является программой явы, которая изображает визуально разные способы сортировки. Это даст возможность с одной стороны лучше понять метод функции разных алгоритмов сортировки а на другой стороне поясняет разницы рабочего времени без сравнения лишь сухих чисел.

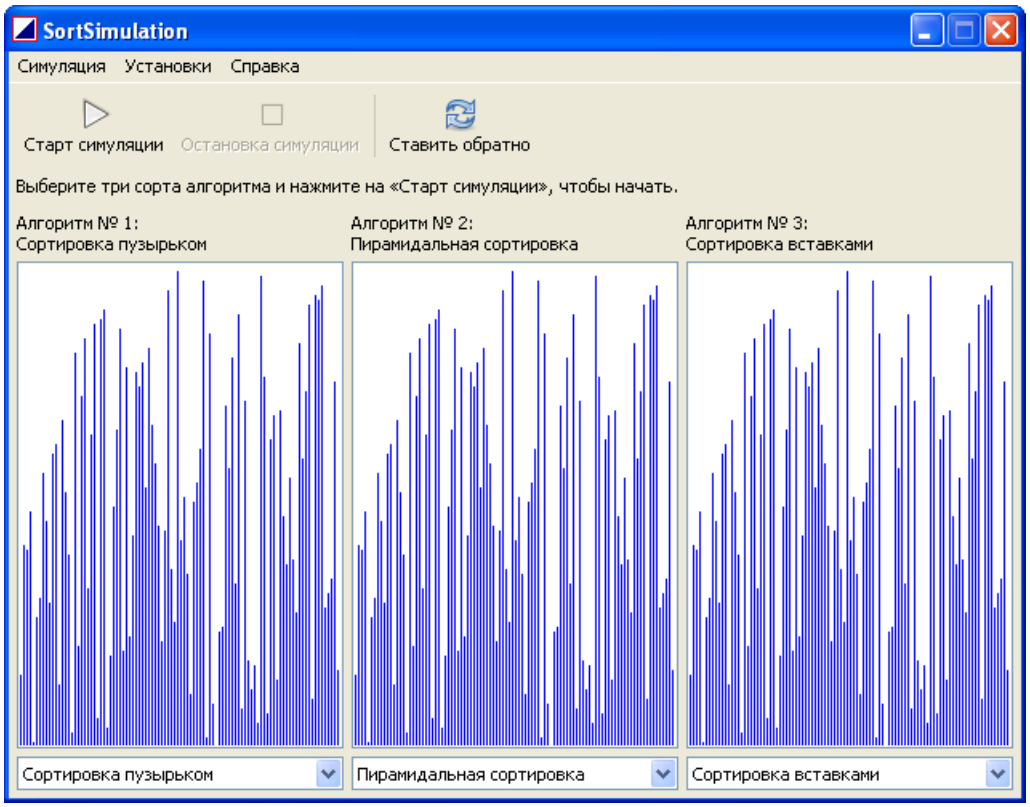

Рис. № 1: *Центральное окно SortSimulation*

Эта документация содержает краткое введение и обслуживание SortSimulation и изображает затем поддерживающие способы сортировки включая имплементацию явы.

2 Обслуживание

Обслуживание SortSimulation возможно простое: Сразу после старта программы уже поля сортировки случайно наполнены. При этом в каждом из этих трёх полей та же самая исходная ситуация. После старта программы тоже установлены три разных способа сортировки; это можно видеть на ящиках выбора под полями сортировки.

Чтобы сравнить друг с другом три алгоритма сортировки, выберите из ящиков выбора соответствующие способы. Симуляция начинается, когда щелкнете по **Старт**

симуляции на панеле инструментов, выберете соответствующее внесение в меню **Симуляция** или щелкнете по ввод клавиша. Если хотите прорвать симуляцию, достаточно щелкнуть по Esc-клавиша или на кнопке **Остановка симуляции**. Чтобы получить снова, после завершенной или прорванной симуляции, сдучайно наполнены поля, щелкните по **Ставить обратно** или нажмите `Ctrl+N`.

SortSimulation обнаруживает несколько установок, с которыми можете приноровить симуляцию. На этих установках остановимся подробно в следующем отрывке.

2.1 Установки

Для вас находятся в распоряжении разные установки, с которыми можете приноровить сортирующие симуляции в соответствии с вашим желанием. Эти возможности установок вы найдёте в меню **Установки**:

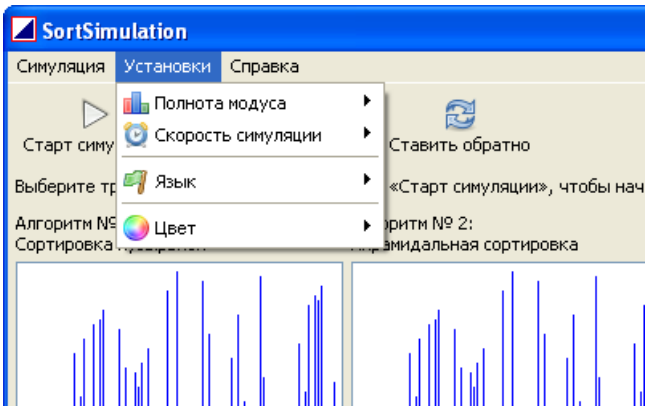

Рис. № 2: *Меню «Установки»*

При помощи **Полнота модуса** можете определить, каким способом должны быть наполнены поля сортировки, когда акция **Ставить обратно** исполнена. При этом есть два модуса в распоряжении: случайный (установлен; `Ctrl+R`) и инверс (`Ctrl+I`). В модусе **Случайный** будут элементы при каждом новом наполнении полей случайно установлены. Так можно производить реалистическим способом опыт разных сортировок с не сортированными данными.

В модусе **Инверс** поля уже сортированно наполнены – разумеется наоборот (понижаемо а не повышаемо). Этой установкой можно видеть, как эффектно сортируют сортировки наполнены наоборот.

Низшее меню **Скорость симуляции** даст вам возможность приноровить скорость симуляции. У вас есть выбор из пятеро степеней. Так как например у алгоритма сортировка пузырьком, которая очень медленная, напрашивается высокая степень скорости симуляции, так стоит наблюдать например быструю сортировку на медленной степе-

ни скорости, чтобы лучше понять образ действия этого способа. Отдельные степени скорости можно получить на клавиши при помощи `Ctrl+Shift+(1-5)`.

SortSimulation не пишет или не изменяет данные на вашем компьютере, поэтому программа начинается каждый раз с английским интерфейсом пользователя. Вы можете изменить язык в меню **Язык** (англ.: **Settings** > **Language**).

В последнем низшем меню **Цвет** можно установить цвет балок. По стандарту установлен синий цвет. На выбор есть восемь различных цветов.

3 Алгоритм сортировки

3.1 Сортировка пузырьком

Сортировка пузырьком является простым алгоритмом, которая сортирует элементы через **постепенное сравнение**. Потому что сортировка пузырьком не эффектная, употребляется этот алгоритм часто как демонстрация плохого метода сортировки.

3.1.1 Идея сортировки пузырьком

Сортировка пузырьком сравняет два соседних элемента и обменяет их, если они находятся в неправильной последовательности. Этот процесс повторяется так долго, пока все элементы не стоят на правильном месте и так они сортируются.

3.1.2 Имплементация явы

```
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 }
```

Листинг № 1: *Имплементация сортировки пузырьком*

3.2 Пирамидальная сортировка

Пирамидальная сортировка быстрый способ сортировки, который был развит в 1964 году **Робертом В Флоудом** (англ. Robert W Floyd) и **Й. В. Й. Вильямсом** (англ. J. W. J. Williams). Пирамидальная сортировка улучшение алгоритма **сортировки выбором**.

3.2.1 Идея пирамидальной сортировки

Пирамидальная сортировка употребляет для сортировки специальную структуру данных: **пирамиду**. Эта структура данных базирует на (почти) полном **бинарном дереве**. Бинарное дерево является (почти) полным, когда все области, быть может без последней, полные.

Когда сортированная очерёдность есть пирамидой, тогда можно взять и выдать самый большой элемент из **корня дерева**. Чтобы прийти к следующему большому элементу, должна быть пирамида **заново установлена**.

3.2.2 Имплементация явы

```
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  }
```

Листинг № 2: *Имплементация пирамидальной сортировки*

3.3 Сортировка вставками

Сортировка вставками является простым методом сортировки. Она не так эффектна как другие комплексные алгоритмы, но её можно **просто имплементировать** и она требует только короткое рабочее время при небольших или уже заранее сортированных данных.

3.3.1 Идея сортировки вставками

При сортировке вставками берётся один элемент из несортированной массы и вставляется на правильное место в расход последствия. Если последствие ещё пустое, элемент будет вставлен на первую позицию.

Сортировка вставками не эффектна, потому что при ней надо часто передвигать элементы через дальнее расстояние.

3.3.2 Имплементация явы

```
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  }
```

Листинг № 3: *Имплементация сортировки вставками*

3.4 Сортировка слиянием

Сортировка слиянием является рекурсным и стабильным сортировным алгоритмом, который базирует как и **быстрая сортировка** на принципе **разделяй и властвуй**. Сортировка слиянием была представлена в 1945 году **Джоном фон Нейманом** (англ. John von Neumann).

3.4.1 Идея сортировки слиянием

Сортировка слиянием разбирает последствие, которое должно быть сортировано, на несколько малых последствий, каждая из них будет сортирована для себя. Затем будут эти сортированные малые последствия собраны методом молнии в большое последствие, пока все элементы не сортированные.

3.4.2 Имплементация явы

```
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  }
```

Листинг № 4: *Имплементация сортировки слиянием*

3.5 Быстрая сортировка

Быстрая сортировка **самый быстрый способ сортировки**, который базирует на принципе **разделяй и властвуй**. Этот рекурсный алгоритм быстрой сортировки развил в своей первоначальной формулировке **Ч. Энтони Р. Хоар** (англ. C. A. R. Hoare) в 1960 году.

3.5.1 Идея быстрой сортировки

Последствие, которое должно быть сортированным, с начала разделяется так на две части, что все элементы в первой части менее или равны всем элементам второй части (**разделяй**). Затем сортируются рекурсно эти две части независимо друг от друга этим же самым способом (**властвуй**). В последнем шагу составляют эти две части сортированное последствие (**соединяй**).

Это разделение реализуется при помощи одного **опорного элемента**, который выбирается в первом шагу из последствия. Все элементы последствия, которые **меньше** опорного эдемента, придут в первю часть. Все елементы, которюе **больше** опорного елемента придут во вторую часть. У элементов, которые одинаковы с опорным элементом, всё ровно, до которой части они придут.

3.5.2 Имплементация явы

```
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  }
```

Листинг № 5: *Имплементация быстрой сортировки*

3.6 Сортировка выбором

Сортировка выбором является наивным сортировным алгоритмом, который работает на месте. Эту сортировку можно сравнить с сортировкой вставками.

3.6.1 Идея сортировки выбором

Сортировка выбором разделяет последствие, которое должно быть сортировано, на **сортированную** и **несортированную** части. Сортированная часть в начале пустая. Сортировка выбором ищет самый малый элемент в несортированной части последствия и сменяет его на первый элемент. После этого шага последствие сортировано до этой позиции. Сортированная часть становится так больше на один элемент, несортированная часть на один элемент меньше. Этот метод повторяется, пока целое последствие не сортировано.

3.6.2 Имплементация явы

```
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 }
```

Листинг № 6: *Имплементация сортировки выбором*

3.7 Сортировка Шелла

Сортировка Шелла метод сортировки, который базирует на **сортировке вставками**. Сортировка Шелла была развита в 1959 году **Доналдом Л. Шеллом** (англ. Donald L. Shell).

3.7.1 Идея сортировки Шелла

Сортировка Шелла компенсирует недостаток сортировки вставками, при которой надо элементы передвигать через дальнее растояние. Сортировка Шелла производит к тому одну **к-раскалывающуюся матрицу**, которой столбцы отдельно сортируются. После этих шагов уже последствие грубо сортировано. Этот шаг повторяется, причём количество столбцов уменьшается при каждом выполнении, пока матрица не состоит только из одного отдельного столбца.

3.7.2 Имплементация явы

```
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  }
```

Листинг № 7: *Имплементация сортировки Шелла*

4 Сотрудники

Здесь надо выразить благодарность следующим персонам, которые деятельно оказали помощь развитию SortSimulation. Новые сотрудники (переводчики, чертёжники, составители документаций и так далее) ищутся постоянно – если вы интересуетесь, войдите в контакт с Петром Фолта (нем. Peter Folta).

4.1 Переводчики

- Folta, Lucia Sonja – Русский
- Folta, Peter – Английский, Немецкий
- Müllner, Jan Sebastian – Французский, Испанский

5 Контакт

Peter Folta
Humboldtstrasse 9
34497 Корбах (нем. Korbach)
Германия

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

Список литературы

[1] Lang, Prof. Dr. Hans Werner: **Algorithmen in Java**. 2-е издание 2006 г. Мюнхен: Oldenbourg Wissenschaftsverlag GmbH 2006. ISBN 978-3-486-57938-3, стр. 5–52

Список иллюстраций

Листинги