Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Прикладная математика

Направление: 24.03.02 «Системы управления движением и навигация»

Программа: «Программное и математическое обеспечение систем навигации и управления»

УТВЕРЖДАЮ

**Зав. кафедрой ВММБ**

\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Ю. Столбов

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

***ЗАДАНИЕ***

***НА КУРСОВУЮ РАБОТУ***

***по дисциплине***

**«Алгоритмы и структуры данных»**

(фамилия, имя, отчество студента; группа)

1. **Тема курсовой работы**

2. **Срок сдачи студентом отчета:** «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

3. **Содержание отчета:**

Теоретическая часть включает в себя информацию о сортировках обменами, их истории, свойствах и

недостатках. В практической части сравниваются алгоритмы сортировок обменами. Количество страниц: 20,

количество рисунков: 6.

Руководитель курсовой работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Научный руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский**

**политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Прикладная математика

Направление: 24.03.02 «Системы управления движением и навигация»

Программа: «Программное и математическое обеспечение систем навигации и управления»

**КУРСОВАЯ РАБОТА**

по дисциплине

**«Алгоритмы и структуры данных»**

Тема: **«**Сравнение сортировок по времени**»**

Выполнил:

студент гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись, дата)*

Принял:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2023**

**Содержание**

**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:

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

**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:

* процесс упорядочения прекращается тогда, когда не выполнено условие i<n, где n – размер массива.
* 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

**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

**Сортировка слиянием**

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности n/2 в единую последовательность размерности n. Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.



Рисунок 3 - сортировка слиянием

**Сравнение сортировок обменами**

Чтобы сравнить сортировки обменами по скорости выполнения, были созданы и заполнен файл рандомными значениями (см. Приложение 1).

Далее были применены гномья, пузырьковая и слиянием сортировки к этому файлу и засечено время выполнения программ (рисунок 4).



Рисуок 4 - сравнение сортировок

**Выводы**

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

Для того, чтобы сравнить алгоритмы сортировки по скорости выполнения, был построен график.

Сортировка слиянием самая быстрая, однако для больших массивов не устойчива.

**Список литературы**

* Ткачев Ф.В. Алгоритмы и структуры данных. - М.: ДМК Пресс, 2010. - 272 с.
* Bimlibik : официальный сайт / Алгоритмы сортировки, 04.11.2020. – URL : [https://bimlibik.github.io/](https://usaprosto.ru/)posts/sorting-algorithm/ дата обращения: 26.10.2023.– Текст: электронный
* Kvodo : официальный сайт / Гномья сортировка, 08.04.2021. – URL : [https://kvodo.ru/gnome-sorting](https://usaprosto.ru/).html дата обращения: 26.10.2023.– Текст: электронный
* Лафоре Р. Структуры данных и алгоритмы в Java. - 2-е изд. - СПб.: Питер, 2013. - 704 с.
* Код : официальный сайт / Как работает пузырьковая сортировка, 05.07.2021. – URL : [https://thecode.media/bubble-sort](https://usaprosto.ru/)/ дата обращения: 26.10.2023.– Текст: электронный
* Хабр : официальный сайт / Сортировки обменами, 20.06.2018. – URL : [https://habr.com/](https://usaprosto.ru/)ru/articles/414653/ дата обращения: 26.10.2023.– 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;

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** arr[], **int** n){

**int** index = 0;

**while** (index < n) {

**if** (index == 0)

index++;

**if** (arr[index] >= arr[index - 1])

index++;

**else** {

**double** temp = 0;

temp = arr[index];

arr[index] = arr[index - 1];

arr[index - 1] = temp;

index--;

}

}

**return**;

}

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

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

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

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

String text = "";

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

text+=br.readLine();

}

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

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

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

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

}

Instant start=Instant.*now*();

*gnomeSort*(a, a.length);

Instant finish = Instant.*now*();

fr.close();

br.close();

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

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

System.***out***.print("Время сортировки файла: " + elapsed + " мс" );

}

}

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

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

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

String text = "";

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

text+=br.readLine();

}

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

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

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

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

}

Instant start=Instant.*now*();

*bubbleSort*(a);

Instant finish = Instant.*now*();

fr.close();

br.close();

**long** elapsed3 = Duration.*between*(start,finish).toMillis();

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

System.***out***.print("Время сортировки файла: " + elapsed + " мс" );

}

}

**Приложение 4**

Сортировка слиянием

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.time.Duration;

**import** java.time.Instant;

**import** java.util.Random;

**class** SortSliyaniemFile {

**public** **static** **void** main(String[] args)

{

**int** mas [] = **new** **int** [20000000];

Random rnd = **new** Random();

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

{

mas[i] = rnd.nextInt(15);

}

**int** n=mas.length;

Instant start=Instant.*now*();

*mergeSort*( mas, n);

Instant end = Instant.*now*();

Duration interval = Duration.*between*(start, end);

System.***out***.println("Execution time in seconds: " + interval.getSeconds());

}

**public** **static** **void** mergeSort(**int**[] mas, **int** n) {

**if** (n < 2) //если меньше 2 элементов в массиве, то возвращаемся

{

**return**;

}

**int** mid = n / 2;// ищем середину массива

//объявляем 2 подмассива

**int**[] mas1 = **new** **int**[mid];

**int**[] mas2 = **new** **int**[n - mid];

//заполняем два вспомогательных массива элементами из первого массива

**for** (**int** i = 0; i < mid; i++) {

mas1[i] = mas[i];

}

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

mas2[i - mid] =mas[i];

}

*mergeSort*(mas1, mid);

*mergeSort*(mas2, n - mid);

*merge*(mas, mas1, mas2, mid, n - mid);

}

**public** **static** **void** merge(**int** mas[], **int** mas1[],**int** mas2[], **int** mid, **int** n)

{

**int** i=0,j=0,k=0;

**while**(i<mid && j<n )

{

**if**(mas1[i]<=mas2[j])

{

mas[k]=mas1[i];

i++;

k++;

} **else**

{

mas[k]=mas2[j];

j++;

k++;

}

}

**while**(i<mid)

{

mas[k]=mas1[i];

i++;

k++;

}

**while**(j<n)

{

mas[k]=mas2[j];

j++;

k++;

}

}

**public** **static** **void** printmas (**int** [] mas)

{

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

{

System.***out***.print(mas[i]+"\t");

}

}

}