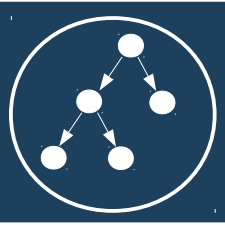


Data Structures and Algorithms

Алгоритмы.
Сортировка перемешиванием

Сведение о алгоритме

Сложность по времени в наихудшем случае $O(n^2)$

Описание сути алгоритма

Сортировка перемешиванием является модификацией алгоритма сортировки пузырьком. Сутью этой модификации является следующие наблюдения.

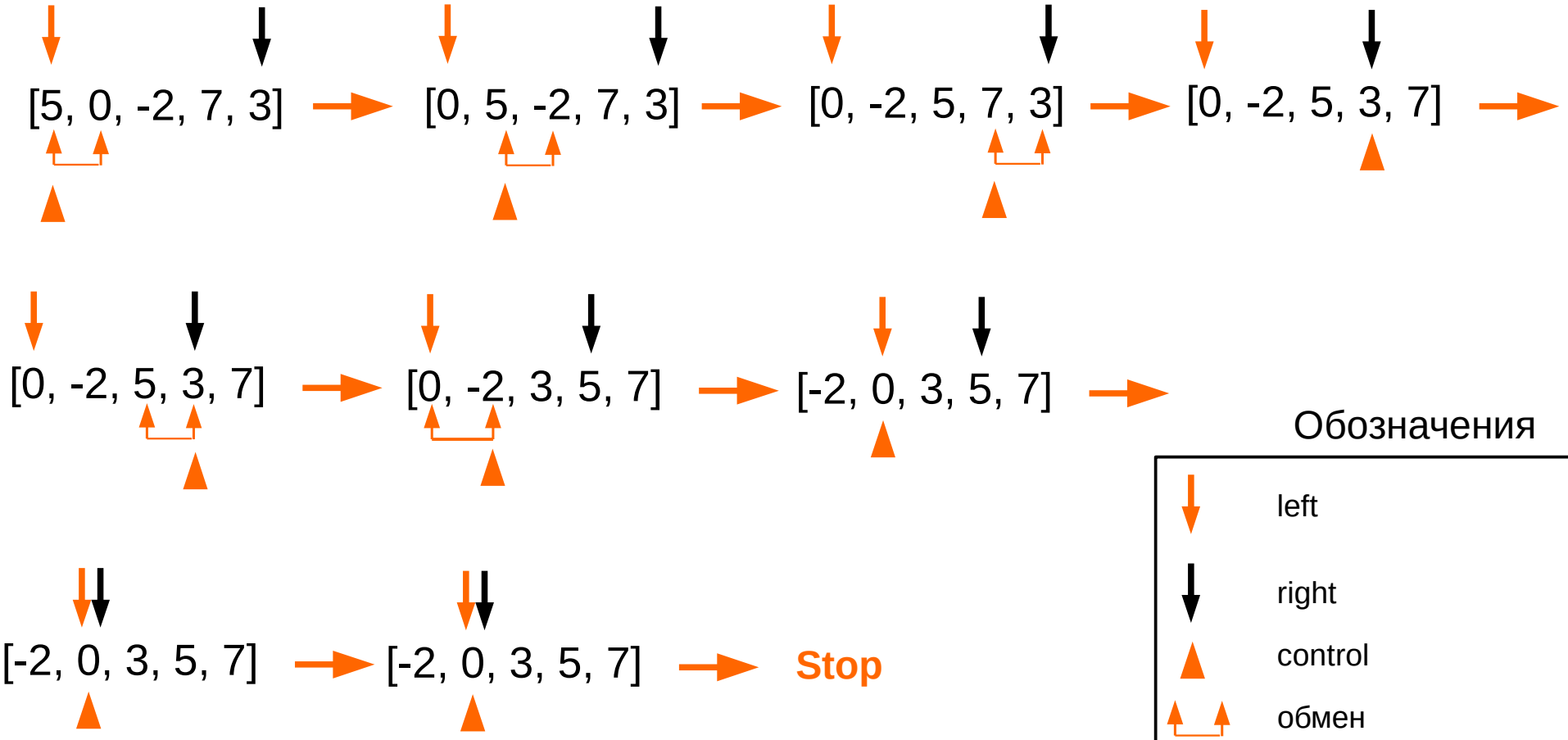
- Первая перестановка в последовательности начинается с определенного индекса и при следующем проходе можно начинать с этого индекса, а не с начала последовательности.
- При проходе в одну сторону элемент сдвигается на одну позицию к соответствующему краю последовательности. И если чередовать проходы с начала последовательности до ее конца с проходами в обратном направлении то можно обеспечить более быстрое «всплывание» элемента к нужному краю последовательности.

Именно реализация этих изменений при работе алгоритма сортировки пузырьком и получила название сортировки перемешиванием.

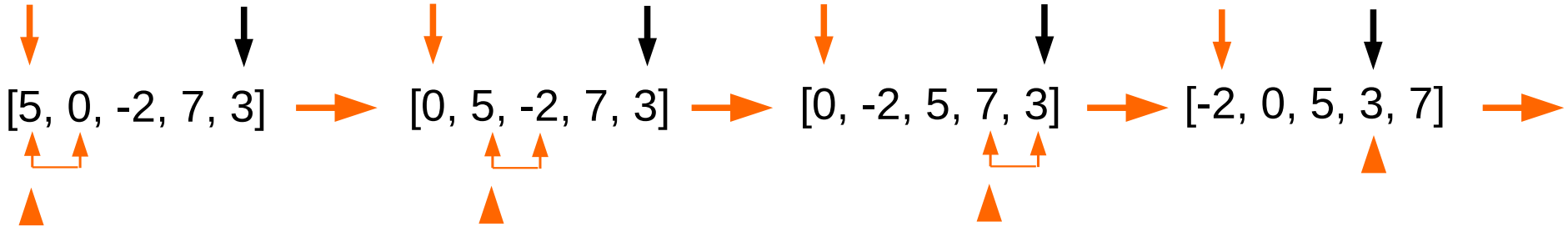
Описание алгоритма

1) Объявляем две дополнительные переменные для хранения границ с которых нужно начинать проход по элементам последовательности (left и right соответственно) и переменную для контроля наличия факта обмена (control). Устанавливаем значение left = 0, right — индексу последнего элемента последовательности, control =right. Перейти к пункту **2**.
2) Начиная от left и до right выполняем проход по элементам последовательности. Если текущий элемент больше следующего элемента то провести их обмен и установить значение control равной индексу текущего элемента. После прохода установить значение right равный control. Перейти к пункту **3**.
3) Начиная от right до left выполняем обратный проход по элементам последовательности. Если текущий элемент меньше предыдущего то выполняем их обмен и устанавливаем значение control равным индексу текущего элемента. После прохода устанавливаем значение left равным control. Перейти в пункту **4**.
4) Если left<right вернуться к пункту **2**. В противном случае **закончить алгоритм**.

Графическая иллюстрация работы алгоритма



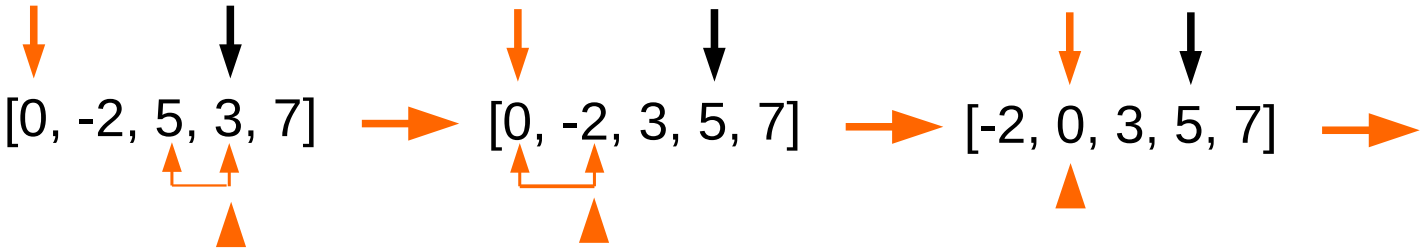
Графическое пояснение алгоритма



В качестве начальных значений устанавливаем left = 0, right = control = 4.

Начинаем проход в прямом направлении от left до right. Если текущий элемент больше следующего то совершаем их обмен и устанавливаем control равный индексу текущего элемента. После того как закончили проход устанавливаем значение right равный control.

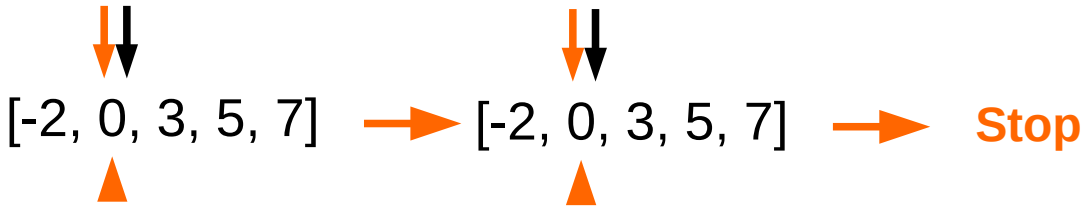
Графическое пояснение алгоритма



Выполняем обратный проход начиная от right до left. Если текущий элемент меньше предыдущего (отчет ведем от начала последовательности) то производим их обмен и устанавливаем значение control равный индексу текущего элемента.

После прохода устанавливаем значение left равным control.

Графическое пояснение алгоритма



Выполняем прямой проход от left до right. Обменов не было следовательно rignt = control = left. Выполняем обратный проход (но так как right = left он нулевой длинны).

Проводим проверку left < right. Это не так. Заканчиваем алгоритм.

Python

Реализация алгоритма на Python

Реализация алгоритма на Python

```
def coctail_sort(sequince):
    left = 0
    right = len(sequince) - 1
    control = right
    while left < right:
        for i in range(left, right):
            if sequince[i] > sequince[i+1]:
                sequince[i], sequince[i+1] = sequince[i+1], sequince[i]
                control = i
        right = control
        for i in range(right, left, -1):
            if sequince[i] < sequince[i-1]:
                sequince[i], sequince[i-1] = sequince[i-1], sequince[i]
                control = i
        left = control
```

Java

Реализация алгоритма на Java

Реализация алгоритма на Java

Вспомогательная функция для обмена двух элементов массива

```
public static void swap(int[] sequince, int i, int j) {
        int temp = sequince[i];
        sequince[i] = sequince[j];
        sequince[j] = temp;
    }
```

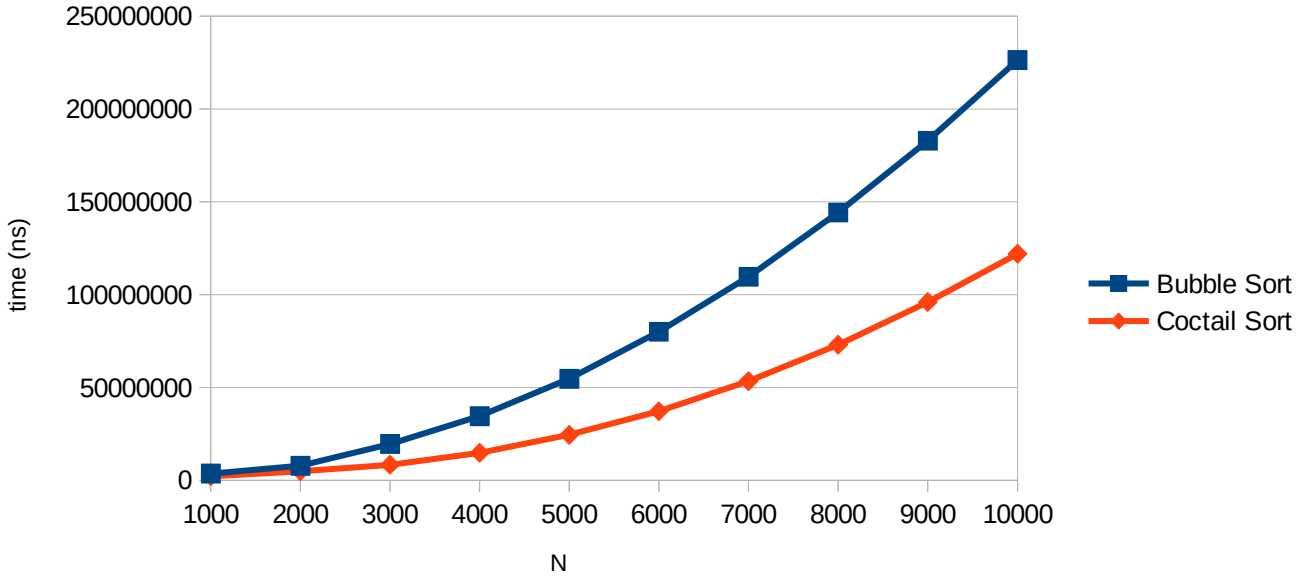
Java

Реализация алгоритма на Java

```
public static void coctailSort(int[] sequince) {
    int left = 0;
    int right = sequince.length - 1;
    int control = right;
    for (; left < right;) {
        for (int i = left; i < right; i++) {
            if (sequince[i] > sequince[i + 1]) {
                swap(sequince, i, i + 1);
                control = i;
            }
        }
        right = control;
        for (int i = right; i > left; i--) {
            if (sequince[i] < sequince[i - 1]) {
                swap(sequince, i, i - 1);
                control = i;
            }
        }
        left = control;
    }
}
```

Вычислительный эксперимент

Для проверки увеличения эффективности данного алгоритма по сравнению с алгоритмом сортировки пузырьком был проведен вычислительный эксперимент. Для массивов разных размеров (заполненных случайными числами) было замерено среднее время сортировки с помощью того и другого алгоритма. На графике приведена зависимость среднего времени сортировки от размера массива.



Как можно видеть из графика алгоритм сортировки перемешиванием обладает более высокой производительностью.

Список литературы

1) Д. Кнут. Искусство программирования. Том 3. «Сортировка и поиск», 2-е изд. ISBN 5-8459-0082-4