|  |
| --- |
|  |
| 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: Кагарманов ВА. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_gjdgxs)

[2 ЗАДАНИЕ №1 5](#_30j0zll)

[2.1 Формулировка задачи 5](#_1fob9te)

[2.2 Математическая модель решения алгоритма 6](#_2et92p0)

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

[2.2.2 Доказательство корректности циклов алгоритма шейкерной сортировки 9](#_3dy6vkm)

[2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма шейкерной сортировки 10](#_1t3h5sf)

2.3 Реализация алгоритма на языке C++, проведение тестирования и построение графика 11

[2.3.1 Реализация алгоритма шейкерной сортировки на языке C++ 11](#_4d34og8)

[2.3.2 Тестирование и построение графика 12](#_2s8eyo1)

[2.4 Математическая модель решения алгоритма 14](#_17dp8vu)

[2.4.1 Описание выполнения и блок-схема алгоритма быстрой сортировки(Хоара) 14](#_3rdcrjn)

[2.4.2 Доказательство корректности циклов алгоритма быстрой сортировки(Хоара) 16](#_26in1rg)

[2.4.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма быстрой сортировки(Хоара) 17](#_lnxbz9)

[2.5 Реализация алгоритма на языке C++, проведение тестирования и построение графика 18](#_35nkun2)

[2.5.1 Реализация алгоритма быстрой сортировки(Хоара) на языке C++ 18](#_1ksv4uv)

[2.5.2 Тестирование и построение графика 19](#_44sinio)

[2.6 Сортировка простым обменом 21](#_2jxsxqh)

[2.7 Сравнение трёх алгоритмов на графике 22](#_z337ya)

[2.8 Тестирование программ для алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 23](#_3j2qqm3)

[2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма шейкерной сортировки 24](#_1y810tw)

2.8.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма шейкерной сортировки 26

[2.8.3 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма быстрой сортировки(Хоара) 29](#_4i7ojhp)

2.8.4 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма быстрой сортировки(Хоара) 31

[2.9 Вывод по заданию №1 34](#_1ci93xb)

[3 ЗАДАНИЕ №2 36](#_2bn6wsx)

[3.1 Формулировка задачи 36](#_qsh70q)

[3.2 Формулы функции роста алгоритма сортировки простым обменом в худшем и лучшем случае 36](#_3as4poj)

3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом 37

[3.4 Графическое представление функции роста и полученных асимптотических оценок сверху и снизу 37](#_1pxezwc)

[3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 38](#_49x2ik5)

[3.5.1 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки 38](#_2p2csry)

[3.5.2 Справочная информация о вычислительной сложности алгоритма быстрой сортировки(Хоара) 39](#_147n2zr)

[3.6 Таблица асимптотической сложности трёх алгоритмов 39](#_3o7alnk)

[4 КОНТРОЛЬНЫЕ ВОПРОСЫ 41](#_23ckvvd)

[5 ВЫВОДЫ 47](#_ihv636)

[6 ЛИТЕРАТУРА 48](#_32hioqz)

# 

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи (Вариант 14, в списке 14)**

Эмпирическая оценка эффективности алгоритмов.

1. Разработать алгоритм Шелла со смещениями Седжвика, 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 Шелла со смещениями Седжвика.

3. Разработать алгоритм быстрой сортировки (Хоара), 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 (Хоара).

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 ускоренной 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 Математическая модель решения алгоритма**

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

Сортировка Шелла является довольно интересной модификацией алгоритма сортировки простыми вставками.

Рассмотрим следующий алгоритм сортировки массива a[0].. a[15].

1. Вначале сортируем простыми вставками каждые 8 групп из 2-х элементов (a[0], a[8[), (a[1], a[9]), ... , (a[7], a[15]).

2. Потом сортируем каждую из четырех групп по 4 элемента (a[0], a[4], a[8], a[12]), ..., (a[3], a[7], a[11], a[15]).

В нулевой группе будут элементы 4, 12, 13, 18, в первой - 3, 5, 8, 9 и т.п.

3. Далее сортируем 2 группы по 8 элементов, начиная с (a[0], a[2], a[4], a[6], a[8], a[10], a[12], a[14]).

4. В конце сортируем вставками все 16 элементов.

Единственной характеристикой сортировки Шелла является *приращение* - расстояние между сортируемыми элементами, в зависимости от прохода. В конце приращение всегда равно единице - метод завершается обычной сортировкой вставками, но именно последовательность приращений определяет рост эффективности.

Использованный в примере набор ..., 8, 4, 2, 1 - неплохой выбор, особенно, когда количество элементов - степень двойки. Однако гораздо лучший вариант предложил Р.Седжвик. Его последовательность имеет вид



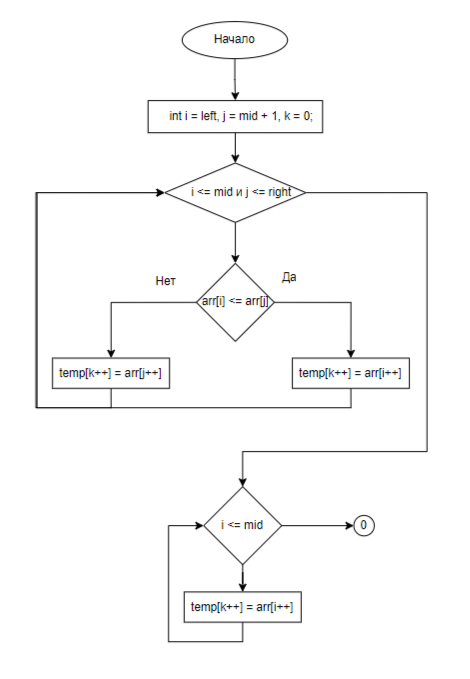


Рисунок 1 – Блок-схема алгоритма сортировки Шелла

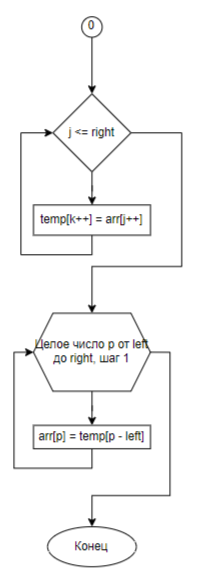


Рисунок 2 – Блок-схема алгоритма сортировки Шелла

### **2.2.2 Доказательство корректности циклов алгоритма шейкерной сортировки**

Инвариант для цикла: значение переменной i всегда меньше или равна mid и j всегда меньше или равна right.

Инвариант для цикла: значение переменной i всегда меньше или равна mid.

Инвариант для цикла: значение переменной j всегда меньше или равна right.

Инвариант для цикла: значение переменной p всегда меньше right.

Докажем конечность циклов. Предположим, что наш алгоритм сортировки Шелла с смещением Седжвика успешно сортирует массив длиной N на каждой итерации. На каждой итерации мы делим массив на подмассивы заданной длины (смещения), сортируем их, а затем сужаем это смещение до 1 для финальной сортировки. Сначала сортируем элементы, которые находятся на расстоянии смещения друг от друга. Это поможет уменьшить количество инверсий в массиве и улучшит порядок элементов. Сужаем смещение до 1 и выполняем финальную сортировку вставками. После этого все элементы массива должны быть отсортированы.

Таким образом, циклы сортировки Шелла со смещением Седжвика должны быть корректными, поскольку процесс деления массива на подмассивы и их последующей сортировки на каждой итерации приводит к правильной укладке элементов в итоговом отсортированном массиве.

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

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

Лучший случай: лучший случай будет, если исходный массив уже почти отсортирован. В этом случае на каждой итерации смещение будет равно 1, и алгоритм будет завершаться после финальной сортировки вставками за время O(n), где n - размер массива.

Средний случай: при использовании смещения Седжвика алгоритм сортировки Шелла обычно достигает хороших результатов на разного вида данных. Сложность среднего случая зависит от выбранной последовательности смещений перед сужением до 1. Обычно сложность времени работы алгоритма в среднем случае составляет примерно O(n log n).

Худший случай: худший случай происходит, когда последовательность элементов в массиве распределена в таком порядке, который делает процесс сортировки максимально затратным. Например, если исходный массив отсортирован в обратном порядке, то сложность худшего случая может быть близка к O(n2).

Для данного метода сортировки, время исполнения в худшем случае увеличивается квадратично с ростом размера входного массива. Следовательно, можно использовать квадратичную функцию для описания функции роста данного сортировочного метода. Время исполнения в лучшем случае увеличивается линейно с ростом размера входного массива.

Ёмкостная сложность алгоритма будет равна O(1).

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

### **2.3.1 Реализация алгоритма шейкерной сортировки на языке C++**

Для реализации алгоритма сортировки Шелла с использованием смещения Седжвика на языке C++, понадобятся следующие библиотеки: iostream, random, chrono. vector и algorithm. iostream используется для работы с вводом-выводом в C++, random - для генерации случайных чисел в определенном диапазоне, chrono - для работы с интервалами времени и таймерами, vector – для создания удобных динамических массивов, а algorithm для выполнения алгоритмических операций над контейнерами и над другими последовательностями.



Рисунок 5 – Программа алгоритма Шелла со смещением Седжвика



Рисунок 6 – Функция main для алгоритма Шелла со смещением Седжвика

### **2.3.2 Тестирование и построение графика**

Задача программы - протестировать алгоритм сортировки на разных размерах массивов: n=10(рис.7), 100, 1000, 10000, 100000, 1000000. Тестирование проводится с помощью случайной генерации чисел. Результаты тестирования для размеров массивов от 100 до 1000000 будут отображены в таблице 1.1. Отобразим данные таблицы 1.1 на графике рисунок 8.

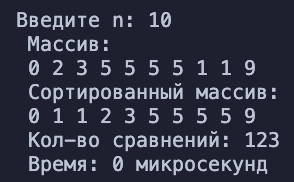


Рисунок 7 - Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.010 | 2476 |
| 1000 | 0.112 | 36182 |
| 10000 | 1.270 | 484826 |
| 100000 | 15.289 | 5968200 |
| 1000000 | 223.139 | 70738750 |

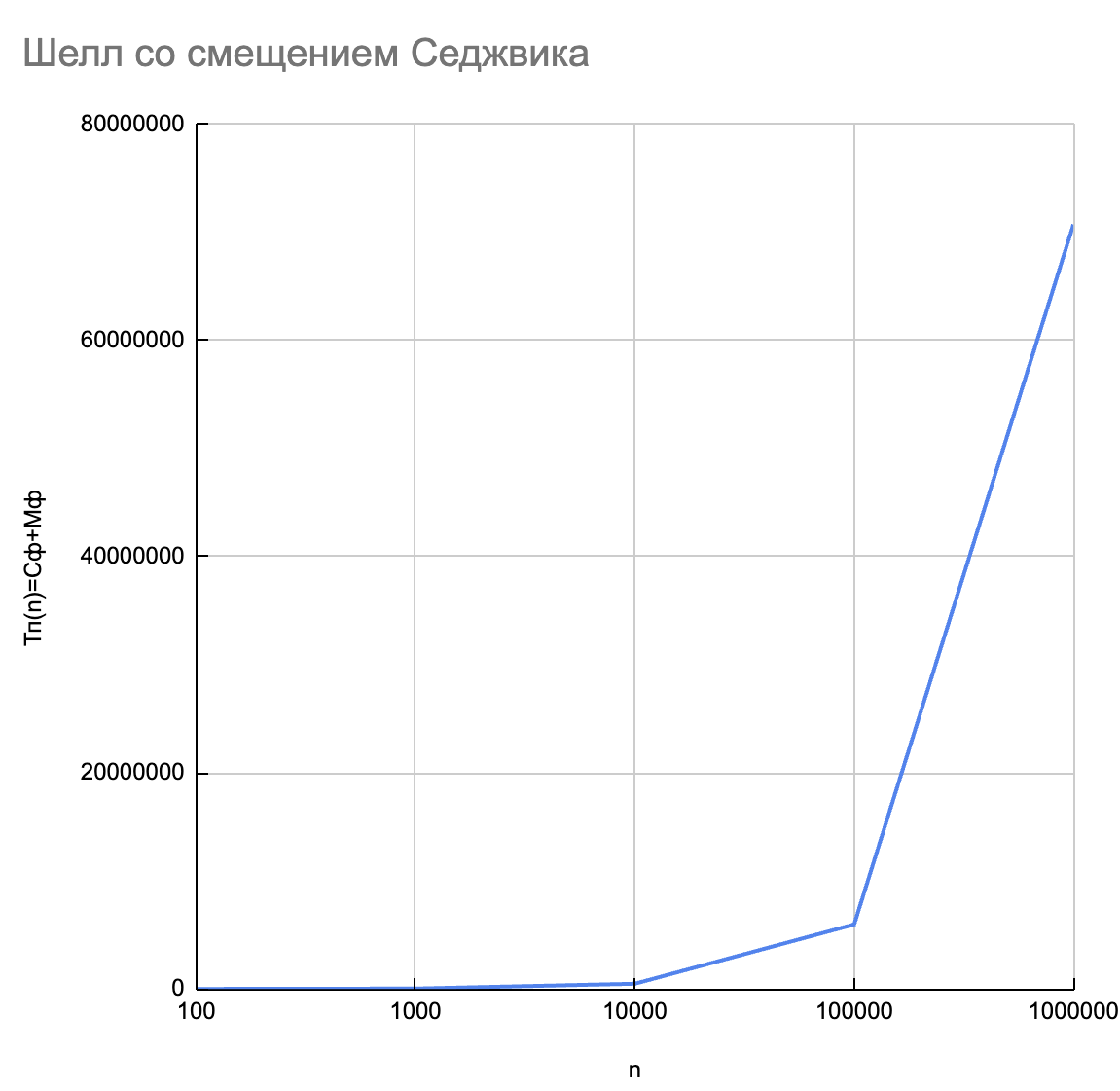


Рисунок 8 - График функции роста Тп алгоритма сортировки Шелла со смещение Седжвика от размера массива n

## **2.4 Математическая модель решения алгоритма**

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

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «[Пузырьковая сортировка](https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)» и «[Шейкерная сортировка](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D0%B9%D0%BA%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Общая идея алгоритма состоит в следующем:

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.9).

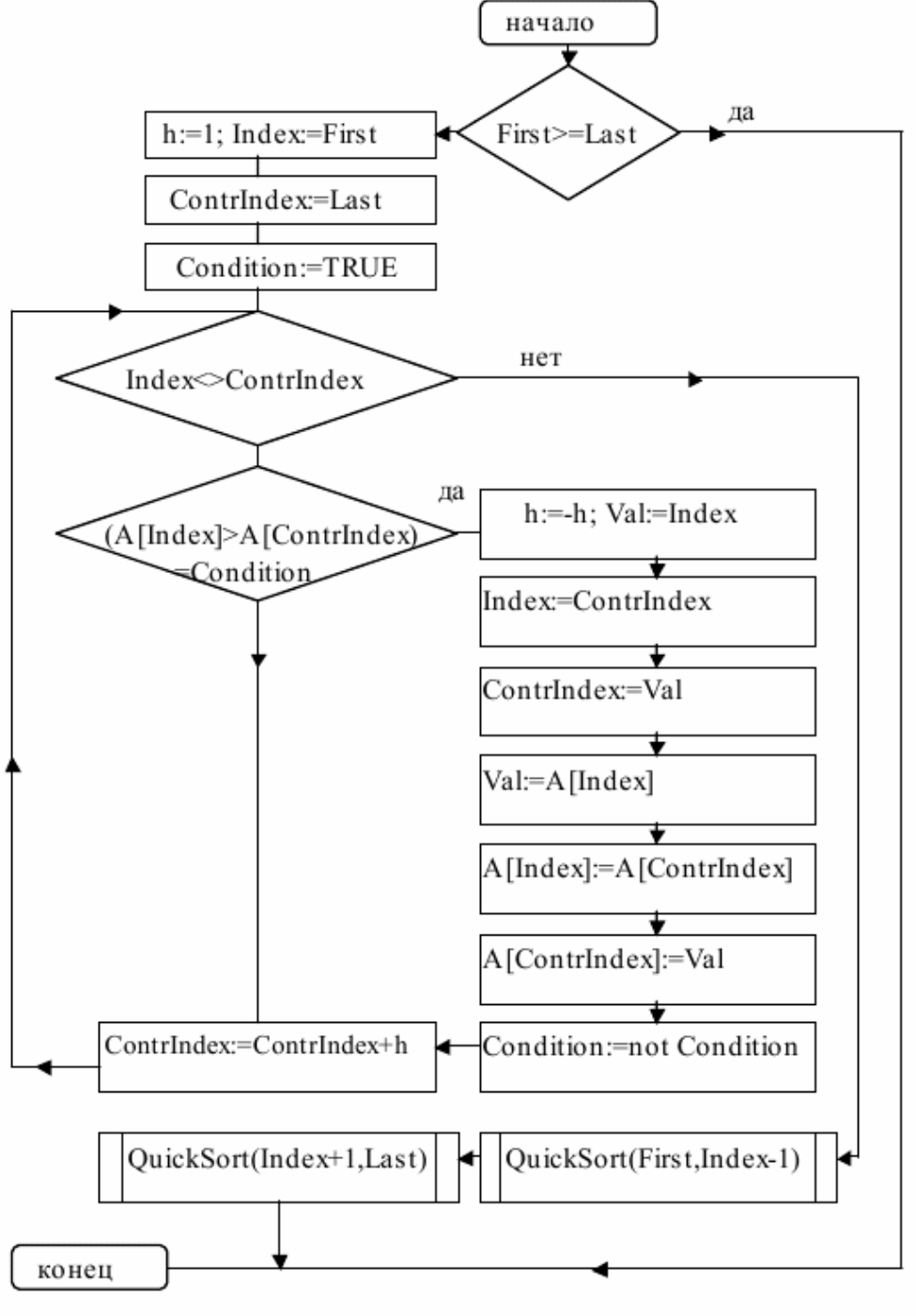


Рисунок 9 – Блок-схема алгоритма быстрой сортировки (Хоара)

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

Инвариант для внешнего цикла: значение переменной i меньше или равно j.

Инвариант для внутреннего цикла: значение переменной A[i] всегда меньше 0.

Инвариант для внутреннего цикла: значение переменной A[j] всегда больше o.

Доказательство корректности циклов алгоритма следующее: На каждой итерации внешнего цикла или внутренних циклов один из индексов (i или j) изменяется, приближая их друг к другу в пределах исходного диапазона. Изначальный диапазон индексов уменьшается с каждой итерацией, что приводит к завершению циклов при условии, что начальный диапазон был корректен (start < end). Рекурсивные вызовы также гарантируют уменьшение диапазона и, в конечном итоге, завершение алгоритма. Таким образом, все циклы в алгоритме являются корректными.

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

a. В наилучшем случае — когда массив уже отсортирован, количество операций сравнения и перемещения минимально и составляет O(n log n). В среднем случае — когда массив заполнен случайными числами, сложность алгоритма также будет O(n log n). В наихудшем случае — когда массив отсортирован в обратном порядке, количество операций также будет O(n2).

b. Функции роста времени: В наилучшем случае: O(n log n). В наихудшем случае: O(n2). Для данного метода сортировки время выполнения в наихудшем случае увеличивается квадратично по мере увеличения размера входного массива. Следовательно, можно использовать квадратичную функцию для описания роста времени выполнения данного метода сортировки. Время выполнения в наилучшем случае увеличивается практически линейно по размеру входного массива.

Ёмкостная сложность алгоритма будет равна O(log n).

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

### **2.5.1 Реализация алгоритма быстрой сортировки (Хоара) на языке C++**

Для реализации алгоритма быстрой сортировки(Хоара) на языке C++, понадобятся следующие библиотеки: iostream, random, chrono и algorithm. iostream используется для работы с вводом-выводом в C++, random - для генерации случайных чисел в определенном диапазоне, chrono - для работы с интервалами времени и таймерами, а algorithm для выполнения алгоритмических операций над контейнерами и над другими последовательностями.



Рисунок 13 – Программа алгоритма быстрой сортировки (Хоара)



Рисунок 14 – Функция main для алгоритма быстрой сортировки (Хоара)

### **2.5.2 Тестирование и построение графика**

Задача программы - протестировать алгоритм сортировки на разных размерах массивов: n=10, 100, 1000, 10000, 100000, 1000000. Тестирование проводится с помощью случайной генерации чисел. Результаты тестирования для размеров массивов от 100 до 1000000 будут отображены в таблице 1.2. Отобразим данные таблицы 1.2 на графике рисунок 16.

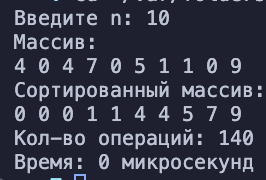


Рисунок 15 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.011 | 2043 |
| 1000 | 0.096 | 30176 |
| 10000 | 1.027 | 360721 |
| 100000 | 11.556 | 4495607 |
| 1000000 | 187.075 | 53918850 |

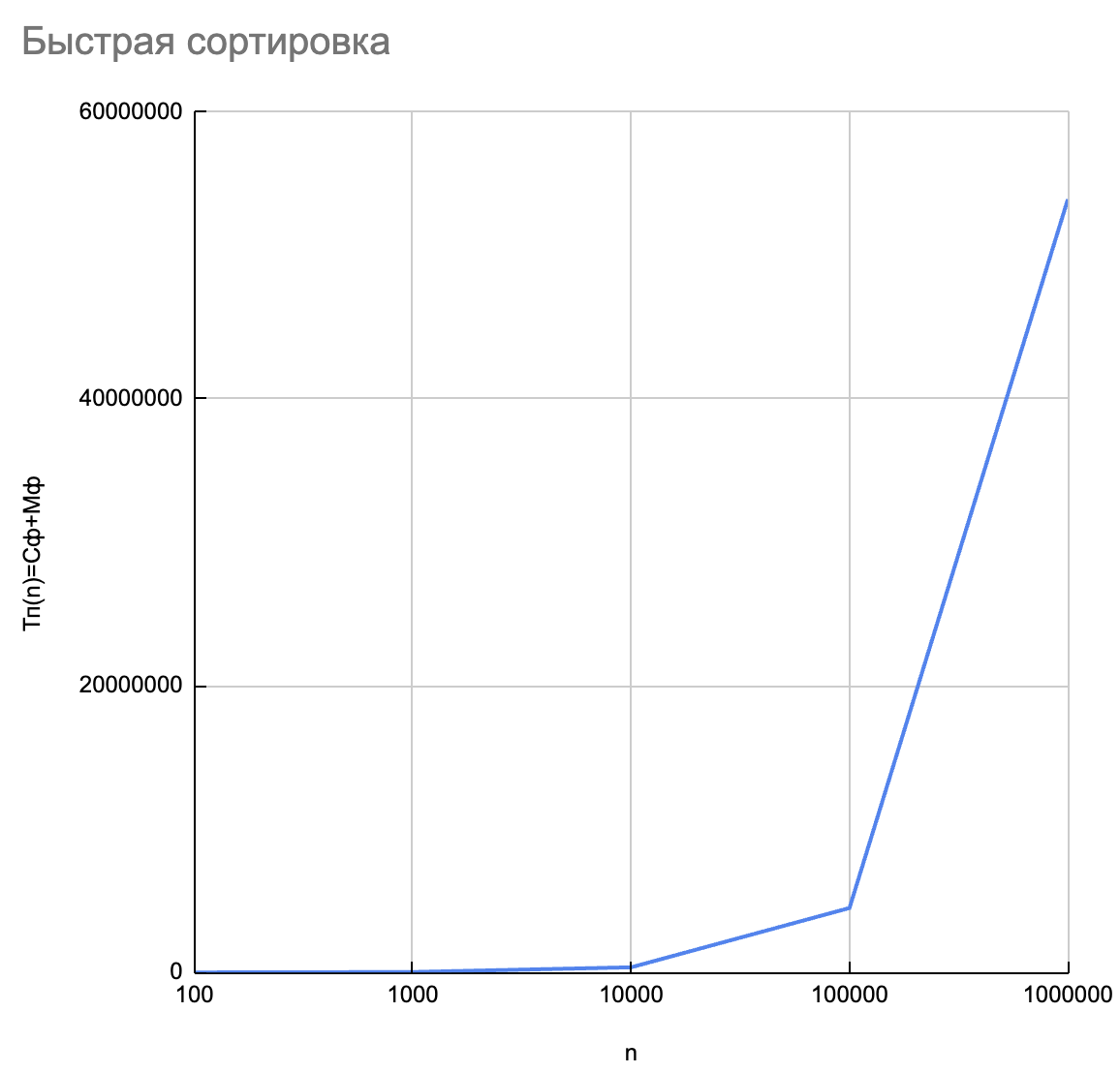


Рисунок 16 - График функции роста Тп алгоритма быстрой сортировки от размера массива n

## **2.6 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 | 0.07 | 12351 |
| 1000 | 2.819 | 1222976 |
| 10000 | 444.271 | 122480570 |
| 100000 | 48554.832 | 12235451560 |
| 1000000 | 6331550.09 | 122249302750 |

## **2.7 Сравнение трёх алгоритмов на графике**

На основе полученных данных, продемонстрированных в таблицах 1.1, 1.2 и 1.3, построим график функции роста Тп алгоритма сортировки Шелла со смещением по Седжвику, быстрой сортировки (Хоара) и сортировки простым обменом в среднем случае от размера массива n. Для наглядности сравнения построим два графика. Первый будет построен на значениях до 1000(рис.17), а второй от 10000 и до 1000000(рис.18). Это позволит нам сделать более точное сравнение.

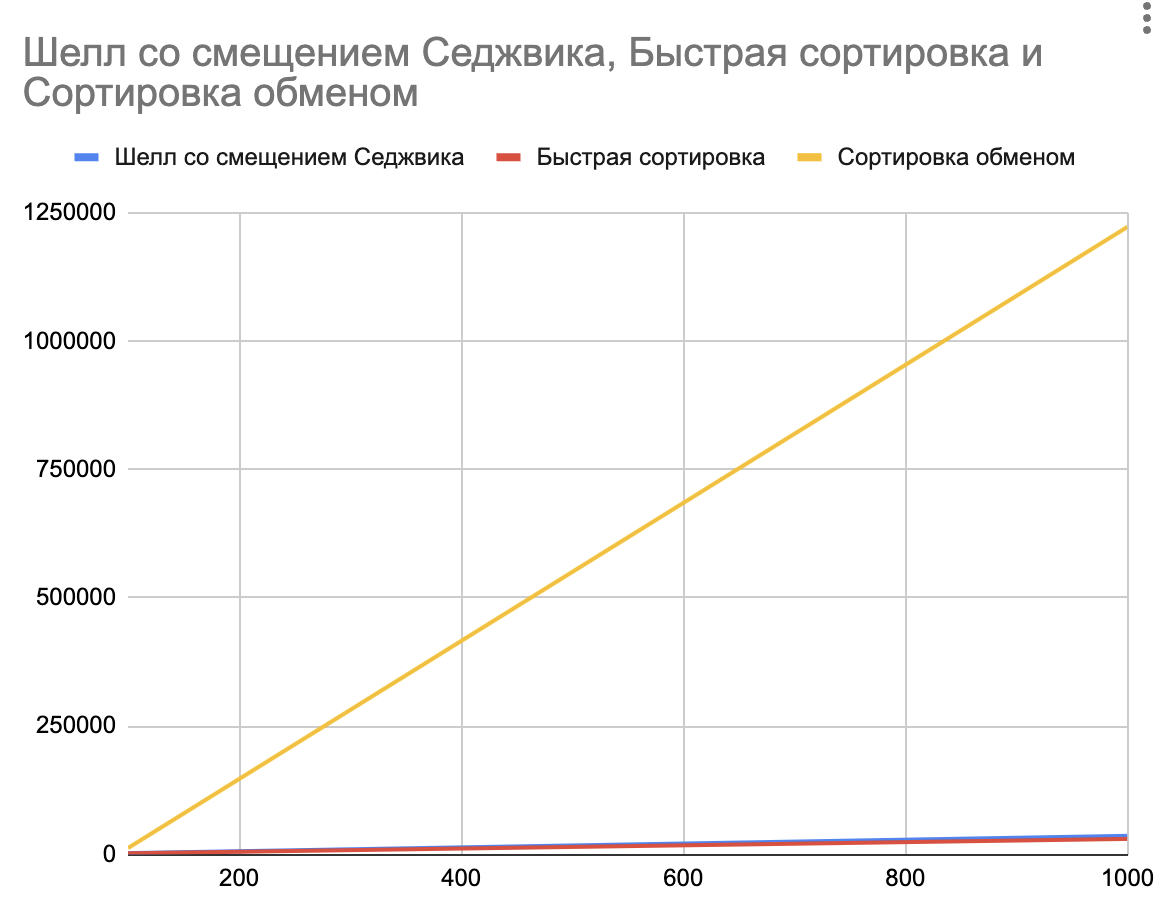
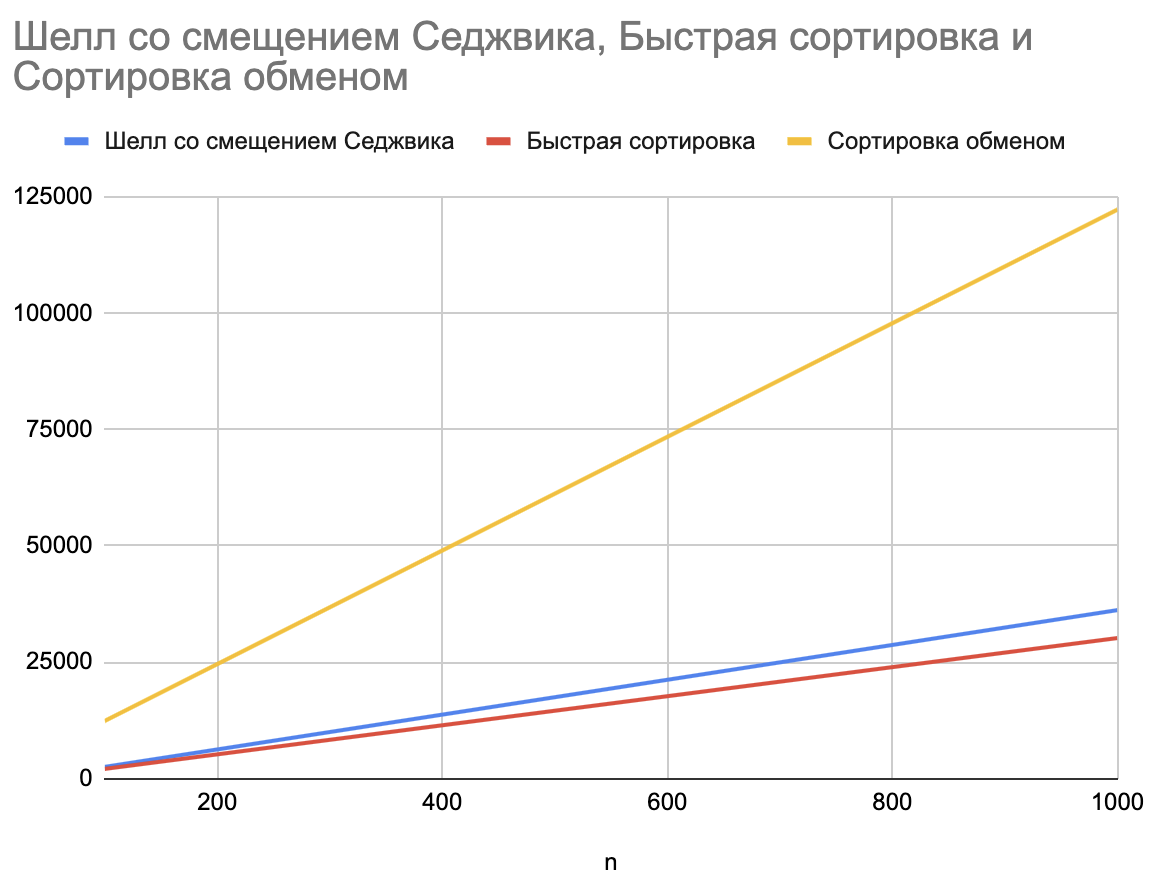


Рисунок 17 - График сравнения трёх сортировок в среднем случае при n до 1000

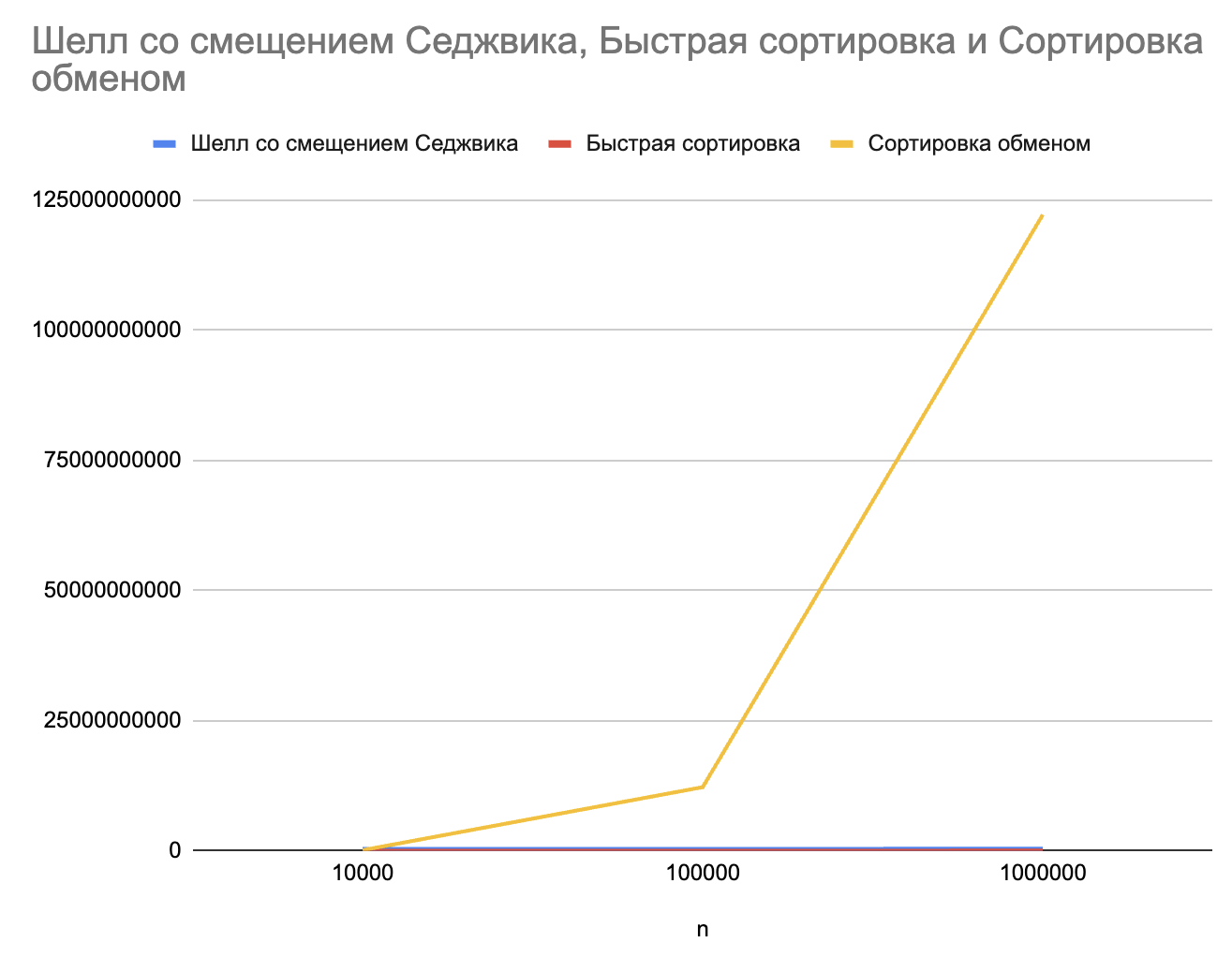


Рисунок 18 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

На основе таблиц 1.1, 1.2 и 1.3 и графиков(рис.7,8), можно сделать вывод, что в среднем случае алгоритм сортировки простого обмена самый неэффективный, алгоритм сортировки Шелла со смещением по Седжвику второй по эффективности, а алгоритм быстрой сортировки (Хоара) самый эффективный.

## **2.8 Тестирование программ для алгоритмов шейкерной сортировки и быстрой сортировки (Хоара)**

### **2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма Шелла со смещением Седжвика**

Программа будет протестирована на массивах с элементами, упорядоченными в строго убывающем порядке для значений n = 100, 1000, 10000, 100000 и 1000000. Продемонстрируем работу программы при n = 10. Алгоритм Шелла со смещением Седжвика останется без изменений, как показано на рисунке 5. На основе данных, представленных в таблице 1.4, мы построим график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по убыванию.

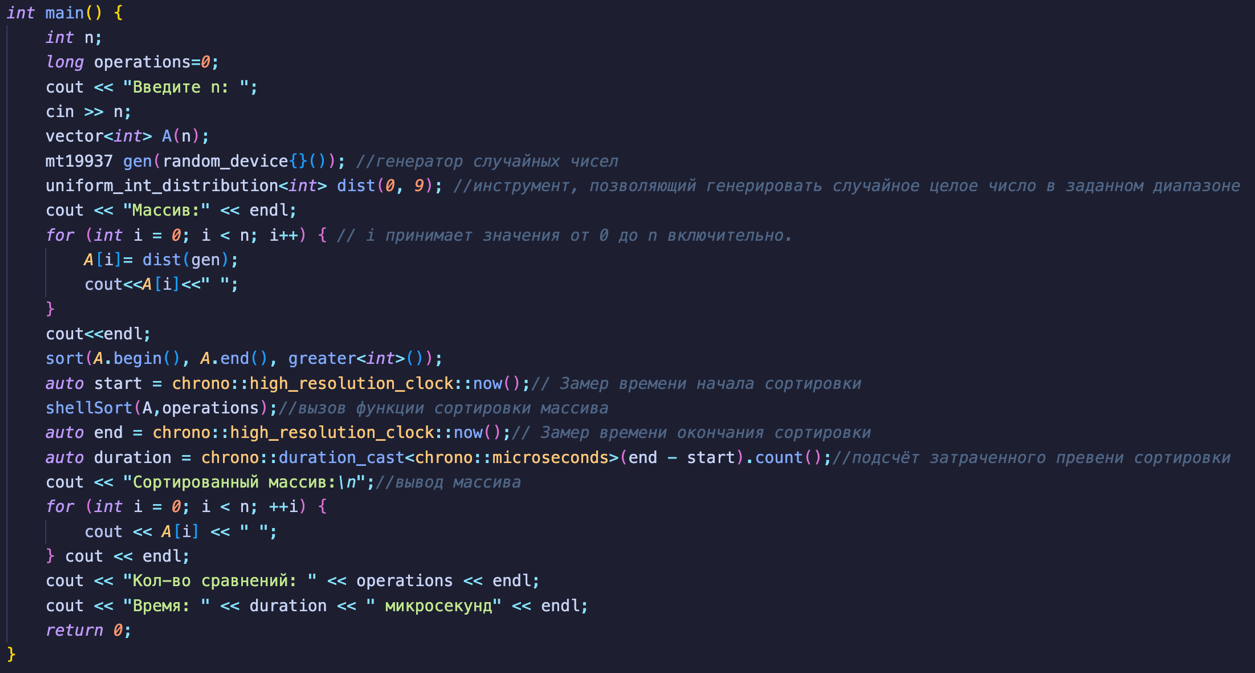


Рисунок 20 – Функция main при n=10 и с отсортированными значениями по убыванию

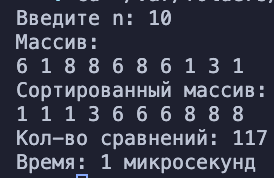


Рисунок 21 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Таблица 1.4. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.006 | 2149 |
| 1000 | 0.075 | 32720 |
| 10000 | 0. 742 | 444863 |
| 100000 | 10.752 | 5174529 |
| 1000000 | 132.324 | 62944219 |

### 

Рисунок 22 - График функции роста Тп Шелла со смещением Седжвика сортировки с отсортированными значениями по убыванию от размера массива n

### **2.8.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма Шелла со смещением Седжвика**

Программа будет протестирована на массивах с элементами, упорядоченными в строго возрастающем порядке для значений n = 100, 1000, 10000, 100000 и 1000000. Продемонстрируем работу программы при n = 10. Алгоритм Шелла со смещением Седжвика останется без изменений, как показано на рисунке 5. На основе данных, представленных в таблице 1.5, мы построим график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по возрастанию.

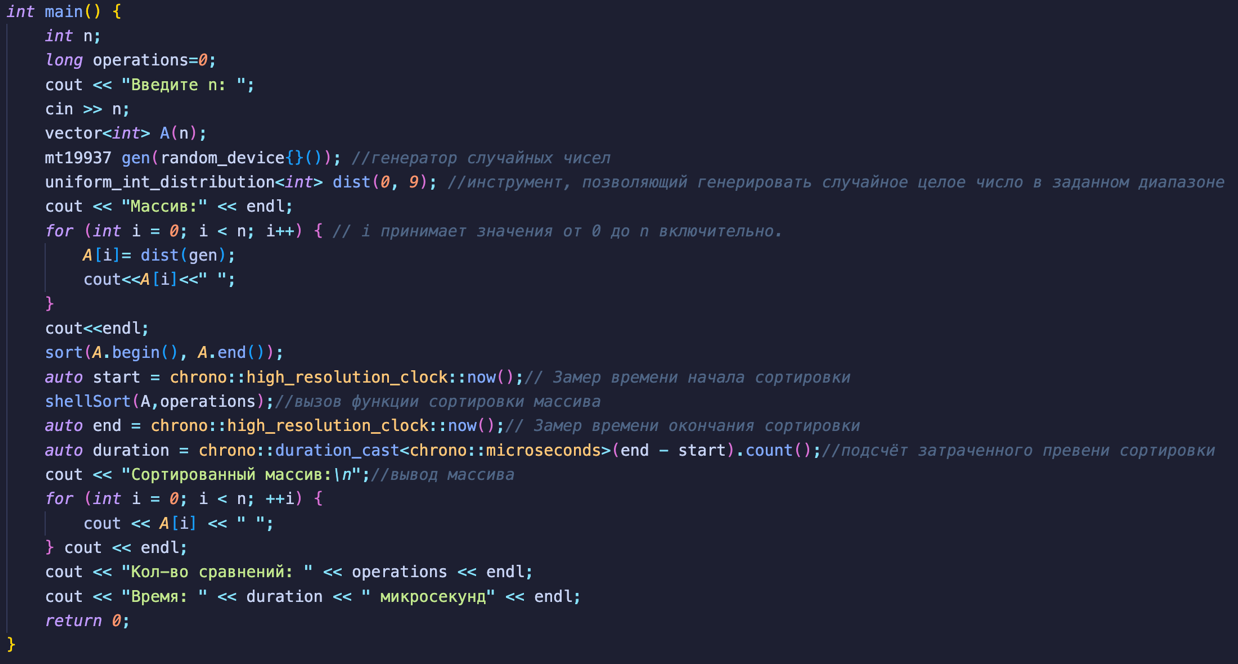


Рисунок 24 – Функция main при n=10 и с отсортированными значениями по возрастанию

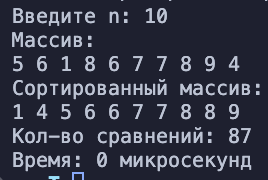


Рисунок 25 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Так как значения элементов массива идут в строго возрастающем порядке, то можно сделать вывод, что данная ситуация будет являться лучшим случаем, а следовательно сложность алгоритма равна O(n). Следовательно, в лучшем случае алгоритм является линейным. Результаты тестирования будут приведены в таблице 1.5.

Таблица 1.5. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп=Cп+Mп** |
| 100 | 0.005 | 1732 |
| 1000 | 0.074 | 27317 |
| 10000 | 0.558 | 376262 |
| 100000 | 7.527 | 4835787 |
| 1000000 | 86.816 | 59021392 |

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.26).

### 

Рисунок 26 - График функции роста Тп алгоритма Шелла со смещением Седжвика с отсортированными значениями по возрастанию от размера массива n

### **2.8.3 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма быстрой сортировки (Хоара)**

Программа будет протестирована на массивах с элементами, упорядоченными в строго убывающем порядке для значений n = 100, 1000, 10000, 100000 и 1000000. Продемонстрируем работу программы при n = 10. Алгоритм быстрой сортировки(Хоара) останется без изменений, как показано на рисунке 5. На основе данных, представленных в таблице 1.5, мы построим график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по убыванию.

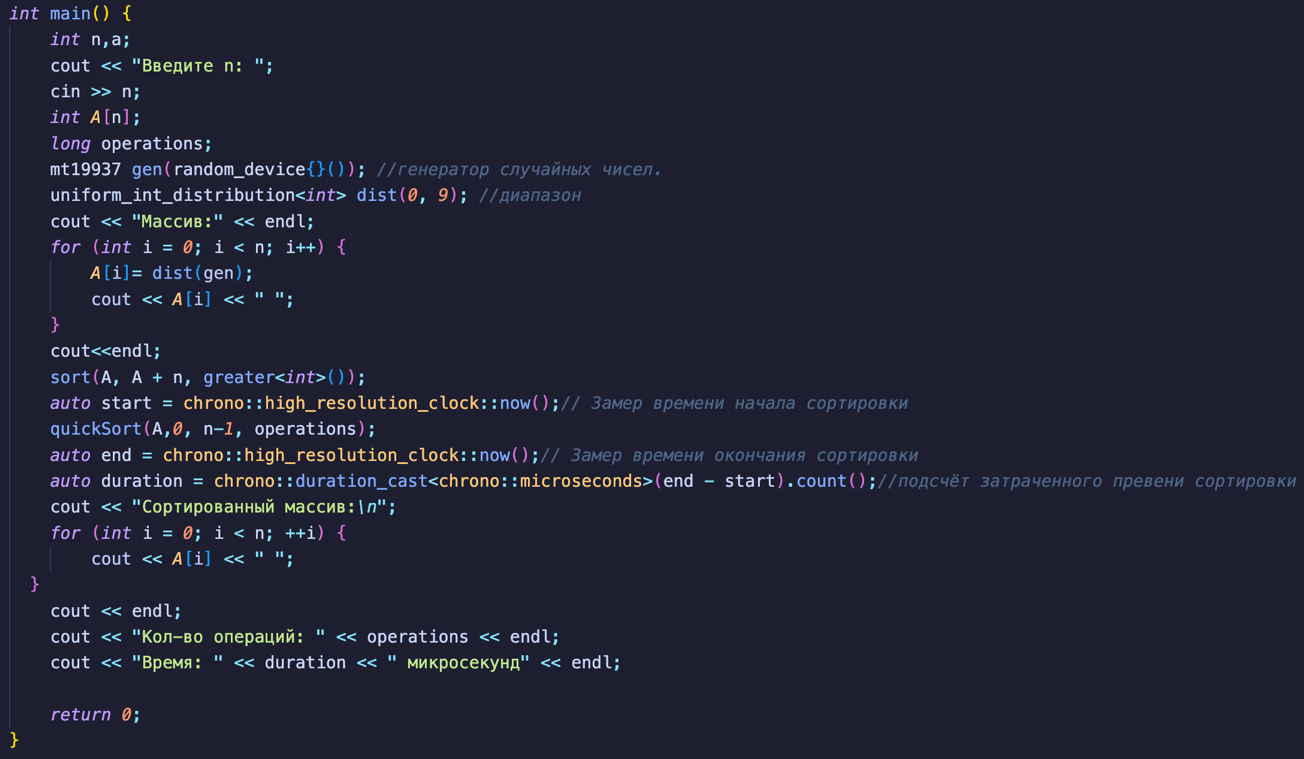


Рисунок 28 – Функция main при n=10 и с отсортированными значениями по убыванию

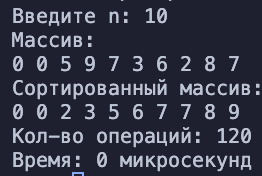


Рисунок 29 – Результат программы при n=10 и с отсортированными значениями по убыванию

Таблица 1.6. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.009 | 1934 |
| 1000 | 0.07 | 26989 |
| 10000 | 0.740 | 349787 |
| 100000 | 9.273 | 4474122 |
| 1000000 | 111.838 | 52657581 |

На основе полученных данных, продемонстрированных в таблице 1.6, построим график функции роста Тп алгоритма сортировки простым обменом с отсортированными значениями по убыванию от размера массива n (рис.30).

### 

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

### **2.8.4 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма быстрой сортировки (Хоара)**

Программа будет протестирована на массивах с элементами, упорядоченными в строго возрастающем порядке для значений n = 100, 1000, 10000, 100000 и 1000000. Продемонстрируем работу программы при n = 10. Алгоритм быстрой сортировки(Хоара) останется без изменений, как показано на рисунке 5. На основе данных, представленных в таблице 1.7, мы построим график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по возрастанию.



Рисунок 32 – Функция main при n=10 и с отсортированными значениями по возрастанию

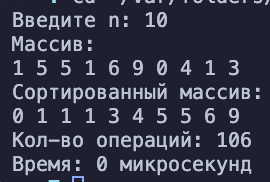


Рисунок 33 – Результат тестирования при n=10 и с отсортированными значениями по возрастанию

Таблица 1.7. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.006 | 1865 |
| 1000 | 0.084 | 26681 |
| 10000 | 0.967 | 339714 |
| 100000 | 12.543 | 4433681 |
| 1000000 | 108.494 | 52250827 |

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.34).

### 

Рисунок 34 - График функции роста Тп алгоритма быстрой сортировки (Хоара) с отсортированными значениями по возрастанию от размера массива n

## **2.9 Вывод по заданию №1**

Исходя из результатов тестирования, алгоритмы сортировки, такие как быстрая сортировка, сортировка обменом и сортировка Шелла со смещением Седжвика, могут иметь различную производительность в зависимости от исходной упорядоченности массива.

1. Быстрая сортировка:

- На уникальных или случайных данных быстрая сортировка часто имеет хорошую производительность и эффективность.

- При сортировке уже упорядоченных массивов быстрая сортировка может работать медленнее из-за большого числа перестановок.

2. Сортировка обменом:

- Сортировка обменом может быть эффективной на частично упорядоченных данных.

- Она работает медленнее на уже упорядоченных данных из-за того, что сравнения и перестановки происходят даже если массив уже отсортирован.

3. Сортировка Шелла со смещением Седжвика:

- Шелл смещением Седжвика обычно хорошо работает на случайных данных и частично упорядоченных массивах.

- Эффективность сортировки Шелла может зависеть от выбранного шага и самого смещения.

Исходя из результатов, алгоритм быстрой сортировки обычно демонстрирует более стабильное поведение на различных типах упорядоченности массива, в то время как сортировка простым обменом может показывать недостаточную эффективность на упорядоченных данных. Сортировка Шелла со смещением Седжвика, в свою очередь, имеет потенциал быть эффективной на частично упорядоченных данных. Важно учитывать эти зависимости при выборе алгоритма сортировки для конкретной задачи.

Таким образом, различные алгоритмы сортировки могут иметь различные зависимости от исходной упорядоченности массива. Конечный выбор алгоритма должен основываться на конкретных условиях и требованиях задачи, таких как характер данных и требования к производительности.

# 

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи**

Асимптотический анализ сложности алгоритмов

Требования по выполнению задания

1. Из материалов предыдущей практической работы приведите в отчёте формулы Тт(n) функций роста алгоритма простой сортировки обменом 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 Шелла со смещением Седжвика и быстрой сортировки (Хоара).

6. Общие результаты свести в табл. 2.

7. Сделать вывод о наиболее эффективном алгоритме из трёх.

## **3.2 Формулы функции роста алгоритма сортировки простым обменом в худшем и лучшем случае**

Лучший случай - массив уже отсортирован. В этом случае количество операций сравнения и перемещения будет минимальным и будет составлять Тт(n)=n.

Средний случай - массив заполнен случайными числами. В этом случае алгоритм будет иметь сложность Тт(n)=(n2-n)/2.

Худший случай - массив отсортирован в обратном порядке. В этом случае количество операций также будет Тт(n)=3\*(n2-n)/2.

Ёмкостная сложность алгоритма будет равна O(1).

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

Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом для худшего случая в О-нотации (оценка сверху) будет равна О(n2).

Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом для лучшего случая в Ω-нотации (оценка снизу) будет равна Ω(n).

Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом для среднего случая в θ-нотации будет равна θ(n2).

Ёмкостная сложность алгоритма простой сортировки обменом O(1).

## **3.4 Графическое представление функции роста и полученных асимптотических оценок сверху и снизу**

На полученных данных в пунктах 3.2 и 3.3, мы можем сделать графическое представление роста и полученных асимптотических оценок сверху и снизу(рис.35).

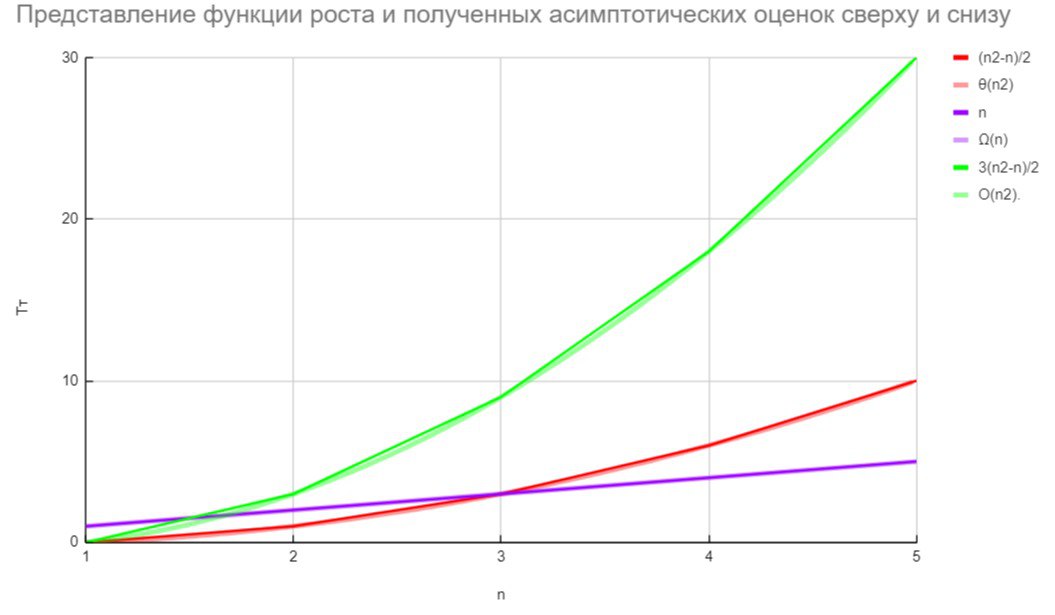


Рисунок 35 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритма Шелла со смещением Седжвика и быстрой сортировки (Хоара)**

### **3.5.1 Справочная информация о вычислительной сложности алгоритма Шелла со смещением Седжвика**

В худшем случае алгоритм шейкерной сортировки имеет оценку сложности O(n3/2), что указывает на возможность значительного увеличения времени выполнения при больших объемах данных. В лучшем случае сложность данного алгоритма составляет Ω(nlog(n)), что говорит о более эффективной работе при оптимальных условиях. Для среднего случая оценка сложности шейкерной сортировки в θ-нотации равна θ(n1.3), что подчеркивает его нестабильность при обработке различных входных данных.

Ёмкостная сложность алгоритма шейкерной сортировки оценивается как O(1), что говорит о небольшом объеме дополнительной памяти, необходимой для его работы.

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

В худшем случае алгоритм быстрой сортировки (Хоара) имеет оценку сложности O(n2) в большинстве случаев. В то же время, в лучшем случае его сложность в Ω-нотации составляет Ω(n log n), что говорит о более эффективной работе алгоритма в оптимальных условиях. Для среднего случая, оценка сложности быстрой сортировки составляет θ(n log n), что является оптимальным сочетанием скорости работы и эффективности.

Ёмкостная сложность алгоритма шейкерной сортировки оценивается как O(log n), что указывает на относительно небольшое количество дополнительной памяти, необходимой для выполнения данного алгоритма.

## **3.6 Таблица асимптотической сложности трёх алгоритмов**

На основе данных из пункта 3.3 и 3.5 заполним таблицу 2 асимптотической сложности алгоритма для алгоритмов сортировки простого обмана, шейкерной сортировки и быстрой сортировки (Хоара). А также укажем ёмкостную сложность данных алгоритмов сортировок.

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) |
| Шелла со смещением Седжвика | О(n3/2) | θ(nlog2n) | θ(n1.3) | О(1) |
| Быстрая сортировка (Хоара) | О(n2) | Ω(nlog2n) | θ(nlog2n) | О(log2n) |

**3.7 Выводы по заданию №2**

По результатам анализа и сравнения трех алгоритмов сортировки - простого обмена, сортировки Шелла с последовательностью Седжвика и быстрой сортировки (Хоара) - можно сделать следующий вывод:

Быстрая сортировка (Хоара) обычно считается наиболее эффективным алгоритмом для сортировки больших объемов данных. Она имеет среднюю асимптотическую сложность O(n log n) и часто превосходит другие алгоритмы, особенно на практике. Быстрая сортировка обычно более эффективна в сравнении с сортировкой Шелла и простым обменом, особенно когда необходимо обработать большие наборы данных.

Таким образом, на основе проведенного анализа наиболее эффективным алгоритмом из трех является быстрая сортировка (Хоара) благодаря своей высокой производительности и быстроте работы на больших объемах данных.

# 5 ВЫВОДЫ

В ходе выполнения практической работы были осуществлены следующие задачи:

- Получены навыки по анализу вычислительной сложности алгоритмов сортировки и выбору наиболее эффективного метода;

- Проанализированы алгоритмы быстрой сортировки (Хоара) и сортировки Шелла с последовательностью Седжвика;

- Написаны программы для алгоритмов быстрой сортировки (Хоара) и Шелла с последовательностью Седжвика;

- Проведено тестирование программ для упомянутых алгоритмов сортировки;

- Построены графики функций роста времени для алгоритмов быстрой сортировки (Хоара) и сортировки Шелла с последовательностью Седжвика;

- Осуществлено сравнение алгоритмов простой обменной сортировки, быстрой сортировки (Хоара) и сортировки Шелла с последовательностью Седжвика;

- Проведен анализ асимптотической сложности алгоритмов простой обменной сортировки, быстрой сортировки (Хоара) и сортировки Шелла с последовательностью Седжвика;

- Выполнено сравнение асимптотической сложности упомянутых алгоритмов сортировки;

- Определено наиболее эффективный алгоритм. Таким образом, главную цель практической работы - получение навыков по анализу вычислительной сложности алгоритмов сортировки и выбору наиболее эффективного метода, можно считать достигнутой.

# 6 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).