|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 3** | |
| --- | --- |
| **Тема:** | |
| **«Определение эффективного алгоритма сортировки на основе эмпирического и асимптотического методов анализа»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Харченко А.А. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_gjdgxs)

[2 ЗАДАНИЕ №1 5](#_30j0zll)

[2.1 Формулировка задачи 5](#_1fob9te)

[2.2 Описание шейкерной сортировки с условием Айверсона 6](#_2et92p0)

[2.2.1 Описание выполнения и блок-схема алгоритма шейкерной сортировки 6](#_tyjcwt)

[2.2.2 Доказательство корректности циклов алгоритма шейкерной сортировки с условием Айверсона 8](#_3dy6vkm)

[2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма шейкерной сортировки с условием Айверсона 9](#_1t3h5sf)

[2.3 Реализация алгоритма на языке C++ 10](#)

[2.4 Проведение тестирования и построение графика 11](#_mvru699w4vf3)

[2.4.1 Тестирование 11](#_2s8eyo1)

[2.4.2 Построение графика 12](#_8qmqvlt1hkm1)

[2.5 Описание алгоритма быстрой сортировки(Хоара) 13](#_17dp8vu)

[2.5.1 Описание выполнения и блок-схема алгоритма быстрой сортировки (Хоара) 13](#_3rdcrjn)

[2.5.2 Доказательство корректности циклов алгоритма быстрой сортировки (Хоара) 15](#_26in1rg)

[2.5.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма быстрой сортировки (Хоара) 16](#_lnxbz9)

[2.6 Реализация алгоритма на языке C++, проведение тестирования и построение графика 16](#_35nkun2)

[2.7 Проведение тестирования и построение графика 18](#_htm6j9tm3ne8)

[2.7.1 Тестирование 18](#_44sinio)

[2.7.2 Построение графика 18](#_3ng547m4bok5)

[2.8 Сортировка простым обменом 19](#_2jxsxqh)

[2.9 Сравнительный график зависимости для трёх алгоритмов 19](#_z337ya)

[2.10 Тестирование программ для алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара) 21](#_3j2qqm3)

[2.10.1 Тестирование с значениями упорядоченными по убыванию и построение графика для алгоритма шейкерной сортировки с условием Айверсона 21](#_1y810tw)

[2.10.2 Тестирование с значениями упорядоченными по возрастанию и построение графика для алгоритма шейкерной сортировки с условием Айверсона 24](#)

[2.10.3 Тестирование с значениями упорядоченном по убыванию и построение графика для алгоритма быстрой сортировки (Хоара) 26](#_4i7ojhp)

[2.10.4 Тестирование с значениями упорядоченном по возрастанию и построение графика для алгоритма быстрой сортировки (Хоара) 29](#)

[2.11 Вывод 31](#_1ci93xb)

[3 ЗАДАНИЕ №2 33](#_2bn6wsx)

[3.1 Формулировка задачи 33](#_qsh70q)

[3.2 Формулы Тт(n) для сортировки обменом 33](#_3as4poj)

[3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом 34](#)

[3.4 График функции роста 34](#_1pxezwc)

[3.5 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки с условием Айверсона 34](#_49x2ik5)

[3.6 Справочная информация о вычислительной сложности алгоритма быстрой сортировки(Хоара) 35](#_ph556g2g2ure)

[3.7 Таблица асимптотической сложности трёх алгоритмов 35](#_3o7alnk)

[4 ВЫВОДЫ 37](#_ihv636)

[5 ЛИТЕРАТУРА 38](#_32hioqz)

# 1 ЦЕЛЬ

Получить навыки по анализу вычислительной сложности алгоритмов сортировки и определению наиболее эффективного алгоритма.

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи**

Вариант 11. В списке 27. Шейкерная сортировка с условием Айверсона и быстрая сортировка(Хоара).

1. Разработать алгоритм усовершенствованной сортировки согласно варианту, реализовать код на языке С++. Сформировать таблицу 1.1 результатов эмпирической оценки сложности сортировки по формату табл. 1 для массива, заполненного случайными числами.

2. Определить ёмкостную сложность алгоритма усовершенствованной сортировки.

3. Разработать алгоритм быстрой сортировки, реализовать код на языке С++. Сформировать таблицу 1.2 результатов эмпирической оценки сортировки по формату табл. 1 для массива, заполненного случайными числами.

4. Определить ёмкостную сложность алгоритма быстрой сортировки.

5. Добавьте в отчёт данные по работе любого из алгоритмов простой сортировки в среднем случае, полученные в предыдущей практической работе (в отчёте – таблица 1.3).

6. Представить на общем сравнительном графике зависимости Тп(n)=Cф+Mф для трёх анализируемых алгоритмов. График должен быть подписан, на нём – обозначены оси.

7. На основе сравнения полученных данных определите наиболее эффективный из алгоритмов в среднем случае (отдельно для небольших массивов при n до 1000 и для больших массивов с n>1000).

8. Провести дополнительные прогоны программ усовершеноствованной и быстрой сортировок на массивах, отсортированных а) строго в убывающем и б) строго возрастающем порядке значений элементов. Заполнить по этим данным соответствующие таблицы 1.4, 1.5, 1.6 и 1.7 для каждого алгоритма по формату табл. 1.

9. Сделайте вывод о зависимости (или независимости) алгоритмов сортировок от исходной упорядоченности массива на основе результатов, представленных в таблицах.

## **2.2 Описание шейкерной сортировки с условием Айверсона**

### **2.2.1 Описание выполнения и блок-схема алгоритма шейкерной сортировки**

Шейкерная сортировка, также известная как сортировка перемешиванием, является усовершенствованным вариантом сортировки пузырьком. Она заключается в проходе по массиву элементов в обе стороны от середины, меняя местами соседние элементы, пока все элементы не будут упорядочены.

Условие Айверсона (также известное как условие Хоара) в шейкерной сортировке подразумевает наличие двух флагов: "flagUp" и "flagDown", показывающих, следует ли проходить массив с начала в конец или с конца в начало.

Процедура выполнения шейкерной сортировки с условием Айверсона:

* Устанавливаем индексы left = 0 и right = длина массива - 1.
* Пока left < right:
* Если flagUp равен true:
* Проходим по элементам массива слева направо с индексами от left до right. Если текущий элемент больше следующего, меняем их местами.
* Устанавливаем right = right - 1.
* Иначе, если flagUp равен false:
* Проходим по элементам массива справа налево с индексами от right до left. Если текущий элемент больше предыдущего, меняем их местами.
* Устанавливаем left = left + 1.
* Переключаем флаг flagUp на противоположное значение.

По завершении внешнего цикла все элементы массива будут упорядочены.

Условие Айверсона позволяет уменьшить количество проходов по неотсортированной части массива, обеспечивая более эффективную сортировку.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.1,2).

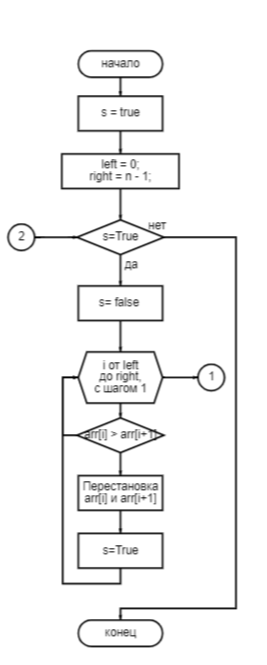


Рисунок 1 – Блок-схема алгоритма шейкерной сортировки

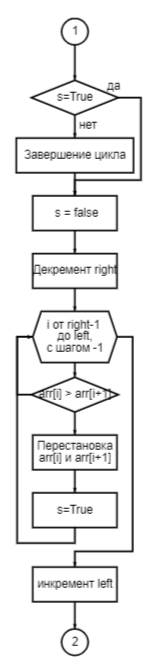


Рисунок 2 – Блок-схема алгоритма шейкерной сортировки

### **2.2.2 Доказательство корректности циклов алгоритма шейкерной сортировки с условием Айверсона**

Внешний цикл: на каждой итерации внешнего цикла массив делится на упорядоченную и неупорядоченную части. Упорядоченная часть находится справа (от индекса 0 до left) и слева (от индекса right до конца).

Внутренние циклы:

* При flagUp = true: на каждой итерации внутреннего цикла элементы слева от left стоят в порядке возрастания.
* При flagUp = false: на каждой итерации внутреннего цикла элементы справа от right стоят в порядке убывания.

При каждом проходе внешнего цикла шейкерной сортировки с условием Айверсона происходит разделение массива на упорядоченную и неупорядоченную части. Внутренние циклы смещаются в сторону ограниченных границ упорядоченной части для каждого прохода, что гарантирует уменьшение зоны неупорядоченной части на каждой итерации.

Таким образом, каждый проход внешнего цикла уменьшает область, требующую сортировки, и устанавливает новые границы упорядоченной части, что приводит к росту уже отсортированной области массива.

В конечном итоге, упорядоченная область расширяется, а неупорядоченная область сокращается на каждом проходе внешнего цикла до тех пор, пока весь массив не будет упорядочен.

Таким образом, конечность циклов шейкерной сортировки с условием Айверсона обусловлена уменьшением области неупорядоченного массива и ростом упорядоченной части на каждой итерации внешнего цикла, что приводит к завершению сортировки после конечного числа итераций. Из этого следует, что все циклы шейкерной сортировки с условием Айверсона корректны.

### **2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма шейкерной сортировки с условием Айверсона**

Если массив отсортирован по возрастанию,то это будет являться лучшим случаем. Тогда количество операций сравнения и перемещения равно O(n).

Если массив заполнен случайными числами,то это будет являться средним случаем. Тогда количество операций сравнения и перемещения равно O(n2).

Если массив отсортирован по убыванию,то это будет являться худшим случаем. Тогда количество операций сравнения и перемещения равно O(n2).

Функция роста времени в лучшем случае равна O(n), а в худшем случае равна O(n2).

## **2.3 Реализация алгоритма на языке C++**

Для реализации алгоритма на языке C++ (рис.3,4) потребуются следующие библиотеки: iostream, random, chrono.



Рисунок 3 – Программа алгоритма шейкерной сортировки с условием Айверсона



Рисунок 4 – Функция main для алгоритма шейкерной сортировки

## **2.4 Проведение тестирования и построение графика**

### **2.4.1 Тестирование**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов. Продемонстрируем результаты тестирования программы от 100 до 1000000 в таблице 1.1, а n=10 продемонстрируем на рисунке 5.

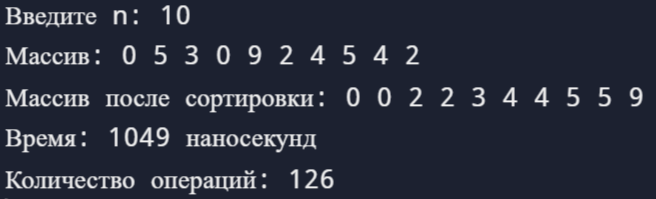


Рисунок 5 - Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.043 | 11954 |
| 1000 | 3.12 | 1209736 |
| 10000 | 322.414 | 119724514 |
| 100000 | 37230.638 | 12020156202 |
| 1000000 | 4182489.87 | 137929780704 |

### **2.4.2 Построение графика**

Протестировав программу и получив данные, мы можем построить график функции роста Тп алгоритма шейкерной сортировки с условием Айверсона в среднем случае от размера массива n (рис.6).

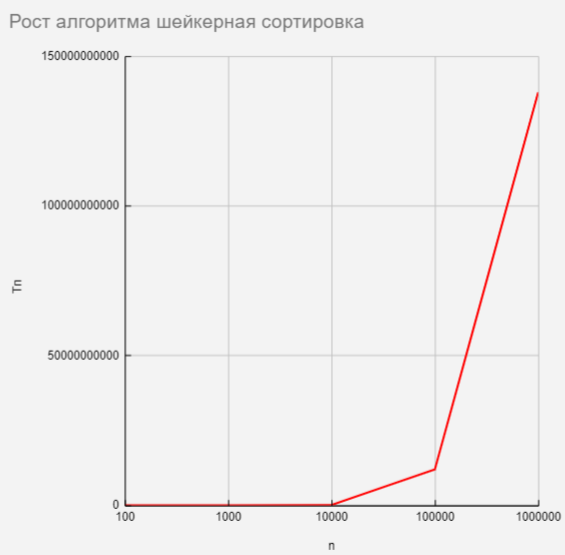


Рисунок 6 - График функции роста алгоритма шейкерной сортировки с условием Айверсона

## **2.5 Описание алгоритма быстрой сортировки(Хоара)**

### **2.5.1 Описание выполнения и блок-схема алгоритма быстрой сортировки (Хоара)**

Быстрая сортировка (или сортировка Хоара) - это эффективный алгоритм сортировки, который работает по принципу "разделяй и властвуй". Алгоритм состоит из двух основных шагов: разделения массива на две подгруппы и рекурсивной сортировки этих подгрупп.

Шаги выполнения быстрой сортировки (Хоара):

Выбирается опорный элемент из массива. Обычно выбирается средний элемент.

Массив разделяется на две подгруппы:

* элементы, меньшие опорного, помещаются слева от него
* элементы, большие опорного, помещаются справа от него

Рекурсивно применяется алгоритм к каждой из подгрупп.

Процесс повторяется до тех пор, пока размер подгруппы не станет равен 1 (базовый случай рекурсии). В этом случае массив считается отсортированным.

В результате этих шагов все элементы массива становятся на свои места в отсортированном порядке.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.7,8).

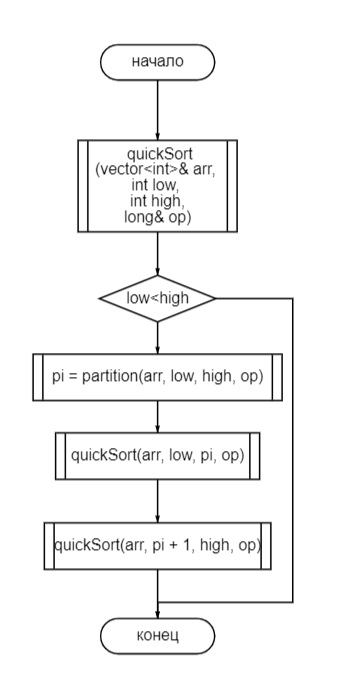


Рисунок 7 – Блок-схема алгоритма быстрой сортировки (Хоара)

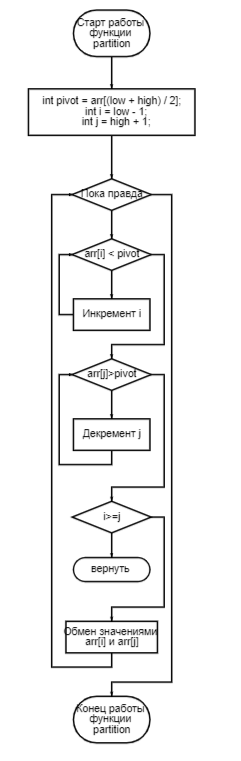


Рисунок 8 – Блок-схема алгоритма быстрой сортировки (Хоара)

### **2.5.2 Доказательство корректности циклов алгоритма быстрой сортировки (Хоара)**

Инвариант разделения: после разделения массива на подмассивы, элементы в левой части будут меньше или равны опорному элементу, а элементы в правой части будут больше или равны опорному элементу.

Инвариант перемещения опорного элемента: после завершения процесса разделения, опорный элемент будет занимать свое окончательное место в отсортированном массиве.

Инвариант разделения гарантирует, что на каждом шаге алгоритма размер подмассивов уменьшается, так как после разделения один из подмассивов содержит по крайней мере один элемент меньше опорного. Это приводит к тому, что на каждом шаге размер проблемы уменьшается, пока подмассивы не станут размером 1 (базовый случай рекурсии).

Инвариант перемещения опорного элемента гарантирует, что опорный элемент занимает окончательное место в отсортированном массиве, что означает, что для этого конкретного элемента алгоритм больше не будет выполняться.

Таким образом, завершение циклов быстрой сортировки (Хоара) обеспечивается уменьшением размера подмассивов на каждом шаге и наличием базового случая, когда размер подмассива становится равным 1, что гарантирует конечность циклов при сортировке.

Из доказательства можно сделать вывод, что все циклы данного алгоритма корректны.

### **2.5.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма быстрой сортировки (Хоара)**

Если массив отсортирован по возрастанию,то это будет являться лучшим случаем. Тогда количество операций сравнения и перемещения равно O(nlog2n).

Если массив заполнен случайными числами,то это будет являться средним случаем. Тогда количество операций сравнения и перемещения равно O(nlog2n).

Если массив отсортирован по убыванию,то это будет являться худшим случаем. Тогда количество операций сравнения и перемещения равно O(n2).

Функция роста времени в лучшем случае равна O(nlog2n), а в худшем случае равна O(n2).

## **2.6 Реализация алгоритма на языке C++, проведение тестирования и построение графика**

Для реализации алгоритма на языке C++ (рис.9,10) потребуются следующие библиотеки: iostream, random, chrono, vector.



Рисунок 9 – Программа алгоритма быстрой сортировки (Хоара)



Рисунок 10 – Функция main для алгоритма быстрой сортировки (Хоара)

## **2.7 Проведение тестирования и построение графика**

### **2.7.1 Тестирование**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов. Продемонстрируем результаты тестирования программы от 100 до 1000000 в таблице 1.1, а n=10 продемонстрируем на рисунке 11.

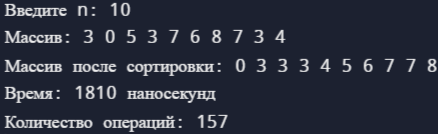


Рисунок 11 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.010668 | 2954 |
| 1000 | 0.197999 | 39476 |
| 10000 | 1.853812 | 511335 |
| 100000 | 24.560063 | 6321418 |
| 1000000 | 276.563814 | 748891551 |

### **2.7.2 Построение графика**

Протестировав программу и получив данные, мы можем построить график функции роста Тп алгоритма быстрой сортировки в среднем случае от размера массива n (рис.12).

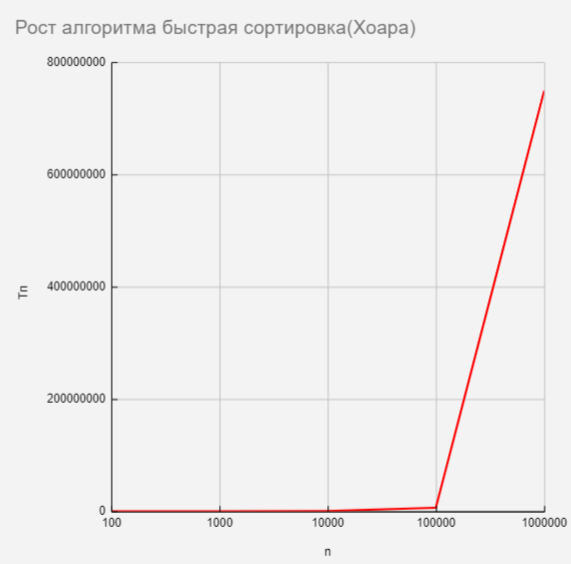


Рисунок 12 - График функции роста Тп алгоритма быстрой сортировки от размера массива n

## **2.8 Сортировка простым обменом**

Добавим из предыдущей работы таблицу результатов тестирования простой сортировки обменом в среднем случае(табл.1.3).

Таблица 1.3. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.038 | 12167 |
| 1000 | 2.97 | 1229394 |
| 10000 | 289.94 | 122272063 |
| 100000 | 36093.85 | 12251406490 |
| 1000000 | 5215467.81 | 1220495346463 |

## **2.9 Сравнительный график зависимости для трёх алгоритмов**

На основании данных из таблиц 1.1, 1.2 и 1.3 были построены графики функции роста времени выполнения алгоритмов шейкерной сортировки с условием Айверсона, быстрой сортировки(Хоара) и сортировки простым обменом в среднем случае в зависимости от размера массива n. Для наглядного сравнения были построены два графика: первый для значений до 1000 (рис.13) и второй для значений от 10000 до 1000000 (рис.14), чтобы обеспечить более точное сравнение.

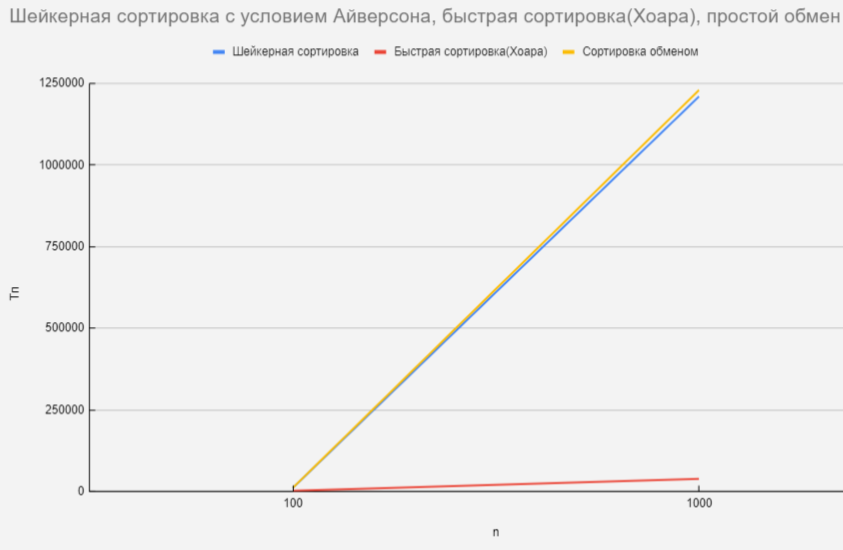


Рисунок 13 - График сравнения трёх сортировок в среднем случае при n до 1000

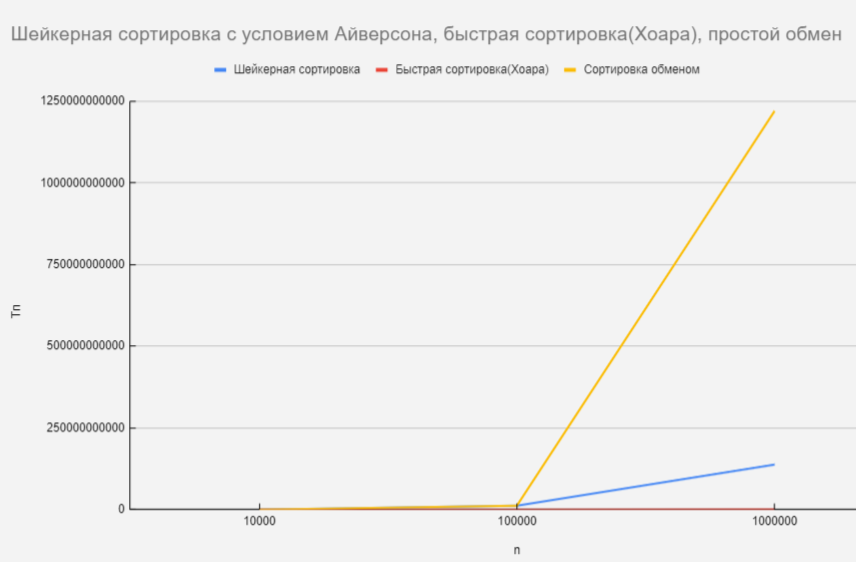


Рисунок 14 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

Анализ таблиц и графиков позволяет сделать вывод, что алгоритм сортировки простого обмена является наименее эффективным в среднем случае, шейкерная сортировка занимает второе место по эффективности, а быстрая сортировка (Хоара) демонстрирует наилучшую эффективность.

## **2.10 Тестирование программ для алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара)**

### **2.10.1 Тестирование с значениями упорядоченными по убыванию и построение графика для алгоритма шейкерной сортировки с условием Айверсона**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort и объект greater<int>() в main(рис.15,16). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем выполнение программы при n=10 (рис.17).

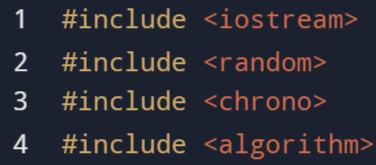


Рисунок 15 – Функция сортировки по убыванию



Рисунок 16 – Функция main

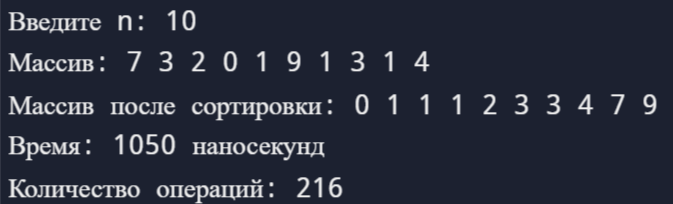


Рисунок 17 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Поскольку значения отсортированы в строго возрастающем порядке, можно сделать вывод о том, что это представляет собой худший случай сортировки, и, следовательно, сложность алгоритма равна O(n2). Таким образом, в лучшем случае алгоритм имеет квадратичную вычислительную сложность. Результаты тестирования будут занесены в таблицу 1.4.

Таблица 1.4. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.069 | 19212 |
| 1000 | 7.325 | 1901952 |
| 10000 | 474.079 | 190019258 |
| 100000 | 48011.604 | 19000255202 |
| 1000000 | 5377779.76 | 190000192254 |

На основе полученных данных, продемонстрированных в таблице 1.4, построим график функции роста Тп алгоритма шейкерной сортировки с условием Айверсона с отсортированными значениями по убыванию от размера массива n (рис.18).

### 

Рисунок 18 - График функции роста Тп шейкерная алгоритма сортировки с отсортированными значениями по убыванию от размера массива n

### **2.10.2 Тестирование с значениями упорядоченными по возрастанию и построение графика для алгоритма шейкерной сортировки с условием Айверсона**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort в main(рис.19,20). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем выполнение программы при n=10 (рис.21).

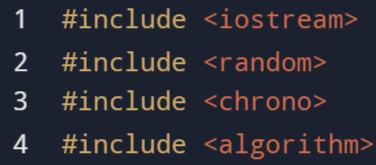


Рисунок 19 – Используемые библиотеки



Рисунок 20 – Функция main

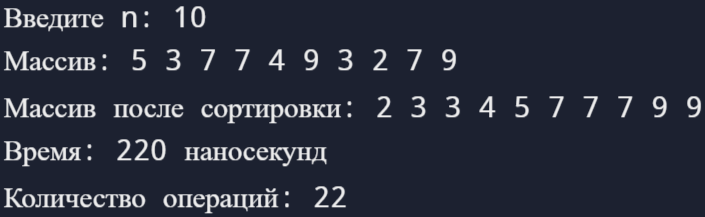


Рисунок 21 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Поскольку значения отсортированы в строго возрастающем порядке, можно сделать вывод о том, что это представляет собой лучший случай сортировки, и, следовательно, сложность алгоритма равна O(n). Таким образом, в лучшем случае алгоритм имеет линейную вычислительную сложность. Результаты тестирования будут занесены в таблицу 1.5.

Таблица 1.5. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп=Cп+Mп** |
| --- | --- | --- |
| 100 | 0.000599 | 202 |
| 1000 | 0.002944 | 2002 |
| 10000 | 0.037969 | 20002 |
| 100000 | 0.323817 | 200002 |
| 1000000 | 4.122395 | 2000002 |

На основе полученных данных, продемонстрированных в таблице 1.5, построим график функции роста Тп этого алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.22).

### 

Рисунок 22 - График функции роста Тп алгоритма шейкерной сортировки с условием Айверсона с упорядоченными значениями по возрастанию от размера массива n

### **2.10.3 Тестирование с значениями упорядоченном по убыванию и построение графика для алгоритма быстрой сортировки (Хоара)**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort и объект greater<int>() в main(рис.23,24). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем выполнение программы при n=10 (рис.25).

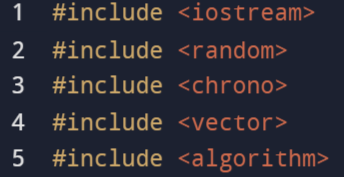


Рисунок 23 – Использованные библиотеки



Рисунок 24 – Функция main

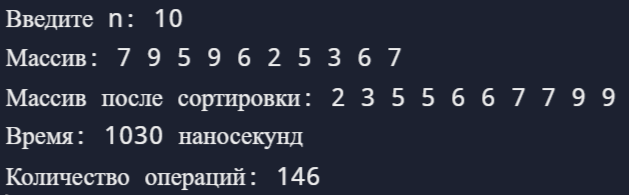


Рисунок 25 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Поскольку значения отсортированы в строго убывающем порядке, можно сделать вывод о том, что это представляет собой худший случай сортировки, и, следовательно, сложность алгоритма равна O(n2). Таким образом, в лучшем случае алгоритм имеет линейную вычислительную сложность. Результаты тестирования будут занесены в таблицу 1.6.

Таблица 1.6. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.01 | 2632 |
| 1000 | 0.09 | 38080 |
| 10000 | 2.097 | 505536 |
| 100000 | 13.989 | 6129351 |
| 1000000 | 173.677 | 73153492 |

На основе полученных данных, продемонстрированных в таблице 1.6, построим график функции роста Тп этого алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.26).

### 

Рисунок 26 - График функции роста Тп алгоритма быстрой сортировки (Хоара) с отсортированными значениями по убыванию от размера массива n

### **2.10.4 Тестирование с значениями упорядоченном по возрастанию и построение графика для алгоритма быстрой сортировки (Хоара)**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке возрастанию. Применим функцию sort в main(рис.27,28). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем выполнение программы при n=10 (рис.29).

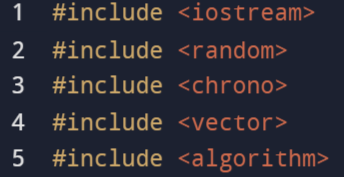


Рисунок 27 – Использованные библиотеки



Рисунок 28 – Функция main

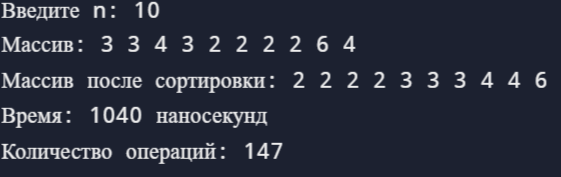


Рисунок 29 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Поскольку значения отсортированы в строго возрастающем порядке, можно сделать вывод о том, что это представляет собой лучший случай сортировки, и, следовательно, сложность алгоритма равна O(nlog2n). Таким образом, в лучшем случае алгоритм имеет линейную вычислительную сложность. Результаты тестирования будут занесены в таблицу 1.7.

Таблица 1.7. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.011 | 2490 |
| 1000 | 0.101 | 36785 |
| 10000 | 1.137 | 491861 |
| 100000 | 13.821 | 5991903 |
| 1000000 | 626.276 | 71753228 |

На основе полученных данных, продемонстрированных в таблице 1.7, построим график функции роста Тп этого алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.30).

### 

Рисунок 30 - График функции роста Тп алгоритма быстрой сортировки (Хоара) с отсортированными значениями по возрастанию от размера массива n

## **2.11 Вывод**

Результаты тестирования и графики указывают на зависимость эффективности шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара) от изначального упорядочения массива. Шейкерная сортировка с условием Айверсона показывает примерно одинаковую производительность как на упорядоченных, так и на перевернутых массивах, что свидетельствует о ее стабильности и эффективности в различных сценариях. В то же время, быстрая сортировка (Хоара) демонстрирует более заметную зависимость от изначального упорядочения массива. Наилучшие результаты достигаются на случайных массивах, в то время как на упорядоченных или перевернутых массивах ее производительность снижается.

Следовательно, можно сделать вывод о том, что шейкерная сортировка с условием Айверсона менее зависима от начального упорядочения массива, в то время как быстрая сортировка (Хоара) более чувствительна к упорядоченности данных.

# 

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи**

1. Из материалов предыдущей практической работы приведите в отчёте формулы Тт(n) функций роста алгоритма простой сортировки в лучшем и худшем случае.

2. На основе определений соответствующих нотаций получите асимптотическую оценку вычислительной сложности простого алгоритма сортировки обменом:

- в О-нотации (оценка сверху) для анализа худшего случая;

- в Ω-нотации (оценка снизу) для анализа лучшего случая.

3. Получите (если это возможно) асимптотически точную оценку вычислительной сложности алгоритма в нотации θ.

4. Реализуйте графическое представление функции роста и полученных асимптотических оценок сверху и снизу.

5. Привести справочную информацию о вычислительной сложности алгоритмов шейкерной сортировки и быстрой сортировки (Хоара).

6. Общие результаты свести в табл. 2.

7. Сделать вывод о наиболее эффективном алгоритме из трёх.

## **3.2 Формулы Тт(n) для сортировки обменом**

Лучший случай для сортировки обменом равен Тт(n)=n.

Средний случай для сортировки обменом равен Тт(n)=(n2-n)/2.

Худший случай для сортировки обменом равен Тт(n)=3(n2-n)/2.

Для простой сортировки обменом равен ёмкостная сложность O(1).

## **3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом**

Асимптотическая оценка в худшем случае равна О(n2).

Асимптотическая оценка в лучшем случае равна Ω(n).

Асимптотическая оценка в среднем случае равна θ(n2).

## **3.4 График функции роста**

Построим график на данных, полученных в ходе анализа простой сортировки обменом(рис.31).

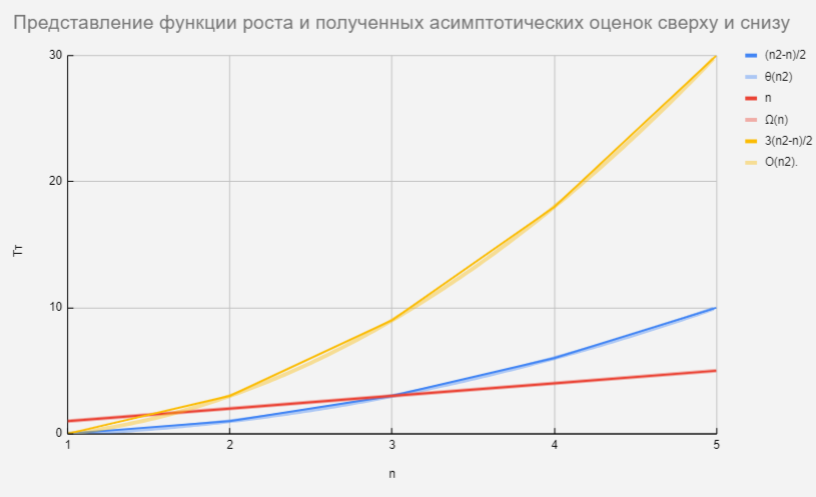


Рисунок 31 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки с условием Айверсона**

Асимптотическая оценка в худшем случае равна О(n2).

Асимптотическая оценка в лучшем случае равна Ω(n).

Асимптотическая оценка в среднем случае равна θ(n2).

Для шейкерной сортировки с условием Айверсона ёмкостная сложность алгоритма равна O(1).

## **3.6 Справочная информация о вычислительной сложности алгоритма быстрой сортировки(Хоара)**

Асимптотическая оценка в худшем случае равна О(n2).

Асимптотическая оценка в лучшем случае равна Ω(nlog2n).

Асимптотическая оценка в среднем случае равна θ(nlog2n).

Для быстрой сортировки(Хоара) ёмкостная сложность алгоритма равна O(log2n).

## **3.7 Таблица асимптотической сложности трёх алгоритмов**

Заполним таблицу 2 на основе данных, полученных в ходе анализа алгоритмов сортировки простого обмена, быстрой сортировки(Хоара) и шейкерной сортировки с условием Айверсона.

Таблица 2. Сводная таблица результатов

| Алгоритм | Асимптотическая сложность алгоритма | | | |
| --- | --- | --- | --- | --- |
| Наихудший случай (сверху) | Наилучший случай (снизу) | Средний случай (точная оценка) | Ёмкостная сложность |
| Простой обмен | О(n2) | Ω(n) | θ(n2) | О(1) |
| Шейкерная сортировка с условием Айверсона | О(n2) | Ω(n) | θ(n2) | О(1) |
| Быстрая сортировка (Хоара) | О(n2) | Ω(nlog2n) | θ(nlog2n) | О(log2n) |

**3.8 Вывод**

Сделаем вывод, что самым эффективным алгоритмом сортировки будет быстрая сортировка(Хоара). Данный вывод можно сделать, опираясь на графики сравнения, результаты тестирования и асимптотическую вычислительную сложность. Быстрая сортировка(Хоара) обладает лучшей ёмкостной сложностью из возможных, а именно O(log2n). Но несмотря на все свои плюсы, быстрая сортировка(Хоара) в худшем случае обладает сложностью O(n2), что сопоставляет сложность всех трёх алгоритмов.

# 

# 4 ВЫВОДЫ

В ходе практической работы были выполнены следующие задачи:

- Получены навыки по анализу вычислительной сложности алгоритмов сортировки и определению наиболее эффективного алгоритма;

- Проведён анализ алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Были реализованы программы алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Проведено тестирование программ для алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Построены графики функции роста Тп алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Произведено сравнение алгоритмов простой сортировки обменом алгоритмов шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Проведен анализ асимптотической сложности алгоритмов сортировки простым обменом, шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Произведено сравнение асимптотической сложности алгоритмов сортировки простым обменом, шейкерной сортировки с условием Айверсона и быстрой сортировки (Хоара);

- Проведено определение наиболее эффективного алгоритма.

Таким образом, главную цель практической работы, а именно получение навыков по анализу вычислительной сложности алгоритмов сортировки и определению наиболее эффективного алгоритма, можно считать выполненной.

# 5 ЛИТЕРАТУРА

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).

12. НОУ ИНТУИТ | Технопарк Mail.ru Group: Алгоритмы и структуры данных [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3496/738/info (дата обращения 15.03.2022).