|  |
| --- |
| 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: Жаворонкова 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 Математическая модель решения алгоритма 6](#_3znysh7)

[2.2.1 Описание выполнения и блок-схема алгоритма шейкерной сортировки 6](#_2et92p0)

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

[2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма шейкерной сортировки 10](#_3dy6vkm)

[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](#_e649yhsttask)

[2.4.1 Описание выполнения и блок-схема алгоритма быстрой сортировки(Хоара) 14](#_51fgcrppcboj)

[2.4.2 Доказательство корректности циклов алгоритма быстрой сортировки(Хоара) 16](#_2bitored8vmr)

[2.4.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма быстрой сортировки(Хоара) 17](#_omwqrk7588p4)

[2.5 Реализация алгоритма на языке C++, проведение тестирования и построение графика 18](#_uedq667lbdxv)

[2.5.1 Реализация алгоритма быстрой сортировки(Хоара) на языке C++ 18](#_wgxamttx0xzb)

[2.5.2 Тестирование и построение графика 19](#_tqghpp5knyyv)

[2.6 Сортировка простым обменом 21](#_3rdcrjn)

[2.7 Сравнение трёх алгоритмов на графике 22](#_ew4qhgrbv55a)

[2.8 Тестирование программ для алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 23](#_35nkun2)

[2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма шейкерной сортировки 24](#_1ksv4uv)

[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](#_so7wqcgexw2f)

[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](#_44sinio)

[3 ЗАДАНИЕ №2 36](#_oz8bmwxvfhap)

[3.1 Формулировка задачи 36](#_pg2eydq5vm9s)

[3.2 Формулы функции роста алгоритма сортировки простым обменом в худшем и лучшем случае 36](#_dgwnybt3r9s4)

[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](#_7dmu0je7z4rn)

[3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 38](#_6exowko9ztm6)

[3.5.1 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки 38](#_hodmdv3inyuw)

[3.5.2 Справочная информация о вычислительной сложности алгоритма быстрой сортировки(Хоара) 39](#_vomghf5fhtsf)

[3.6 Таблица асимптотической сложности трёх алгоритмов 39](#_147n2zr)

[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**

Эмпирическая оценка эффективности алгоритмов.

1. Разработать алгоритм шейкерной сортировки, 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 шейкерной сортировки.

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 т 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**

Шейкерная сортировка (также называется коктейльная, перемешиванием, двунаправленная пузырьковая – Shaker Sort, Shuffle Sort, Shuttle Sort) – это улучшение обменной сортировки (пузырька), в том числе за счёт:

- условия Айверсона: все элементы с индексами, превышающими k (индекс элемента, участвовавшего в последнем обмене), уже упорядочены;

- чередования направлений просмотра массива, что обеспечивает более быстрое перемещение элементов на свои места.

Шейкерная сортировка отличается от пузырьковой тем, что просмотры элементов выполняются один за другим в противоположных направлениях, при этом большие элементы стремятся к концу массива, а маленькие - к началу. Своим появлением эта сортировка обязана такому недостатку сортировки «пузырьком», при котором движение от конца массива к началу минимального элемента «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо. Такая ассиметрия и вызвала появление новой сортировки, на каждом шаге которой осуществляется проход как в одну сторону, так и в другую. Таким образом, на каждом шаге минимальный элемент всплывает на поверхность, а максимальный элемент − тонет.

Основные характеристики шейкерной сортировки:

* Просмотр массива осуществляется не до конца (начала) массива, а до конкретной позиции.
* Границы сортируемой части массива сдвигаются на 1 позицию на каждой итерации.
* Массив просматривается поочередно справа налево и слева направо.
* Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания).
* Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.1,2,3,4).

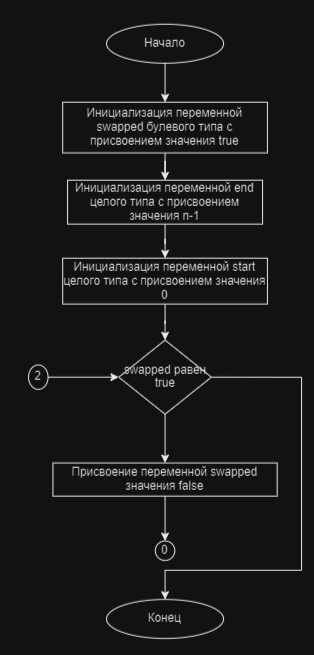


Рисунок 1 – Блок-схема алгоритма шейкерной сортировки

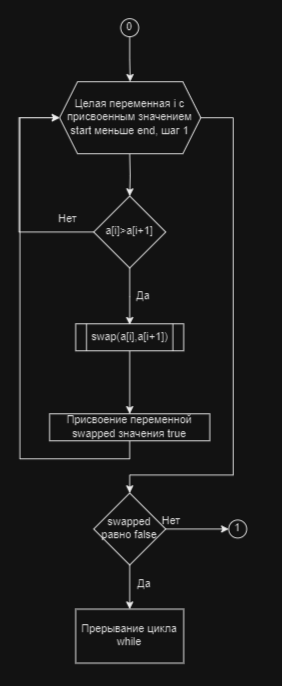


Рисунок 2 – Блок-схема алгоритма шейкерной сортировки

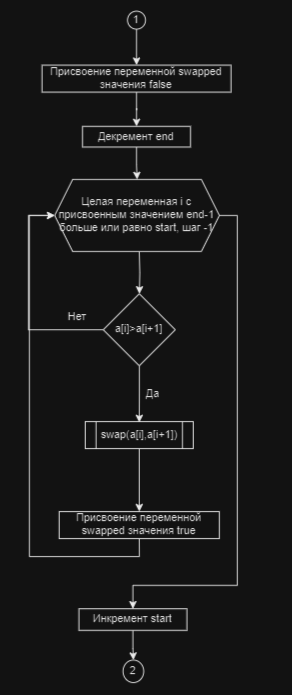


Рисунок 3 – Блок-схема алгоритма шейкерной сортировки

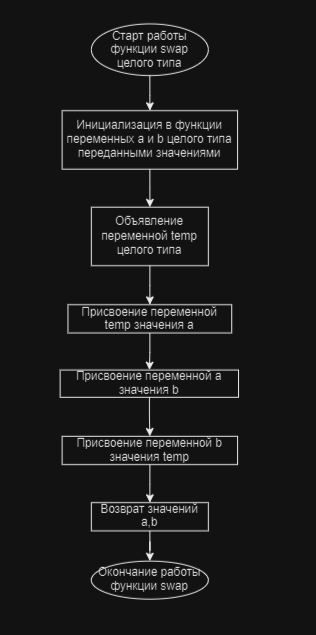


Рисунок 4 – Блок-схема подпрограммы обмена элементов

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

Инвариант для внешнего цикла: значение переменной swapped равно true.

Инвариант для внутреннего цикла: значение переменной i всегда меньше end.

Инвариант для внутреннего цикла: значение переменной i всегда больше или равно start.

Докажем конечность циклов. Внешний цикл алгоритма выполняется, пока переменная swapped равна true, что означает, что произошла хотя бы одна перестановка элементов. Каждый элемент массива будет просмотрен хотя бы один раз за один проход по массиву. При каждом проходе внутренних циклов наименьший (наибольший) элемент "всплывает" на своё место. Поэтому длина неотсортированной части массива уменьшается на каждом повторении. Если в процессе обработки массива не было произведено ни одной перестановки элементов, то массив уже отсортирован и алгоритм завершает работу. Следовательно, циклы данного алгоритма конечны.

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**

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

Средний случай - массив заполнен случайными числами. В этом случае алгоритм будет иметь сложность O(n2).

Худший случай - массив отсортирован в обратном порядке. В этом случае количество операций также будет O(n2).

b. Функции роста времени:

Лучший случай: O(n).

Худший случай: 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++**

Реализуем данный алгоритм на языке C++(рис.5,6). Для реализации понадобятся такие библиотеки, как iostream, random, chrono. Iostream — это заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++.Random - позволяет генерировать случайные числа в диапазоне. В данной программе задан диапазон от 1 до 10. Chrono позволяет реализовать такие концепции, как: интервалы времени, моменты времени, таймеры. Для подсчёта количество операций присваивания или сравнения введём переменную operations, которая представляет собой целое число в диапазоне от -2 147 483 648 до 2 147 483 648 и занимает 4 байта в памяти.

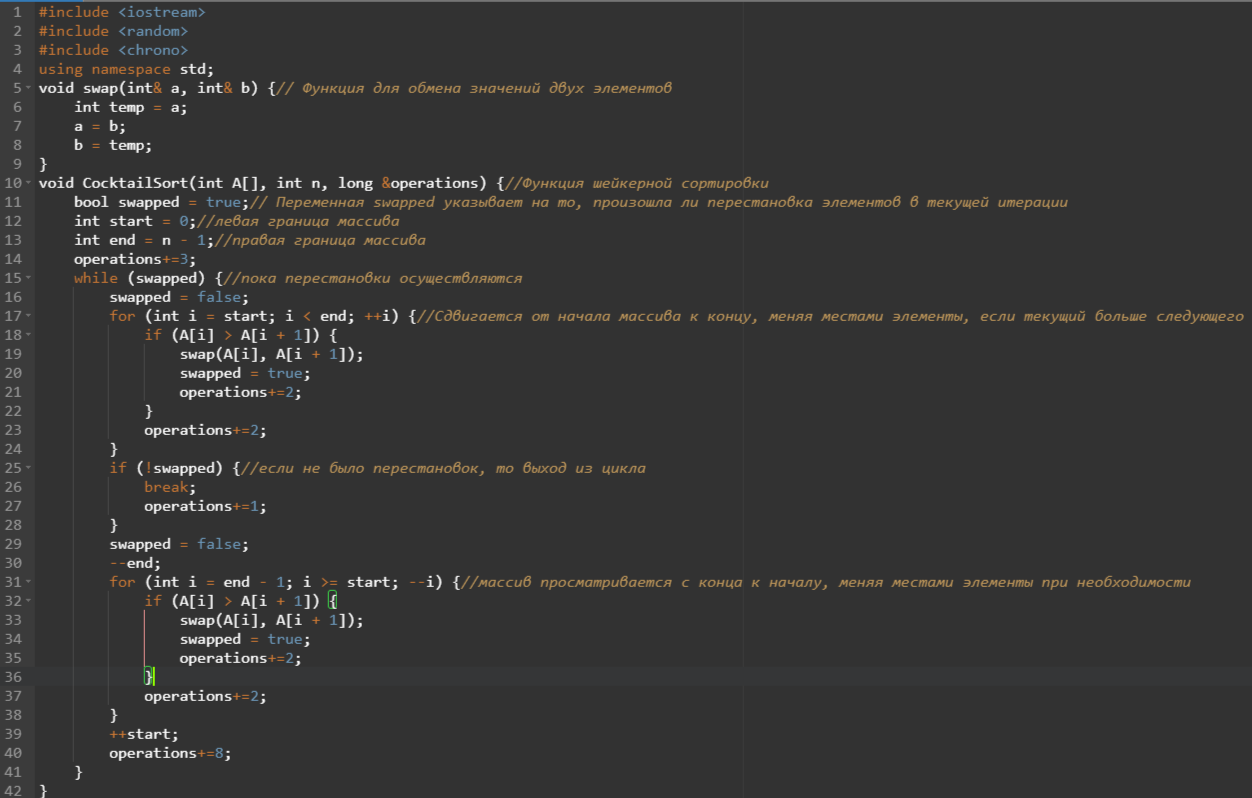


Рисунок 5 – Программа алгоритма шейкерной сортировки и обмена значений



Рисунок 6 – Функция main для алгоритма шейкерной сортировки

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

Стоит задача протестировать программу с заданным размером массива n=10 (рис.7), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование понадобился ввод с случайной генерацией числа. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.1. Воспользуемся структурой high\_resolution\_clock для подсчёта затраченного времени на сортировку. Для более точных результатов в программе будем рассматривать микросекунды, которые в дальнейшем, для заполнения таблицы, переведем в миллисекунды.

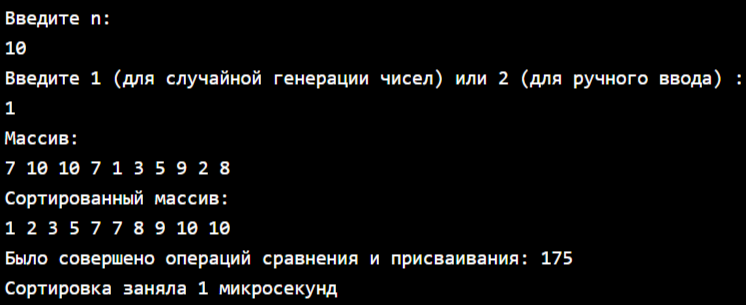


Рисунок 7 - Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.039 | 10989 |
| 1000 | 2.965 | 1197661 |
| 10000 | 348.584 | 118280917 |
| 100000 | 49047.421 | 11038257949 |
| 1000000 | 4215867.28 | 1170495346447 |

На основе полученных данных, продемонстрированных в таблице 1.1, построим график функции роста Тп алгоритма сортировки простым обменом с отсортированными значениями по убыванию от размера массива n (рис.8).

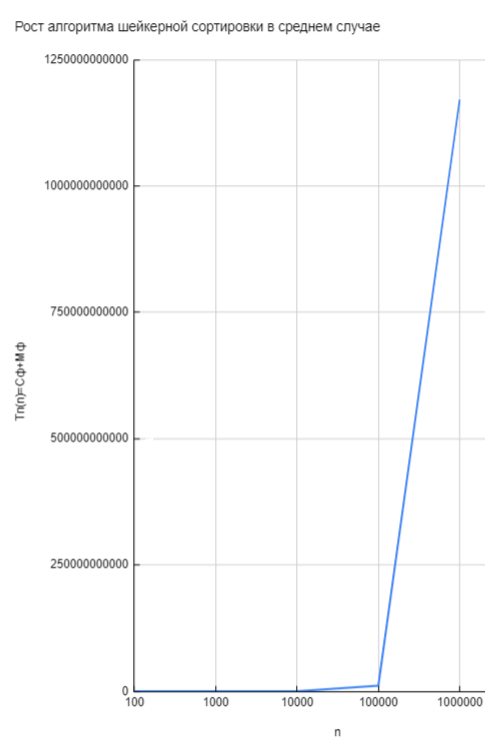


Рисунок 8 - График функции роста Тп алгоритма шейкерной сортировки от размера массива n

## **2.4 Математическая модель решения алгоритма**

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

Это улучшенный вариант сортировки пузырьком. Принципиальное отличие от сортировки пузырьком или шейкерной сортировки 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). Основные шаги:

1. Выбрать опорный элемент.

2. Разбиение: элементы меньше опорного помещаются перед ним, а не меньшие – после.

3. Рекурсивно применить шаги 1 и 2 к двум подмассивам слева и справа от опорного элемента (в подмассиве должно быть >1 элемента).

4. Комбинирование не нужно (подмассивы сортируются на месте) – в результате весь массив оказывается отсортирован.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.9,10,11,12).

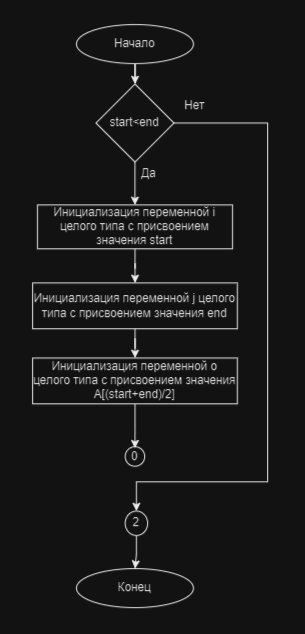


Рисунок 9 – Блок-схема алгоритма быстрой сортировки(Хоара)

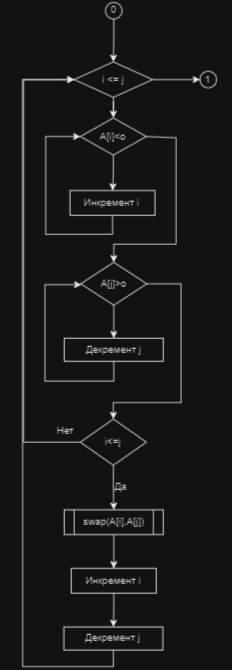


Рисунок 10 – Блок-схема алгоритма быстрой сортировки(Хоара)

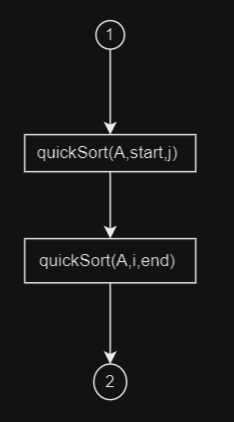


Рисунок 11 – Блок-схема алгоритма быстрой сортировки(Хоара)

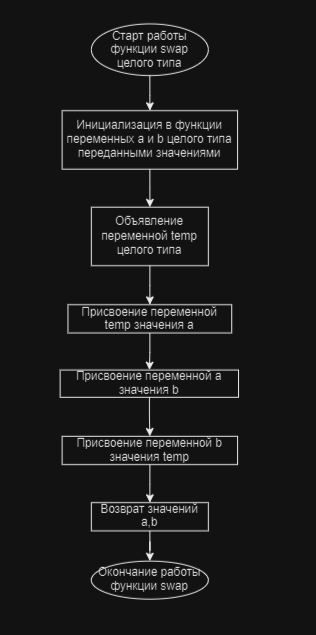


Рисунок 12 – Блок-схема подпрограммы обмена элементов

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

Инвариант для внешнего условного оператора: значение переменной start меньше end.

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.

Инвариант для внутреннего условного цикла: значение переменной i всегда больше j.

Докажем конечность циклов. Переменные i и j ограничены диапазоном start, end. На каждом повторении внешнего цикла хотя бы один из указателей (i или j) смещается (увеличивается или уменьшается). После каждого смещения значения i и j приближаются друг к другу, так как один указатель увеличивается, а другой уменьшается. Внутренние цикл при каждом повторении увеличивает i или уменьшает i. Таким образом, диапазон для i и j будет уменьшаться на каждой итерации, и в какой-то момент i будет больше j или равно j, что приведет к завершению цикла. При каждом рекурсивном вызове размер диапазона (end - start) уменьшается, так как либо j уменьшается (в случае рекурсии с параметрами A, start, j), либо i увеличивается (в случае рекурсии с параметрами A, i, end). Так как размер диапазона уменьшается при каждом рекурсивном вызове, исходное условие start < end гарантирует, что диапазон будет уменьшаться до тех пор, пока он не станет меньше или равен 0, что приведет к завершению рекурсии. Следовательно, циклы данного алгоритма конечны.

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

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

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

Средний случай - массив заполнен случайными числами. В этом случае алгоритм будет иметь сложность O(n log n).

Худший случай - массив отсортирован в обратном порядке. В этом случае количество операций также будет O(n2).

b. Функции роста времени:

Лучший случай: O(n log n).

Худший случай: 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 квазилинейным ростом размера 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++**

Реализуем данный алгоритм на языке C++(рис.13,14). Для реализации понадобятся такие библиотеки, как iostream, random, chrono. Iostream — это заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++.Random - позволяет генерировать случайные числа в диапазоне. В данной программе задан диапазон от 1 до 10. Chrono позволяет реализовать такие концепции, как: интервалы времени, моменты времени, таймеры. Для подсчёта количество операций присваивания или сравнения введём переменную operations, которая представляет собой целое число в диапазоне от -2 147 483 648 до 2 147 483 648 и занимает 4 байта в памяти.



Рисунок 13 – Программа алгоритма быстрой сортировки(Хоара)



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

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

Стоит задача протестировать программу с заданным размером массива n=10 (рис.15), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование понадобился ввод с случайной генерацией числа. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.2. Воспользуемся структурой high\_resolution\_clock для подсчёта затраченного времени на сортировку. Для более точных результатов в программе будем рассматривать микросекунды, которые в дальнейшем, для заполнения таблицы, переведем в миллисекунды.

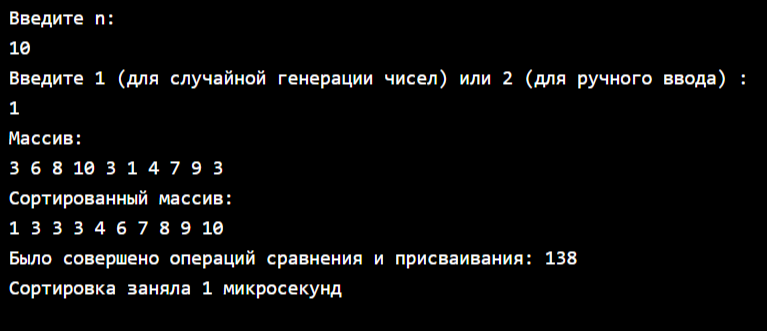


Рисунок 15 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.007 | 2079 |
| 1000 | 0.089 | 28895 |
| 10000 | 1.004 | 367205 |
| 100000 | 11.095 | 4489825 |
| 1000000 | 381.813 | 55224847 |

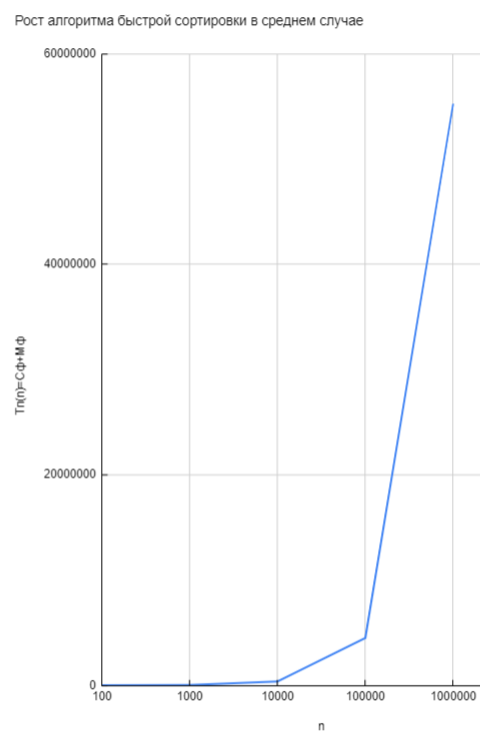


Рисунок 16 - График функции роста Тп алгоритма быстрой сортировки от размера массива n

## **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 | 0.05 | 12236 |
| 1000 | 4.526 | 1234682 |
| 10000 | 394.969 | 122374417 |
| 100000 | 33726.936 | 12254127887 |
| 1000000 | 4215867.28 | 1270495346447 |

## **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 шейкерной сортировки, 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.

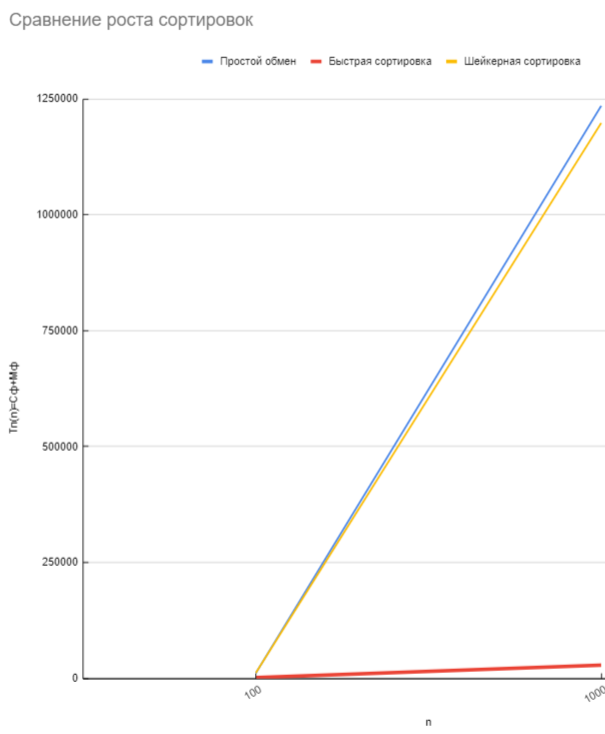


Рисунок 17 - График сравнения трёх сортировок в среднем случае при n до 1000

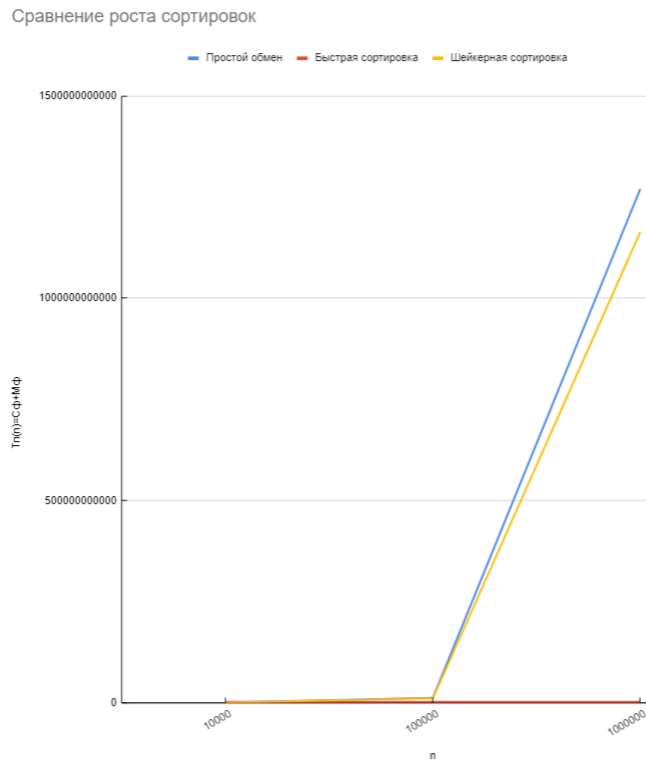


Рисунок 18 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

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 шейкерной сортировки второй по эффективности, а алгоритм быстрой сортировки(Хоара) самый эффективный.

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

Дополнительное тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов.

### **2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма шейкерной сортировки**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго убывающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по убыванию (рис.19). В функцию main добавим вызов функции сортировки по убыванию (рис.20) и продемонстрируем работу программы при n=10 (рис.21). Алгоритм шейкерной сортировки не изменяется и соответствует продемонстрированному на рисунке 5.



Рисунок 19 – Функция сортировки по убыванию



Рисунок 20 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

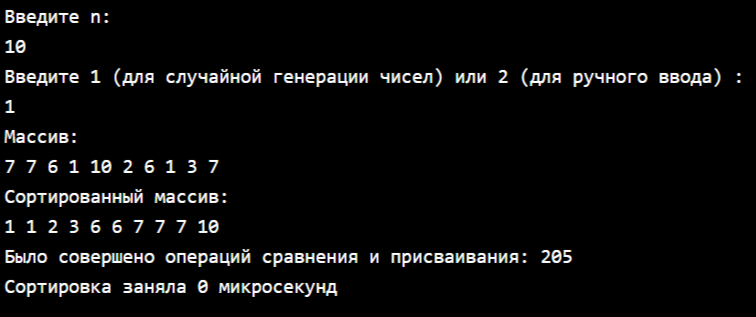


Рисунок 21 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Так как значения идут в строго убывающем порядке, то можно сделать вывод, что данная ситуация являться худшим случаем, а следовательно имеет сложность O(n2). Следовательно, в худшем случае алгоритм является квадратичным. Результаты тестирования будут приведены в таблице 1.4.

Таблица 1.4. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.041 | 19231 |
| 1000 | 3.462 | 1902533 |
| 10000 | 292.569 | 190004835 |
| 100000 | 29070.841 | 18999848665 |
| 1000000 | 3061159.557 | 1949841099825 |

На основе полученных данных, продемонстрированных в таблице 1.4, построим график функции роста Тп алгоритма сортировки простым обменом с отсортированными значениями по убыванию от размера массива n (рис.22).

### 

Рисунок 22 - График функции роста Тп шейкерная алгоритма 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 шейкерной сортировки**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго возрастающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по возрастанию(рис.23). В функцию main добавим вызов функции сортировки по возрастанию(рис.24) и продемонстрируем работу программы при n=10 (рис.25). Алгоритм шейкерной сортировки не изменяется и соответствует продемонстрированному на рисунке 5.

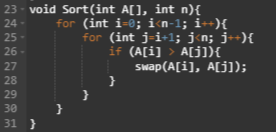


Рисунок 23 – Функция сортировки по возрастанию



Рисунок 24 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

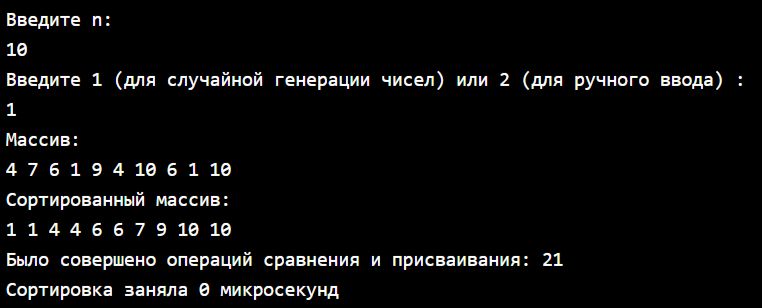


Рисунок 25 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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 | 0.000698 | 201 |
| 1000 | 0.004 | 2001 |
| 10000 | 0.024 | 20001 |
| 100000 | 0.286 | 200001 |
| 1000000 | 0.847 | 2000001 |

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 шейкерной сортировки 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)**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго убывающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по убыванию (рис.27). В функцию main добавим вызов функции сортировки по убыванию (рис.28) и продемонстрируем работу программы при n=10 (рис.29). Алгоритм быстрой сортировки(Хоара) не изменяется и соответствует продемонстрированному на рисунке 13.



Рисунок 27 – Функция сортировки по убыванию



Рисунок 28 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

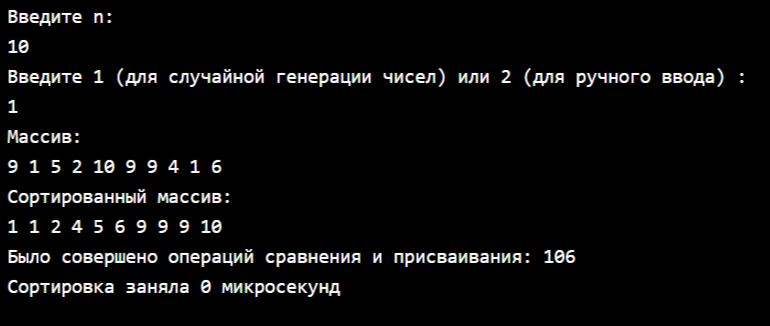


Рисунок 29 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Так как значения идут в строго убывающем порядке, то можно сделать вывод, что данная ситуация являться худшим случаем, а следовательно имеет сложность O(n2). Следовательно, в худшем случае алгоритм является квадратичным. Результаты тестирования будут приведены в таблице 1.6.

Таблица 1.6. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.005 | 1931 |
| 1000 | 0.041 | 26843 |
| 10000 | 0.467 | 349281 |
| 100000 | 5.956 | 4472806 |
| 1000000 | 93.509 | 67092090 |

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)**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго возрастающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по возрастанию(рис.31). В функцию main добавим вызов функции сортировки по возрастанию(рис.32) и продемонстрируем работу программы при n=10 (рис.33). Алгоритм быстрой сортировки(Хоара) не изменяется и соответствует продемонстрированному на рисунке 13.

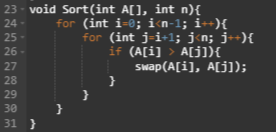


Рисунок 31 – Функция сортировки по возрастанию



Рисунок 32 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

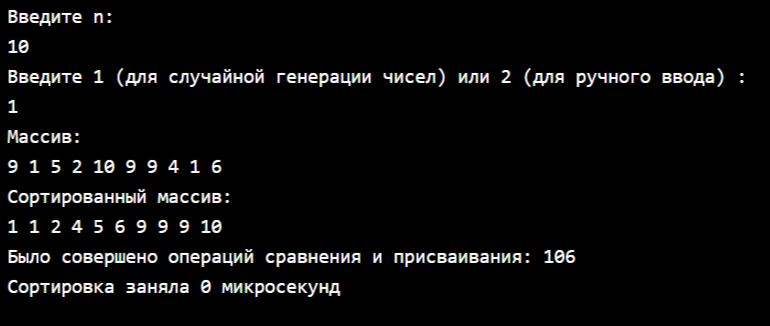


Рисунок 33 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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 log n). Следовательно, в лучшем случае алгоритм является квазилинейным. Результаты тестирования будут приведены в таблице 1.7.

Таблица 1.7. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.005 | 1921 |
| 1000 | 0.039 | 26611 |
| 10000 | 0.464 | 350450 |
| 100000 | 5.824 | 4432964 |
| 1000000 | 97.843 | 66494460 |

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 шейкерной сортировки с отсортированными значениями по возрастанию от размера массива n

## **2.9 Вывод по заданию №1**

Исходя из результатов тестирования, представленного в таблицах 1.1,1.2,1.4,1.5,1.6,1.7, можно сделать вывод о зависимости алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) от исходной упорядоченности массива. Шейкерная сортировка работает примерно одинаково хорошо как на упорядоченных, так и на перевернутых массивах. Это значит, что результаты ее работы не сильно меняются в зависимости от того, как изначально упорядочены элементы в массиве. В целом, шейкерная сортировка показывает стабильность и работает эффективно в любом случае. Быстрая сортировка(Хоара), в отличие от шейкерной сортировки, демонстрирует более выраженную зависимость от начальной упорядоченности массива. Наиболее эффективные результаты она показывает на случайных массивах, в то время как при упорядоченных или перевернутых массивах скорость ее работы снижается.

Таким образом, можно сделать вывод о том, что шейкерная сортировка менее зависима от начальной упорядоченности массива, в то время как быстрая сортировка(Хоара) более чувствительна к упорядоченности массива.

# 

# 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 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 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).

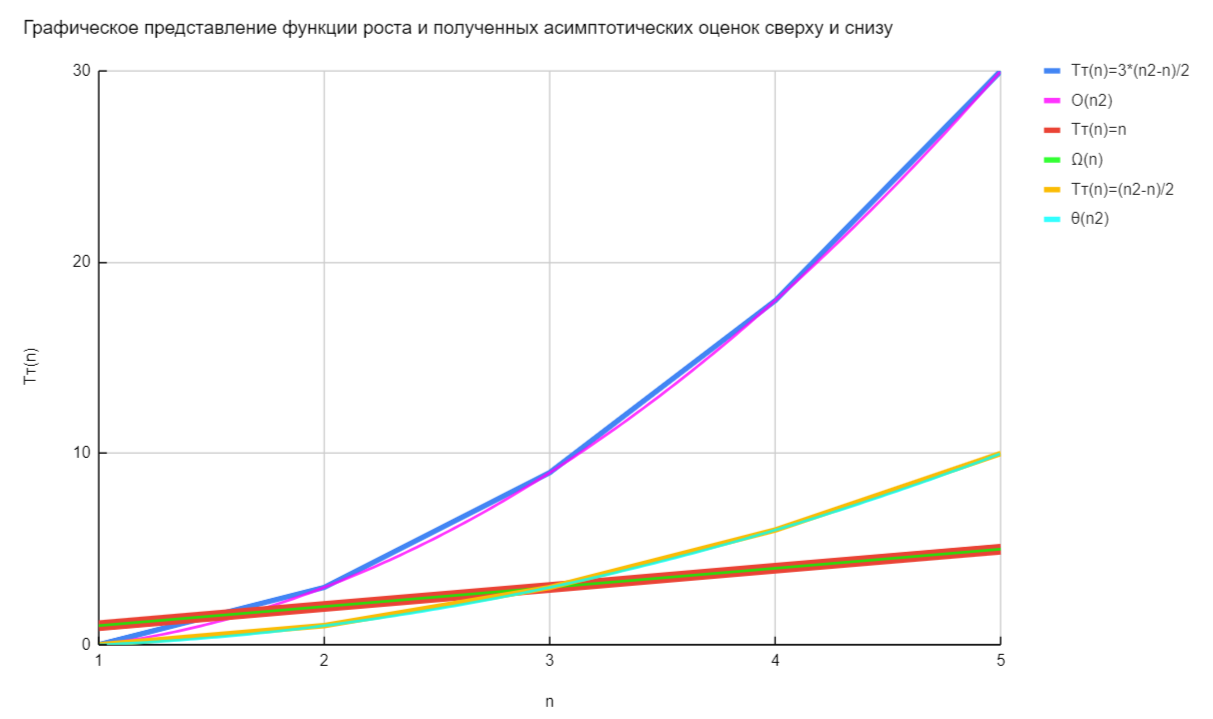


Рисунок 35 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и быстрой сортировки(Хоара)**

### **3.5.1 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки**

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

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

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

Ёмкостная сложность алгоритма шейкерной сортировки O(1).

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

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

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

Асимптотическая оценка вычислительной быстрой сортировки(Хоара) для среднего случая в θ-нотации будет равна θ(n log n).

Ёмкостная сложность алгоритма шейкерной сортировки O(log n).

## **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) |
| Шейкерная сортировка | О(n2) | Ω(n) | θ(n2) | О(1) |
| Быстрая сортировка (Хоара) | О(n2) | Ω(n log n) | θ(n log n) | О(log n) |

**3.7 Выводы по заданию №2**

На основе таблицы 2, можно сделать вывод, что наиболее эффективным алгоритмом сортировки является быстрая сортировка (Хоара). В среднем случае её асимптотическая сложность составляет θ(n log n), что является лучшим показателем среди представленных алгоритмов. В лучшем случае быстрая сортировка (Хоара) имеет асимптотическую сложность Ω(n log n), что также является очень хорошим показателем. В этом случае алгоритм быстрой сортировки работает оптимально и эффективно. Также быстрая сортировка имеет лучшую ёмкостную сложность O(log n), что также является преимуществом. Однако следует учитывать, что в наихудшем случае алгоритм быстрой сортировки имеет сложность O(n2), что может быть недостатком в некоторых ситуациях.

# 4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чём заключается улучшение сортировки?

Улучшить сортировку – т.е. уменьшить количество сравнений С и/или перемещений М значений в массиве.

2. В чём отличия шейкерной сортировки от простой сортировки обменом?

Шейкерная сортировка отличается от пузырьковой тем, что она двунаправленная: алгоритм перемещается не строго слева направо, а сначала слева направо, затем справа налево.

3. Насколько повышается эффективность в случае шейкерной сортировки?

Асимптотика у алгоритма такая же, как и у сортировки пузырьком, однако реальное время работы лучше.

4. Как работает сортировка Шелла?

Это улучшенный вариант сортировки вставками.

Сортируются подгруппы (подмножества) элементов массива, причём в подгруппе элементы идут не последовательно, а равномерно выбираются с некоторой дельтой (d-смещением) по индексу. Смещение методично уменьшается, пока не составит 1.

5. Почему затраты времени на частичную сортировку оправдывают себя?

Затраты времени на частичную сортировку могут быть оправданы по нескольким причинам:

* Уменьшение нагрузки на систему.
* Улучшение качества сортировки.
* Экономия времени и ресурсов.
* Увеличение производительности.
* Повышение точности.

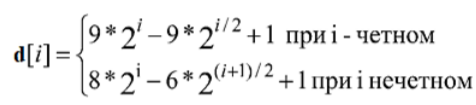
6. От чего зависит эффективность сортировки Шелла в среднем случае?

Эффективность сортировки Шелла в среднем случае зависит от нескольких факторов:

* Размер массива: Чем больше массив, тем дольше будет выполняться сортировка.
* Количество элементов в массиве: Чем больше элементов в массиве, тем дольше будет выполняться сортировка.
* Скорость процессора: Чем быстрее процессор, тем быстрее будет выполняться сортировка.

7. Опишите вариант использования смещения Седжвика в сортировке Шелла.

Смещение Седжвика – О(n4/3) в худшем и О(n7/6) в среднем случаях: Значение смещения, записываемого в элемент массива d, вычисляется по формуле:



Остановить создание и заполнение массива d необходимо на значении d[i-1], если 3\*d[i] > n (больше размера массива).

8. Опишите вариант использования смещений Кнута в сортировке Шелла.

Один из вариантов использования смещений Кнута - сортировка больших массивов данных. Если массив очень большой, то сортировка всего массива за один проход может занять много времени. Однако, если разбить массив на несколько частей и отсортировать каждую часть отдельно, то процесс сортировки будет проходить быстрее.

9. Назовите автора алгоритма сортировки простым слиянием. Каким был исторический контекст этой разработки?

Автор алгоритма сортировки простым слиянием Джон фон Нейман. Исторический контекст этой разработки связан с развитием компьютерных технологий и необходимостью разработки более эффективных алгоритмов сортировки данных.

10. Объясните идею алгоритма сортировки простым слиянием.

Алгоритм сортировки простым слиянием работает путем разделения исходного массива на две части и сравнения этих частей. Если одна из частей меньше другой, то меньшая часть помещается в результирующий массив, а большая часть остается без изменений. Затем процесс повторяется для каждой из частей, пока все элементы не будут отсортированы.

11. Как оценивается асимптотическая сложность алгоритма простого слияния?

Асимптотическая оценка вычислительной сложности алгоритма простого слияния для худшего случая в О-нотации (оценка сверху) будет равна О(n log n).

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

Асимптотическая оценка вычислительной простого слияния для среднего случая в θ-нотации будет равна θ(n log n).

12. В чём отличие алгоритма естественного слияния от простого?

Алгоритм естественного слияния отличается от простого слияния тем, что он использует более эффективные методы сравнения и сортировки данных. Он также использует более сложные алгоритмы для определения порядка сортировки, что позволяет получить более точные результаты.

13. Какой вариант сортировки слиянием более эффективен для работы с внешними данными?

Для работы с внешними данными более эффективным вариантом сортировки слиянием является алгоритм естественного слияния. Он позволяет работать с большими объемами данных и получать более точные результаты сортировки.

14. Как работает быстрая сортировка?

Это улучшенный вариант сортировки пузырьком. Основные шаги:

1. Выбрать опорный элемент.

2. Разбиение: элементы меньше опорного помещаются перед ним, а не меньшие – после.

3. Рекурсивно применить шаги 1 и 2 к двум подмассивам слева и справа от опорного элемента (в подмассиве должно быть >1 элемента).

4. Комбинирование не нужно (подмассивы сортируются на месте) – в результате весь массив оказывается отсортирован.

15. Как оценивается асимптотическая сложность алгоритма быстрой сортировки? От чего она зависит?

Асимптотическая сложность алгоритма быстрой сортировки оценивается как O(n log n). Она зависит от размера входных данных (n) и логарифма этого размера.

16. Какие есть варианты выбора опорного элемента?

Один из них - выбор среднего элемента в качестве опорного. Другой вариант - выбор минимального элемента в качестве опорного. Также можно выбрать случайный элемент в качестве опорного.

17. В чём отличие схемы Хоара в алгоритме быстрой сортировки?

Схема Хоара - это один из методов выбора опорного элемента в алгоритме быстрой сортировки. Он основан на том, что выбирается элемент с максимальным значением, и затем он меняется местами с элементом, который находится в середине массива. Это позволяет улучшить эффективность алгоритма, так как уменьшается количество сравнений и перестановок элементов.

18. Что такое скорость роста функции?

Скорость роста функции - это как быстро работает алгоритм при увеличении размера входных данных. Например, если функция имеет асимптотическую сложность O(n2), то это означает, что алгоритм будет работать в квадратичной зависимости от размера входных данных.

19. Что такое асимптотический анализ сложности алгоритмов?

Поведение Т(n) в зависимости от увеличения n в пределе называют асимптотической сложностью алгоритма.

20. Объясните смысл асимптотических оценок времени работы алгоритмов в нотациях θ, О, Ω, о, ω.

Асимптотические оценки времени работы алгоритмов используются для определения скорости работы алгоритма при увеличении размера входных данных. Нотация θ (тета) используется для оценки асимптотического поведения функции, она показывает порядок роста функции. Нотация О (большое О) используется для оценки верхней границы функции, она показывает насколько быстро растет функция. Нотация Ω (большое омега) используется для оценки нижней границы функции, она показывает насколько медленно растет функция. Нотация о (малое о) используется для оценки нижней границы функции, она показывает насколько медленно функция приближается к нулю. Нотация ω (омега) используется для оценки порядка роста функции.

21. В чём отличие нотаций О и о, а также Ω и ω?

Нотация O используется для оценки верхней границы функции, т.е. она показывает, насколько быстро растет функция. Нотация o используется для оценки нижней границы функции и показывает, насколько медленно растет функция.

Нотации Ω и ω используются для оценки асимптотического поведения функции. Ω оценивает нижнюю границу функции, а ω оценивает порядок роста функции.

22. Какие нотации чаще всего используют в практике асимптотической оценки сложности алгоритмов?

O, Ω, Θ, ω, o

23. Что такое асимптотически точная оценка сложности?

Асимптотически точная оценка сложности (Θ) - это оценка скорости работы алгоритма при увеличении размера входных данных. Она позволяет определить, насколько быстро или медленно работает алгоритм, и помогает выбрать наиболее эффективный алгоритм для решения конкретной задачи.

24. Назовите основные правила подсчёта асимптотической сложности.

Правило сложения: если f(n) и g(n) являются функциями сложности, то их сумма f(n)+g(n) также является функцией сложности.

Правило умножения: если f(n) и g(n) являются функциями сложности, то их произведение f(n)\*g(n) также является функцией сложности.

Правило деления: если f(n) является функцией сложности, то функция 1/f(n) также является функцией сложности.

25. Назовите основные классы сложности алгоритмов.

1, log2n, n, n\* log2n, n2, n3, 2n, n!

26. Перечислите свойства асимптотических сравнений.

1. Транзитивность.
2. Рефлексивность
3. Симметричность
4. Перестановочная симметрия.

# 5 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 и быстрой сортировки(Хоара);

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

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

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

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

- Проведен анализ асимптотической сложности алгоритмов сортировки простым выбором, шейкерной сортировки и быстрой сортировки(Хоара);

- Произведено сравнение асимптотической сложности алгоритмов сортировки простым выбором, шейкерной сортировки 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.

# 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).