**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 №2

|  |  |
| --- | --- |
|  | Выполнил:  студент группы АСОиУб – 21 – 1  Быльцев С.А.  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 2: Алгоритм сортировки обменом (пузырьком), Алгоритм пирамидальной 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. [Пирамидальная сортировка](#_Приложение_1) – алгоритм сортировки, может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает (min-heap) / тонет (max-heap) по многим путям.

Использует бинарное сортирующее дерево.

Шаги алгоритма:

1. Выстраиваем элементы массива в виде сортирующего дерева.
2. Удаляем элементы из корня по одному за раз и перестраиваем дерево. (продолжаем процесс до тех пор, пока в сортирующем дереве не останется один элемент)

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

auto Array = new int[min] – динамический 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 startTime = clock() - время начала работы сортировок.

auto endTime = clock() - время конца работы сортировок.

double duration – время работы сортировок.

long long changeCount - количество перестановок во время работы сортировки.

long long comparisonCount - количество сравнений во время работы сортировки.

void BubbleSort - сортировка пузырьком.

void HeapSort - сортировка пирамидальная.

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 clock().

При выводе вычисляем выражение разницы двух моментов времени (startTime, endTime) и сохраняем в переменную duration, 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 определённых размеров массива. Рассмотрим массивы размерами SCALE изменяется на отрезке [10000; 100000] с шагом step = 10000.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Лучший случай | Средний случай | Худший случай |
| Сортировка пузырьком | O(n) | O(n2) | O(n2) |
| Пирамидальная сортировка | O(n\*log(n)) | O(n\*log(n)) | O(n\*log(n)) |

Трудоёмкость сортировки пузырьком в лучшем случае составляет O(n), это означает, что уже отсортированный массив сортируем дважды. Тогда в среднем и худшем случае O(n2), означает, что сортировка будет хорошо работать при небольшом количестве входных данных и по мере их увеличения время работы алгоритма будет возрастать квадратично.

Трудоёмкость пирамидальной сортировки составляет O(n\*log(n)), это означает, что данная сортировка удобная в использование во всех возможных случаях, удобная в работе с большими массивами.

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 пирамидальной сортировки

Таблица 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Лучший случай перестановок | | | Средний случай перестановок | | | Худший случай перестановок | | |
|  | Лучший случай сравнения | | | Средний случай сравнения | | | Худший случай сравнения | | |
| Размер | Сравнение | Перестановки | Время | Сравнение | Перестановки | Время | Сравнение | Перестановки | Время |
| 1000 | 500 | 1000 | 0.001 | 500 | 1000 | 0 | 500 | 1000 | 0 |
| 10000 | 5000 | 10000 | 0.003 | 5000 | 10000 | 0.004 | 5000 | 10000 | 0.003 |
| 20000 | 10000 | 20000 | 0.008 | 10000 | 20000 | 0.007 | 10000 | 20000 | 0.007 |
| 30000 | 15000 | 30000 | 0.01 | 15000 | 30000 | 0.011 | 15000 | 30000 | 0.01 |
| 40000 | 20000 | 40000 | 0.014 | 20000 | 40000 | 0.016 | 20000 | 40000 | 0.021 |
| 50000 | 25000 | 50000 | 0.022 | 25000 | 50000 | 0.02 | 25000 | 50000 | 0.018 |
| 60000 | 30000 | 60000 | 0.022 | 30000 | 60000 | 0.024 | 30000 | 60000 | 0.021 |
| 70000 | 35000 | 70000 | 0.027 | 35000 | 70000 | 0.029 | 35000 | 70000 | 0.025 |
| 80000 | 40000 | 80000 | 0.031 | 40000 | 80000 | 0.034 | 40000 | 80000 | 0.029 |
| 90000 | 45000 | 90000 | 0.035 | 45000 | 90000 | 0.04 | 45000 | 90000 | 0.033 |
| 100000 | 50000 | 100000 | 0.042 | 50000 | 100000 | 0.044 | 50000 | 100000 | 0.037 |

Рисунок 1 – График характеристики пирамидальной сортировки.

Из графика (рис. 1) видно, что сложность пирамидольной сортировки с увеличением входных данных, количество перестановок возрастает логарифмично.

1. число обменов и число сравнений для алгоритма сортировки пузырьком

Таблица 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Лучший случай перестановок | | Средний случай перестановок | | | Худший случай перестановок | | |
|  | Лучший случай сравнения | | Средний случай сравнения | | | Худший случай сравнения | | |
| Размер | Сравнение | Время | Сравнение | Перестановки | Время | Сравнение | Перестановки | Время |
| 1000 | 999 | 1.4e-06 | 968031 | 254317 | 0.002 | 999000 | 499500 | 0.003 |
| 10000 | 9999 | 1.17e-05 | 98370162 | 25040289 | 0.164 | 99990000 | 49995000 | 0.194 |
| 20000 | 19999 | 2.34e-05 | 394580270 | 99317818 | 0.694 | 399980000 | 199990000 | 0.785 |
| 30000 | 29999 | 3.48e-05 | 884250524 | 225703586 | 1.719 | 899970000 | 449985000 | 1.753 |
| 40000 | 39999 | 4.64e-05 | 1580600484 | 400186559 | 3.241 | 1599960000 | 799980000 | 3.132 |
| 50000 | 49999 | 5.79e-05 | 2492250154 | 626432351 | 5.374 | 2499950000 | 1249975000 | 4.899 |
| 60000 | 59999 | 6.95e-05 | 3580440325 | 899374523 | 7.959 | 3599940000 | 1799970000 | 7,075 |
| 70000 | 69999 | 8.11e-05 | 4883200239 | 1221790568 | 11,032 | 4899930000 | 2449965000 | 9.537 |
| 80000 | 79999 | 9.99e-05 | 6365920425 | 1601704737 | 14.61 | 6399920000 | 3199960000 | 12,49 |
| 90000 | 89999 | 0.0001047 | 8091900089 | 2032548819 | 18.635 | 8099910000 | 4049955000 | 15.837 |
| 100000 | 99999 | 0.0001313 | 9980100198 | 2504405759 | 23.242 | 9999900000 | 4999950000 | 19.551 |

Рисунок 2 – График характеристик сортировки пузырьком

Из графика (рис. 2) видно, что сложность пузырьковой сортировки с увеличением входных данных, количество перестановок и сравнений возрастает квадратично.

1. временной график алгоритмов элементарной и эффективной сортировки

Таблица 3

|  |  |  |
| --- | --- | --- |
|  | Пирамидальная сортировка | Пузырьковая сортировка |
| 1000 | 0 | 0,002 |
| 10000 | 0,004 | 0,164 |
| 20000 | 0,007 | 0,694 |
| 30000 | 0,011 | 1,719 |
| 40000 | 0,016 | 3,241 |
| 50000 | 0,02 | 5,374 |
| 60000 | 0,024 | 7,959 |
| 70000 | 0,029 | 11,032 |
| 80000 | 0,034 | 14,61 |
| 90000 | 0,04 | 18,635 |
| 100000 | 0,044 | 23,242 |

Рисунок 3 – Временная зависимость сортировок

Из графика (рис. 3) видно, что временная сложность сортировки пузырьком с увеличением входных данных, возрастает квадратично, из-за быстродействия пирамидальной сортировки трудно сказать, что временная сложность возрастает логарифмически.

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

Таблица 4

|  |  |  |
| --- | --- | --- |
|  | Пирамидальная сортировка | Пузырьковая сортировка |
| 1000 | 500 | 254317 |
| 10000 | 5000 | 25040289 |
| 20000 | 10000 | 99317818 |
| 30000 | 15000 | 225703586 |
| 40000 | 20000 | 400186559 |
| 50000 | 25000 | 626432351 |
| 60000 | 30000 | 899374523 |
| 70000 | 35000 | 1221790568 |
| 80000 | 40000 | 1601704737 |
| 90000 | 45000 | 2032548819 |
| 100000 | 50000 | 2504405759 |

Рисунок 4 – Количество сравнений сортировок

Из графика (рис. 4) видно, что количество сравнений сортировки пузырьком с увеличением входных данных, возрастает квадратично, из-за быстродействия пирамидальной сортировки трудно сказать, что количество сравнений возрастает логарифмически.

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

Таблица 5

|  |  |  |
| --- | --- | --- |
|  | Пирамидальная сортировка | Пузырьковая сортировка |
| 1000 | 1000 | 968031 |
| 10000 | 10000 | 98370162 |
| 20000 | 20000 | 394580270 |
| 30000 | 30000 | 884250524 |
| 40000 | 40000 | 1580600484 |
| 50000 | 50000 | 2492250154 |
| 60000 | 60000 | 3580440325 |
| 70000 | 70000 | 4883200239 |
| 80000 | 80000 | 6365920425 |
| 90000 | 90000 | 8091900089 |
| 100000 | 100000 | 9980100198 |

Рисунок 5 – количество обменов сортировок

Из графиков видно, что количество обменов сортировки пузырьком с увеличением входных данных, возрастает квадратично, из-за быстродействия пирамидальной сортировки трудно сказать, что количество обменов возрастает логарифмически.

## 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 пирамидальную сортировку. При возрастании массива в N раз, число сравнений должно возрасти в N \* log(N) раз.

Допустим, если при n = 1000 – количество сравнений 762, то при 10000 – количество должно возрасти в 10\*log10=10 раз. Проверяем 7620, а по программе – 7782. Вывод: все подтвердилось. По графикам нельзя явно сказать, что это логарифмическая зависимость.

Если говорить о времени работы алгоритма сортировок для обоих случаев, то важно отметить, что пирамидальная сортировка гораздо быстрее производит сортировку, чем сортировка пузырьком. Так, для самого большого массива размером 100000, время работы сортировки пузырьком в худшем случае – 19,551 с, а пирамидальной сортировки – 0,037 c.

Также по графикам видно, что число сравнений и перестановок в сортировке пузырьком в разы больше, чем в пирамидальной сортировке.

# **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.

К недостаткам можно отнести:

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

Достоинствами пирамидальной сортировки является:

* никаких дополнительных переменных, нужно лишь понимать схему.
* узлы хранятся от вершины и далее вниз, уровень за уровнем.
* узлы одного уровня хранятся в массиве слева направо.

К недостаткам можно отнести:

* Сложен в реализации.
* На почти отсортированных массивах работает столь же долго, как и на хаотических данных.

# **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

bool flag = true;

while (flag)

{

flag = false;

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

if (Arr[i] > Arr[i + 1])

{

int temp = Arr[i];

Arr[i] = Arr[i + 1];

Arr[i + 1] = temp;

flag = true;

changeCountBS++;

}

comparisonCountBS++;

}

}

1. Пирамидальная сортировка

for (int i = scale / 2 - 1; i >= 0; i--)

{

comparisonCountHS++;

Heap(Arr, scale, i);

}

for (int i = scale - 1; i >= 0; i--)

{

swap(Arr[0], Arr[i]);

changeCountHS++;

Heap(Arr, i, 0);

}

# 

# **Приложение 2**

1. формирования исходных наборов данных для лучшего случая.

void bestHappening(int\* Arr, int scale)

{

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

Arr[i] = i;

}

1. формирования исходных наборов данных для среднего случая.

void averageHappening(int\* Arr, int scale)

{

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

Arr[i] = 0 + rand() % (scale - 0 + 1);

}

1. Формирование исходных данных для худшего случая.

void badHappening(int\* Arr, int scale)

{

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

Arr[i] = INT\_MAX - i;

}

# **Приложение 3**

// best happening for Bubble sort

//cout << "BEST:" << endl;

bestHappening(Array, i);

BubbleSort(Array, i);

currentScale = i;