**МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«тюменский ИНДУСТРИАЛЬНЫЙ университет»**

**ИНСТИТУТ ГЕОЛОГИИ И НЕФТЕГАЗОДОБЫЧИ**

Кафедра Кибернетических систем

ОТЧЕТ

по практической работе

по дисциплине «Структуры и алгоритмы обработки данных»

на тему «Сортировка массива»

Вариант №11

|  |  |
| --- | --- |
|  | Выполнил:  студент группы АСОиУб – 21 – 2  Астахов С.О  Проверил**:**  ст. преподаватель кафедры КС  Лозикова И.О. |

Тюмень,

ТИУ

2022 г.

Оглавление

[Постановка задачи 3](#_Toc119512339)

[Описание алгоритмов сортировки 5](#_Toc119512340)

[Описание массива и способы формирования исходнных данных 6](#_Toc119512341)

[Определение переменных программы 7](#_Toc119512342)

[Анализ сортировок 8](#_Toc119512343)

[1. Описание методики тестирования трудоёмкости алгоритмов сортировки 8](#_Toc119512344)

[2. Оценка трудоёмкости алгоритмов операций для наилучшего, среднего и наихудшего случаев функционирования 9](#_Toc119512345)

[3. Сравнительный анализ теоретических и экспериментальных оценок эффективности алгоритмов сортировок 16](#_Toc119512346)

[Заключение 17](#_Toc119512347)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 18](#_Toc119512348)

[Приложение 1 19](#_Toc119512349)

[Приложение 2 20](#_Toc119512350)

[Приложение 3 21](#_Toc119512351)

# **Постановка задачи**

Цель: рассмотреть алгоритм элементарной и эффективной сортировки, разработать, протестировать и проанализировать данные алгоритмы, оценить производительность данных алгоритмов и сравнить их между собой по различным характеристикам.

Задание:

1.  Реализовать и провести тестовые испытания разных методов сортировок одномерного массива.

2. Разработать консольную программу с заданным интерфейсом в виде меню операций и набором дополнительных операций для вывода показателей эффективности. Архитектура программы – функциональная декомпозиция всех операций

Интерфейс программы включает следующие операции:

* изменение размера массива с шагом в заданном диапазоне
* формирование в массиве случайной выборки значений,
* формирование в массиве упорядоченной выборки значений,
* чтение/запись по индексу,
* элементарная сортировка (по варианту задания),
* эффективная сортировка (по варианту задания).

Для тестирования эффективности алгоритмов сортировки программа реализует вычисления и вывод в виде таблицы показателей эффективности:

* опрос числа выполненных сравнений,
* опрос числа выполненных обменов.
* вычисление системного времени

3. Выполнить отладку и тестирование отдельных операций с помощью меню операций.

4. Выполнить сравнительное тестирование эффективности алгоритмов сортировки для лучшего, худшего и среднего случаев для массива

5. Провести анализ экспериментальных показателей трудоёмкости алгоритмов сортировки.

6. Составить **отчёт** по практической работе. Отчёт должен содержать следующие пункты:

 1) титульный лист,

 2) цель лабораторной работы,

 3) общее задание (пункты 1-5) и вариант задания,

 4) описание алгоритмов сортировки и ссылка на Приложение 1 с исходным текстом функций сортировки,

 5) описание массива, способы формирования исходных данных для наилучшего, наихудшего и среднего случая со ссылкой на Приложение 2,

 6) определение переменных программы (показателей эффективности) и описание методики тестирования эффективности алгоритмов сортировки со ссылкой на Приложение 3 и Приложение 1,

 7) таблицы и графики с полученными оценками показателей эффективности алгоритмов операций для наилучшего, наихудшего и среднего случаев функционирования. Должны быть приведены следующие графики:

1. число обменов и число сравнений для лучшего, худшего и среднего случаев для первого алгоритма сортировки (графики совмещены в одной системе координат) при разных размерах исходных данных
2. число обменов и число сравнений для лучшего, худшего и среднего случаев для второго алгоритма сортировки (графики совмещены) при разных размерах исходных данных
3. временной график, где показана зависимость времени выполнения от размера массива для обоих методов сортировки,
4. число сравнений алгоритмов методов сортировки для среднего случая при разных размерах исходных данных,
5. число обменов алгоритмов методов сортировки для среднего случая при разных размерах исходных данных

 8) сравнительный анализ теоретических и экспериментальных оценок эффективности алгоритмов сортировок,

 9) выводы,

 10) список использованной литературы,

 11) приложение с текстами программ:

* Приложение 1. Исходный текст алгоритмов сортировок программы.
* Приложение 2. Исходный текст функций формирования исходных наборов данных.
* Приложение 3. Текст программы тестирования эффективности сортировок.

Вариант 11: Алгоритм сортировки пузырьком, алгоритм шейкерной сортировки.

# **Описание алгоритмов сортировки**

1. Сортировка пузырьком — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале.

Идея метода состоит в том, чтобы на каждом шаге находить наибольший элемент из двух соседних и ставить этот элемент в конец пары.

Шаги алгоритма:

1. Берём самый первый элемент массива и сравниваем его со вторым. Если первый больше второго — меняем их местами с первым, если нет — ничего не делаем.
2. Затем берём второй элемент массива и сравниваем его со следующим — третьим. Если второй больше третьего — меняем их местами, если нет — ничего не делаем.
3. Проходим так до предпоследнего элемента, сравниваем его с последним и ставим наибольший из них в конец массива. Всё, мы нашли самое большое число в массиве и поставили его на своё место.
4. Возвращаемся в начало алгоритма и делаем всё снова точно так же, начиная с первого и второго элемента. Только теперь даём себе задание не проверять последний элемент — мы знаем, что теперь в конце массива самый большой элемент.
5. Когда закончим очередной проход — уменьшаем значение финальной позиции, до которой проверяем, и снова начинаем сначала.
6. Так делаем до тех пор, пока у нас не останется один элемент
7. Шейкерная сортировка – сортировка представляет собой двунаправленную пузырьковую сортировку. В этом случае алгоритм обрабатывает массив сначала слева направо, перемещая таким образом наибольший элемент в конец массива, а затем справа налево, перемещая наименьший элемент в начало массива.

Идея метода состоит в том, чтобы на каждом обходе слево на право находить наибольший элемент из двух соседних и ставить этот элемент в конец пары ,а при обходе справо на лево наоборот.

Шаги алгоритма:

1. Берём самый первый элемент массива и сравниваем его со вторым. Если первый больше второго — меняем их местами с первым, если нет — ничего не делаем.
2. Затем берём второй элемент массива и сравниваем его со следующим — третьим. Если второй больше третьего — меняем их местами, если нет — ничего не делаем.
3. Проходим так до предпоследнего элемента, сравниваем его с последним и ставим наибольший из них в конец массива. Всё, мы нашли самое большое число в массиве и поставили его на своё место.
4. С конца массива сравнивание поселедний элемент с предпоследним и меняем если последний меньше предыдущего. И так для всех следующих элементов.
5. Когда закончим очередной проход — уменьшаем значение финальной позиции, до которой проверяем с обоих строн, и снова начинаем сначала.
6. Так делаем до тех пор, пока у нас не останется один элемент

# **Описание массива и способы формирования исходнных данных**

const int size = 1000 - переменная для размера массива

int Array[size] – статический массив, в котором происходит заполнение значениями.

Способы формирования исходных данных:

1. [Лучший случай](#_Приложение_2)

Формирование исходных данных для лучшего случая является уже полностью отсортированный массив в лексикографическом порядке начиная с нуля с помощью цикла for.

1. [Средний случай](#_Приложение_2)

Формирование исходных данных для среднего случая является массив, заполненный случайными числами в диапазоне от нуля до размера массива с помощью цикла for.

1. [Худший случай](#_Приложение_2)

Формирование исходных данных для худшего случая является массив, отсортированный в обратном лексикографическом порядке начиная с максимально возможного целого числа с помощью цикла for.

# **Определение переменных программы и Описание методики тестирования трудоёмкости алгоритмов сортировки**

auto t0 = clock() - время начала работы сортировок

auto t1 = clock() - время конца работы сортировок

double duration – время работы сортировок

long long changeCounterBubble – количество перестановок во время работы сортировки пузырьком

long long compareCounterBubble – количество сравнений во время работы сортировки пузырьком

long long changeCounterMixing – количество перестановок во время работы сортировки шейкером

long long compareCounterMixing – количество сравнений во время работы сортировки шейркером

void BubbleSort() – сортировка выбором

void MixingSort() – быстрая сортировка

void WriteData()- Запись данных сортировки в txt файл

Описание методики тестирования трудоёмкости алгоритмов сортировки:

1. [Методика тестирования временной сложности сортировок](#_Приложение_3):

Для определения необходимого времени программе для выполнения сортировки можно с помощью функции steady\_clock::now ()

Текущее системное время сохраняется в переменной с типом данных time\_point — это текущий момент времени. При выводе вычисляем выражение разницы двух моментов времени (start, end) и сохраняем в переменную elapsed, показывающий время работы алгоритма сортировки в секундах. Таким набором операторов С++ вы сможете замерить время работы части программы, результат будет в секундах.

# **Анализ сортировок**

1. Описание трудоёмкости алгоритмов сортировки

Чтобы сравнить данные сортировки была организована программа, которая считает количество сравнений и перестановок в алгоритме, а также время работы алгоритма для определенных размеров массива. Рассмотрим массивы размерами N изменяется на отрезке [1000; 100000] с шагом dN=10000.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Лучший случай | Средний случай | Худший случай |
| Сортировка пузырьком | O(n2) | O(n2) | O(n2) |
| Шейкерная сортировка | O(n2) | O(n2) | O(n2) |

Трудоёмкость сортировки пузырьком составляет O(N2), это означает, что по мере увеличения количества входных данных время работы алгоритма (использование памяти, либо другой измеряемый параметр) возрастает квадратично.

Трудоёмкость сортировки шейкером составляет O(N2), это означает, что по мере увеличения количества входных данных время работы алгоритма (использование памяти, либо другой измеряемый параметр) возрастает квадратично.

Этими закономерностями и будем пользоваться при тестировании трудоёмкости алгоритмов. А именно, за сравниваемые величины взяты количество сравнений и количество обменов.

1. Оценка трудоёмкости алгоритмов операций для наилучшего, среднего и наихудшего случаев функционирования
2. число обменов и число сравнений для алгоритма сортировки пузырьком

Таблица 1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Лучший случай | | Средний случай | | | Худший случай | | |
| Размер | Число сравнений | Время | Число сравнений | Число перестановок | Время | Число сравнений | Число перестановок | Время |
| 1000 | 998001 | 0,002 | 998001 | 259178 | 0.013 | 998001 | 499500 | 0,012 |
| 10000 | 99980001 | 0,262 | 99980001 | 24807578 | 0,933 | 99980001 | 49995000 | 1,296 |
| 20000 | 399960001 | 1,021 | 399960001 | 100762321 | 3,639 | 399960001 | 199990000 | 5,231 |
| 30000 | 899940001 | 2,292 | 899940001 | 224893591 | 8,194 | 899940001 | 449985000 | 11,689 |
| 40000 | 1599920001 | 4,098 | 1599920001 | 401317365 | 14,654 | 1599920001 | 799980000 | 20,797 |
| 50000 | 2499900001 | 6,376 | 2499900001 | 624647794 | 22,826 | 2499900001 | 1249975000 | 32,552 |
| 60000 | 3599880001 | 9,363 | 3599880001 | 898102801 | 32,8 | 3599880001 | 1799970000 | 46,869 |
| 70000 | 4899860001 | 12,566 | 4899860001 | 1225297454 | 44,582 | 4899860001 | 2449965000 | 63,975 |
| 80000 | 6399840001 | 16,326 | 6399840001 | 1603815220 | 58,308 | 6399840001 | 3199960000 | 83,602 |
| 90000 | 8099820001 | 20,660 | 8099820001 | 2029511469 | 73,666 | 8099820001 | 4049955000 | 107,209 |
| 100000 | 9999800001 | 25,450 | 9999800001 | 2491848451 | 91,853 | 9999800001 | 4999950000 | 133,204 |

Рисунок 1 – Графики характеристик сортировки пузырьком

Из графиков (рис. 1) видно, что сложность сортировки пузырьком с увеличением входных данных, количество перестановок возрастает квадратично.

1. число обменов и число сравнений для алгоритма шейкерной сортировки

Таблица 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Лучший случай | | Средний случай | | | Худший случай | | |
| Размер | Число сравнений | Время | Число сравнений | Число перестановок | Время | Число сравнений | Число перестановок | Время |
| 1000 | 500500 | 0 | 500500 | 259178 | 0.008 | 500500 | 499500 | 0.011 |
| 10000 | 50005000 | 0.124 | 50005000 | 24807578 | 0.728 | 50005000 | 49995000 | 1.219 |
| 20000 | 200010000 | 0.502 | 200010000 | 100762321 | 2.907 | 200010000 | 199990000 | 4.831 |
| 30000 | 450015000 | 1.132 | 450015000 | 224893591 | 6.567 | 450015000 | 449985000 | 11.067 |
| 40000 | 800020000 | 2.019 | 800020000 | 401317365 | 11.748 | 800020000 | 799980000 | 19.355 |
| 50000 | 1250025000 | 3.139 | 1250025000 | 624647794 | 18.305 | 1250025000 | 1249975000 | 29.501 |
| 60000 | 1800030000 | 4.534 | 1800030000 | 898102801 | 26.435 | 1800030000 | 1799970000 | 43.339 |
| 70000 | 2450035000 | 6.199 | 2450035000 | 1225297454 | 35.705 | 2450035000 | 2449965000 | 57.855 |
| 80000 | 3200040000 | 8.066 | 3200040000 | 1603815220 | 46.713 | 3200040000 | 3199960000 | 75.572 |
| 90000 | 4050045000 | 10.165 | 4050045000 | 2029511469 | 59.378 | 4050045000 | 4049955000 | 96.767 |
| 100000 | 5000050000 | 12.605 | 5000050000 | 2491848451 | 73.539 | 5000050000 | 4999950000 | 118.121 |

Рисунок 2 – Графики характеристик шейкерной сортировки

Из графиков (рис. 2) видно, что сложность сортировки шейкером с увеличением входных данных, количество перестановок возрастает квадратично.

1. временной график алгоритмов элементарной и эффективной сортировки

Таблица 3

|  |  |  |
| --- | --- | --- |
| Размер | Сортировка пузырьком | шейкерная сортировка |
| 1000 | 0.013 | 0,008 |
| 10000 | 0,933 | 0,728 |
| 20000 | 3,639 | 2,907 |
| 30000 | 8,194 | 6,567 |
| 40000 | 14,654 | 11,748 |
| 50000 | 22,826 | 18,305 |
| 60000 | 32,8 | 26,435 |
| 70000 | 44,582 | 35,705 |
| 80000 | 58,308 | 46,713 |
| 90000 | 73,666 | 59,378 |
| 100000 | 91,853 | 73,539 |

Рисунок 3 – Временная зависимость сортировок

Из графиков (рис. 3) видно, что временная сложность обоих алгоритмов сортировки с увеличением входных данных, возрастает квадратично

1. число сравнений алгоритмов элементарной и эффективной сортировки для среднего случая

Таблица 4

|  |  |  |
| --- | --- | --- |
| Размер | Сортировка пузырьком | Шейкерная сортировка |
| 1000 | 998001 | 500500 |
| 10000 | 99980001 | 50005000 |
| 20000 | 399960001 | 200010000 |
| 30000 | 899940001 | 450015000 |
| 40000 | 1599920001 | 800020000 |
| 50000 | 2499900001 | 1250025000 |
| 60000 | 3599880001 | 1800030000 |
| 70000 | 4899860001 | 2450035000 |
| 80000 | 6399840001 | 3200040000 |
| 90000 | 8099820001 | 4050045000 |
| 100000 | 9999800001 | 5000050000 |

Рисунок 4 – количество сравнений сортировок

Из графиков (рис. 3) видно, что временная сложность обоих алгоритмов сортировки с увеличением входных данных, возрастает квадратично

1. число обменов алгоритмов элементарной и эффективной сортировки для среднего случая

Таблица 5

|  |  |  |
| --- | --- | --- |
| Размер | сортировка пузырьком | шейкерная сортировка |
| 1000 | 259178 | 259178 |
| 10000 | 24807578 | 24807578 |
| 20000 | 100762321 | 100762321 |
| 30000 | 224893591 | 224893591 |
| 40000 | 401317365 | 401317365 |
| 50000 | 624647794 | 624647794 |
| 60000 | 898102801 | 898102801 |
| 70000 | 1225297454 | 1225297454 |
| 80000 | 1603815220 | 1603815220 |
| 90000 | 2029511469 | 2029511469 |
| 100000 | 2491848451 | 2491848451 |

Рисунок 5 – количество обменов сортировок

Из графиков (рис. 3) видно, что временная сложность обоих алгоритмов сортировки с увеличением входных данных, возрастает квадратично

## Сравнительный анализ теоретических и экспериментальных оценок эффективности алгоритмов сортировок

Рассмотрим сортировку пузырьком. При возрастании массива в N раз, число сравнений должно также возрасти в N^2 раз.

Допустим, если при n = 1000 – количество сравнений 499500, то при 10000 – количество должно возрасти в 100 раз. Проверяем 49950000, а по программе – 49995000. Вывод: все подтвердилось.

Рассмотрим сортировку шейкером. При возрастании массива в N раз, число сравнений должно также возрасти в N^2 раз.

Допустим, если при n = 1000 – количество сравнений 499500, то при 10000 – количество должно возрасти в 100 раз. Проверяем 49950000, а по программе – 49995000. Вывод: все подтвердилось.

Если говорить о времени работы алгоритма сортировок для обоих случаев, то важно отметить, что шейкерная сортировка быстрее производит сортировку, чем сортировка пузырьком. Так, для самого большого массива размером 100000, время работы сортировки пузырьком в худшем случае – 133,204 с, а быстрой шейкером – 0,0123648 c.

Также по графикам видно, что число сравнений и перестановок в сортировке шейкером меньше чем в пузырьковой.

# **Заключение**

Сортировка пузырьком имеет несколько преимуществ:

* Очень прост в реализации.
* Имеет преимущество в производительности в определенных ситуациях, особенно когда вспомогательная память ограничена.
* Также данная сортировка является устойчивой.

Из недостатков:

* Сортировка пузырьком всегда имеет сложность O(n2), что делает его неэффективным для использования в больших списках.

Достоинствами шейкерной сортировки является:

* Реализация не является сложной
* Модифицированная версия пузырьковой сортировки и работает немного быстрее.

К недостаткам можно отнести:

* Сортировка пузырьком всегда имеет сложность O(n2), что делает его неэффективным для использования в больших списках.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Ахо Альфред В. Структуры данных и алгоритмы: классическое издание. / Ахо Альфред В., Хопкрофт Джон Э., Ульман Джеффри Д. – Москва: Вильямс, 2018 – 400с. – Текст: непосредственный.
2. Хайнеман, Джордж, Поллис, Гэри, Селков, Стэнли. Алгоритмы. Справочник с примерами на С, C++, Java и Python, 2-е изд.: Пер. с англ. - СпБ. : ООО "Альфа-книга", 2017. - 432с.

# **Приложение 1**

1. Сортировка пузырьком

void BubbleSort(int\* array, int size)

{

auto t0 = clock();

for (int i = 0; i < size - 1; i++)

{

for (int j = 0; j < size - 1; j++)

{

if (array[j] > array[j + 1])

{

swap(array[j], array[j + 1]);

changeCounterBubble++;

}

compareCounterBubble++;

}

}

1. Быстрая шейкером

void MixingSort(int\* array, int size)

{

auto t0 = clock();

int left, right, i;

left = 0;

right = size - 1;

while (left <= right)

{

for (i = right; i >= left; i--) {

if (array[i - 1] > array[i]) {

swap(array[i - 1], array[i]);

changeCounterMixing++;

}

compareCounterMixing++;

}

left++;

for (i = left; i <= right; i++) {

if (array[i - 1] > array[i]) {

swap(array[i - 1], array[i]);

changeCounterMixing++;

}

compareCounterMixing++;

}

right--;

}

# **Приложение 2**

1. формирования исходных наборов данных для лучшего случая.

void SetSortValues(int\* array, int size)

{

for (int i = 0; i < size; i++)

array[i] =i;

}

1. формирования исходных наборов данных для среднего случая.

void SetRandValues(int\* array, int size)

{

for (int i = 0; i < size; i++)

array[i] = rand() % 100001;

}

1. Формирование исходных данных для худшего случая.

void SetSortValues(int\* array, int size)

{

for (int i = 0; i < size; i++)

array[i] =size-i;

}

# **Приложение 3**

// best case for bubble sort

SetRandValues(arr1, i);

auto t0 = clock();

BubbleSort(arr1, i);

auto t1 = clock();

duration <double> elapsed = t1 - t0;

WriteData(size,duration,changeCounterBubble, compareCounterBubble);