ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

Факультет прикладной математики и механики

Кафедра «Прикладная математика»

Курсовая работа

по дисциплине «Алгоритмы и структуры данных»

Тема: «Сортировки обменами»

Выполнил студент группы ИВК-21-1б

Каримов Руслан Альберточич

Проверил: асс. Абакшин Д.С.

Пермь, 2023

Содержание

[Введение 3](#_Оглавление000002)

[Цель и задачи 4](#_Оглавление000015)

[1. Гномья сортировка (Gnome sort) 5](#_Оглавление000004)

[2. Пузырьковая сортировка (Bubble sort) 7](#_Оглавление000005)

[3. Шейкерная сортировка (Shaker sort) 8](#_Оглавление000006)

[4. Сравнение сортировок обменами 9](#_Оглавление000007)

[Выводы 10](#_Оглавление000008)

[Список литературы 11](#_Оглавление000009)

[Приложение.. 12](#_Оглавление000010)

# Введение

Обработка и поиск информации значительно ускоряется, если данные расположены в определенной последовательности. Например, списки учащихся, студентов, сотрудников в алфавитной последовательности, цифры от меньшего значения к большему и наоборот.

Под сортировкой обычно понимают процесс перестановки объектов некоторого множества в определенном порядке. Цель сортировки – облегчить в дальнейшем поиск элементов отсортированного множества. Это очень часто выполняемая, фундаментальная операция.[1]

Каждый элемент из множества имеет определенное количество параметров, не влияющих на сортировку, и один сортировочный параметр, называемый ключом. На массиве ключей задано отношение порядка, то есть линейного упорядочивания, для которого должны быть выполнены два условия: закон трихотомии, при котором ключ одного элемента может быть только больше, меньше или равен ключу другого элемента; и транзитивность, при которой если ключ одного элемента, больше ключа второго элемента, который в свою очередь больше ключа третьего элемента, то ключ первого элемента должен быть больше ключа третьего элемента.[2]

Сортировка может быть по возрастанию, задачей которой является нахождение перестановки элементов таким образом, что ключи располагаются в порядке равенства или возрастания.

Точно также только для равенства и убывания можно обозначить сортировку в обратном порядке.

Существуют различные методы сортировки, различающиеся по степени эффективности, которая определяется количеством сравнений и числом обменов. В работе будут рассмотрены гномья, пузырьковая и шейкерная сортировки.

# Цель и задачи

Цель данной работы: анализ и сравнение алгоритмов сортировки по скорости выполнения.

Задачи данной работы:

* определить понятие алгоритма сортировки данных;
* рассмотреть гномью, пузырьковую и шейкерную сортировки;
* написать программы на java, реализующие гномью, пузырьковую и шейкерную сортировки;
* сравнить гномью, пузырьковую и шейкерную сортировки;
* провести анализ и сделать выводы.

# 

# 1. Гномья сортировка (Gnome sort)

Гномная сортировка была впервые предложена в 2000 году Хамидом Сарбази-Азадом. Он назвал ее «Глупая сортировка, простейший алгоритм сортировки всего с одним циклом…». Действительно, глупым этот метод не назовешь, поскольку в нем используется только один цикл сортировки. Позже, голландский ученый Дик Грун, на страницах одной из своих книг, привел следующую аналогию для гномьей сортировки.[3]

Гномья сортировка основывается на использовании техники, которую используют садовые гномы. Вот так садовник сортирует цветочные горшки. По сути, он наблюдает за двумя соседними цветочными горшками. Если они правильно расположены, то он переходит на один из них, а если нет, то переходит на другой горшок. Граничное условие: если позади нет горшков, он идет вперед, и если нет следующего, то он заканчивает.

Заменим гнома и горшки более формальными объектами, такими как указатель (номера текущих элементов) и элемент массива. Рассмотрим принцип действия алгоритма снова. Сначала указатель ставится на второй элемент массива.

Затем сравниваются два соседних элемента A[i] и A[i-1]. Если элементы стоят на своем месте, то сдвигают указатель на позицию i+1 и продолжают сравнение с нового положения; если же они находятся не на своем месте, то передвигают указатель на другую позицию влево, чтобы продолжить сравнение. Так алгоритм будет продолжать выполняться, пока весь массив не будет упорядочен.

Здесь нужно выделить два момента:

1. процесс упорядочения прекращается тогда, когда не выполнено условие i<n, где n – размер массива.
2. указатель может перемещаться как вперед, так назад, но если он находится над первым элементом списка, то его нужно сместить вправо, чтобы не пришлось сравнивать два элемента, одного из которых не существует.

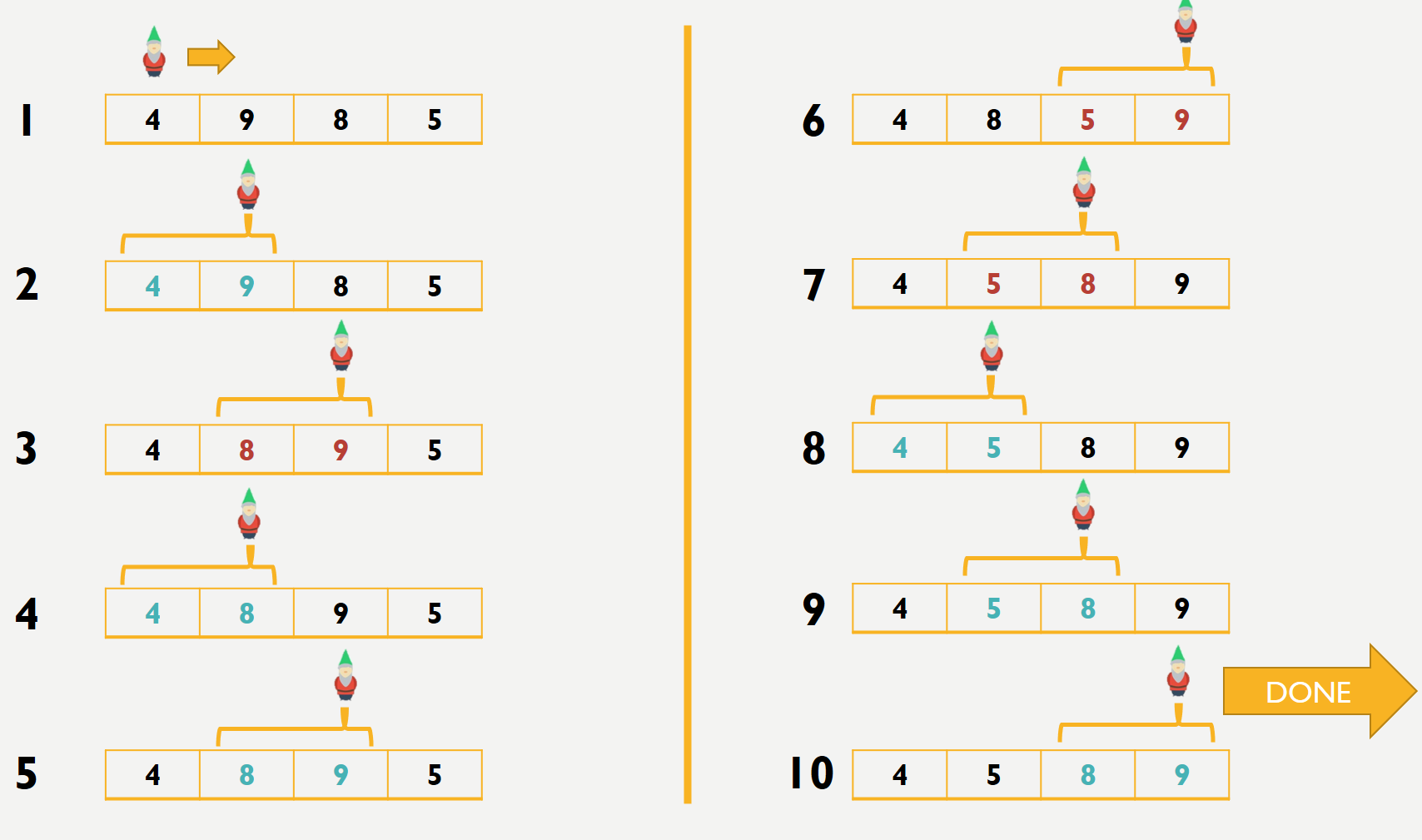


Рисунок 1 - Гномья сортировка

# 2. Пузырьковая сортировка (Bubble sort)

Пузырьковая сортировка известна своей низкой скоростью, однако на концептуальном уровне — это простейший алгоритм сортировки.[4]

Идея состоит в том, чтобы сравнивать значения соседних элементов с их значением. Если текущий элемент больше следующего, то меняем их местами. Повторение алгоритма необходимо до того, пока массив не будет отсортирован.

Расстояние и направление элементов в процессе сортировки влияют на производительность, так как они перемещаются по разным направлениям с различной скоростью. Элемент, который должен быть перемещен в конец списка, может быстро перемещаться, так как он может участвовать в последовательной замене. Например, самый большой элемент в списке будет выигрывать от каждого обмена, и поэтому он перемещается на свою позицию после первого прохода. С другой стороны, элемент, который должен быть в начале списка, может двигаться очень медленно, так как он не сможет сделать больше одного шага за один проход. Если самый маленький элемент списка находится в конце списка, требуется n-1 проход для того, чтобы его переместить.[5]

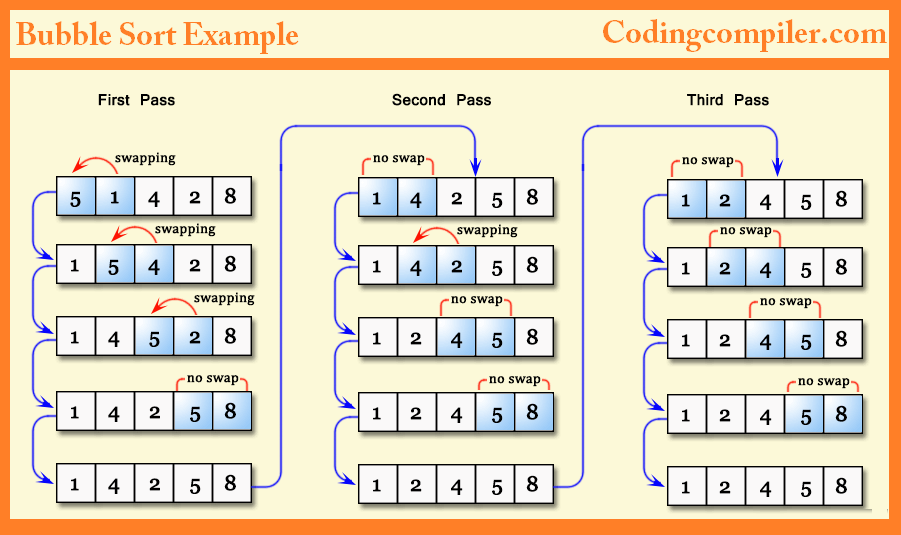


Рисунок 2 - Пузырьковая сортировка

# 3. Шейкерная сортировка (Shaker sort)

Шейкерная сортировка также называется сортировкой перемешиванием или двунаправленной. Данная сортировка представляет собой разновидность пузырьковой.[6]

Анализируя процесс сортировки пузырьком, можно заметить два момента:

1. если в процессе движения по массиву не происходит перестановок, то эта часть массива уже была отсортирована и её можно исключить.
2. при переходе от конца к началу массива минимальный элемент сдвигается на первую позицию, а максимальный элемент сдвигается на одну позицию вправо.

Это две идеи могут привести к следующим изменениям в процессе сортировки пузырьком. Границы части массива, в которой происходит движение, определяются в месте, где происходит последний обмен на каждой итерации. Массив просматривается поочередно слева направо и справа налево.

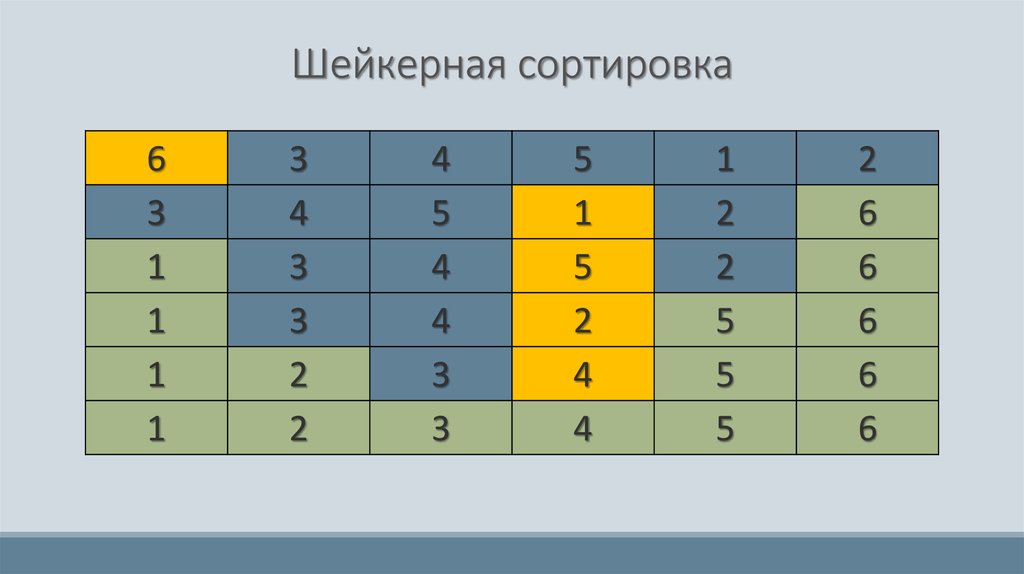


Рисунок 3 - Шейкерная сортировка

# 4. Сравнение сортировок обменами

Чтобы сравнить сортировки обменами по скорости выполнения, были созданы и заполнены числами три файла: первый файл: 300000 чисел в порядке возрастания, второй файл: 300000 чисел в порядке убывания, третий файл: числа от 1 до 10, 300000 чисел вразброс (см. Приложение 1).

Далее были применены гномья, пузырьковая и шейкерная сортировки к этим трем файлам и засечено время выполнения программ:

1. Гномья сортировка

При сортировке первого файла время выполнения программы составило 0,003 секунды, второго файла – 132 секунды, третьего файла – 60 секунд (см. Приложение 2).

2. Пузырьковая сортировка

При пузырьковой сортировке первого файла время выполнения программы составило 0,005 секунд, второго файла – 120 секунд, третьего файла – 159 секунд (см. Приложение 3).

3. Шейкерная сортировка

При шейкерной сортировке первого файла время выполнения программы составило 0,003 секунды, второго файла – 64 секунды, третьего файла – 117 секунд (см. Приложение 4).

Сортировка первого файла заняла меньше времени при гномьей и шейкерной сортировке. Сортировка второго файла получилась быстрее с помощью шейкерной сортировки. Сортировка третьего файла заняла меньше времени при гномьей сортировке.

Временная сложность для этих трех алгоритмов одинаковая: в лучшем случае O(n), в худшем случае O(n2) и в среднем случае O(n2).

# Выводы

В ходе выполнения курсовой работы были рассмотрены и изучены методы сортировки данных обменами: гномья, пузырьковая и шейкерная. Также, были написаны программы на java, реализующие данные сортировки.

Для того, чтобы сравнить алгоритмы сортировки по скорости выполнения, было создано три файла и засечено время выполнения сортировок.

Шейкерная сортировка оказалось самой быстрой, если сравнить среднее время по всем файлам. Однако, гномья сортировка оказалась в два раза быстрее при ее выполнении к третьему файлу. Пузырьковая сортировка занимает больше всего времени и является самой медленной.

Время выполнения сортировок разное, но это расхождение незначительное, потому что временная сложность этих алгоритмов одинаковая.

# Список литературы

1. Ткачев Ф.В. Алгоритмы и структуры данных. - М.: ДМК Пресс, 2010. - 272 с.
2. Bimlibik : официальный сайт / Алгоритмы сортировки, 04.11.2020. – URL : [https://bimlibik.github.io/](https://usaprosto.ru/)posts/sorting-algorithm/ дата обращения: 29.10.2023.– Текст: электронный
3. Kvodo : официальный сайт / Гномья сортировка, 08.04.2021. – URL : [https://kvodo.ru/gnome-sorting](https://usaprosto.ru/).html дата обращения: 29.10.2023.– Текст: электронный
4. Лафоре Р. Структуры данных и алгоритмы в Java. - 2-е изд. - СПб.: Питер, 2013. - 704 с.
5. Код : официальный сайт / Как работает пузырьковая сортировка, 05.07.2021. – URL : [https://thecode.media/bubble-sort](https://usaprosto.ru/)/ дата обращения: 29.04.2023.– Текст: электронный
6. Хабр : официальный сайт / Сортировки обменами, 20.06.2018. – URL : [https://habr.com/](https://usaprosto.ru/)ru/articles/414653/ дата обращения: 29.10.2023.– Текст: электронный

# Приложение 1

Заполнение трех файлов

**import** java.io.\*;

**import** java.util.Random;

**public** **class** Test {

**public** **static** **void** main(String[] args) **throws** IOException{

**int** n = 300000;

PrintWriter file1 = **new** PrintWriter(**new** FileWriter("D:\\1.txt"));

**for** (**int** i = 1; i < n; i++) {

file1.print(i + " ");

}

file1.close();

PrintWriter file2 = **new** PrintWriter(**new** FileWriter("D:\\2.txt"));

**for** (**int** i = n; i > 0; i--) {

file2.print(i + " ");

}

file2.close();

Random r = **new** Random();

PrintWriter file3 = **new** PrintWriter(**new** FileWriter("D:\\3.txt"));

**for** (**int** i = 1; i < n; i++) {

**int** a = r.nextInt(10) + 1;

file3.print(a + " ");

}

file3.close();

}

}

# Приложение 2

Гномья сортировка (Gnome sort)

**import** java.io.\*;

**import** java.time.Duration;

**import** java.time.Instant;

**public** **class** Gnome {

**static** **void** gnomeSort(**double** a[], **int** n){

**int** i = 0;

**while** (i < n) {

**if** (i == 0)

i++;

**if** (a[i] >= a[i - 1])

i++;

**else** {

**double** temp = 0;

temp = a[i];

a[i] = a[i - 1];

a[i - 1] = temp;

i--;

}

}

**return**;

}

**public** **static** **void** main(String[] args) **throws** IOException{

File file1 = **new** File("D:\\1.txt");

FileReader fr = **new** FileReader(file1);

BufferedReader br = **new** BufferedReader(fr);

String text = "";

**while** (br.ready()){

text+=br.readLine();

}

String[] words = text.split(" ");

**double** a []= **new** **double** [words.length];

**for** (**int** i=0;i<words.length;i++){

a[i]=Integer.*parseInt*(words[i]);

}

Instant start=Instant.*now*();

*gnomeSort*(a, a.length);

Instant finish = Instant.*now*();

System.***out***.println();

**for**(**double** i : a)

System.***out***.print(i + " ");

fr.close();

br.close();

**long** elapsed = Duration.*between*(start,finish).toMillis();

System.***out***.println();

System.***out***.print("time in milliseconds: " + elapsed );

}

}

# Приложение 3

Пузырьковая сортировка (Bubble sort)

**import** java.io.\*;

**import** java.time.Duration;

**import** java.time.Instant;

**public** **class** Bubble {

**static** **void** bubbleSort(**double** a[]){

**boolean** f = **false**;

**double** b;

**while**(!f) {

f = **true**;

**for** (**int** i = 0; i < a.length-1; i++) {

**if**(a[i] > a[i+1]){

f = **false**;

b = a[i];

a[i] = a[i+1];

a[i+1] = b;

}

}

}

}

**public** **static** **void** main(String[] args) **throws** IOException{

File file1 = **new** File("D:\\1.txt");

FileReader fr = **new** FileReader(file1);

BufferedReader br = **new** BufferedReader(fr);

String text = "";

**while** (br.ready()){

text+=br.readLine();

}

String[] words = text.split(" ");

**double** a []= **new** **double** [words.length];

**for** (**int** i=0;i<words.length;i++){

a[i]=Integer.*parseInt*(words[i]);

}

Instant start=Instant.*now*();

*bubbleSort*(a);

Instant finish = Instant.*now*();

System.***out***.println();

**for**(**double** i : a)

System.***out***.print(i + " ");

fr.close();

br.close();

**long** elapsed = Duration.*between*(start,finish).toMillis();

System.***out***.println();

System.***out***.print("time in milliseconds: " + elapsed );

}

}

# Приложение 4

Шейкерная сортировка (Shaker sort)

**import** java.io.\*;

**import** java.time.Duration;

**import** java.time.Instant;

**public** **class** Shaker {

**static** **void** swap(**double**[] a, **int** i, **int** j){

**double** temp = a[i];

a[i] = a[j];

a[j] = temp;

}

**static** **void** shakerSort(**double** a[], **int** n){

**boolean** swapped = **true**;

**int** start = 0;

**int** end = n - 1;

**while** (swapped) {

swapped = **false**;

**for** (**int** i = start; i < end; i++) {

**if** (a[i] > a[i + 1]) {

*swap*(a, i, i + 1);

swapped = **true**;

}

}

**if** (!swapped)

**break**;

swapped = **false**;

end--;

**for** (**int** i = end - 1; i >= start; i--) {

**if** (a[i] > a[i + 1]) {

*swap*(a, i, i + 1);

swapped = **true**;

}

}

start++;

}

}

**public** **static** **void** main(String[] args) **throws** IOException{

File file1 = **new** File("D:\\1.txt");

FileReader fr = **new** FileReader(file1);

BufferedReader br = **new** BufferedReader(fr);

String text = "";

**while** (br.ready()){

text+=br.readLine();

}

String[] words = text.split(" ");

**double** a []= **new** **double** [words.length];

**for** (**int** i=0;i<words.length;i++){

a[i]=Integer.*parseInt*(words[i]);

}

Instant start=Instant.*now*();

*shakerSort*(a, a.length);

Instant finish = Instant.*now*();

System.***out***.println();

**for**(**double** i : a)

System.***out***.print(i + " ");

fr.close();

br.close();

**long** elapsed = Duration.*between*(start,finish).toMillis();

System.***out***.println();

System.***out***.print("time in milliseconds: " + elapsed );

}

}