|  |
| --- |
| 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: Лелюхин Н.С. |
|  |  |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_gjdgxs)

[2 ЗАДАНИЕ №1 5](#_30j0zll)

[2.1 Формулировка задачи (Вариант 2, в списке 18) 5](#_1fob9te)

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

[2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма шейкерной сортировки 8](#_tyjcwt)

[2.4 Реализация алгоритма на языке C++ и тестирование 8](#)

[2.5 Описание выполнения и блок-схема алгоритма простого слияния 10](#_4d34og8)

[2.6 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма простого слияния 11](#_17dp8vu)

[2.7 Реализация алгоритма на языке C++ и тестирования 12](#_3rdcrjn)

[2.8 Сортировка простым обменом 14](#_35nkun2)

[2.9 Сравнение трёх алгоритмов на графике 14](#_1ksv4uv)

[2.10 Тестирование программ для алгоритмов шейкерной сортировки и простого слияния 17](#_44sinio)

[2.10.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма шейкерной сортировки 17](#_2jxsxqh)

[2.10.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма шейкерной сортировки 18](#)

[2.10.3 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма простого слияния 19](#)

[2.10.4 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма простого слияния 21](#)

[2.11 Вывод по заданию №1 22](#_3j2qqm3)

[3 ЗАДАНИЕ №2 23](#_4i7ojhp)

[3.1 Формулировка задачи 23](#_2xcytpi)

[3.2 Формулы функции роста алгоритма сортировки простым обменом в худшем и лучшем случае 23](#_1ci93xb)

[3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом 24](#)

[3.4 Графическое представление функции роста и полученных асимптотических оценок сверху и снизу 24](#_3whwml4)

[3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и простого слияния 25](#_2bn6wsx)

[3.5.1 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки 25](#_qsh70q)

[3.5.2 Справочная информация о вычислительной сложности алгоритма простого слияния 25](#_3as4poj)

[3.6 Таблица асимптотической сложности трёх алгоритмов 26](#_1pxezwc)

[5 ВЫВОДЫ 27](#_2p2csry)

[6 ЛИТЕРАТУРА 28](#_147n2zr)

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи (Вариант 2, в списке 1**8**)**

Шейкерная сортировка, простое слияние

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 простого слияния, 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 простого слияния.

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 Описание выполнения и блок-схема алгоритма шейкерной сортировки**

Шейкерная сортировка, также известная как коктейльная сортировка или двунаправленная пузырьковая сортировка, представляет собой улучшенную версию обменной сортировки. Она включает в себя такие улучшения, как условие Айверсона, где элементы с индексами, превышающими определенное значение, уже отсортированы, а также изменение направлений просмотра массива для более эффективного перемещения элементов. В отличие от обычной пузырьковой сортировки, шейкерная сортировка осуществляет просмотр элементов в противоположных направлениях, перемещая большие элементы к концу массива и маленькие к его началу. Это позволяет эффективнее упорядочивать элементы и устранять недостатки классической пузырьковой сортировки.

Характеристики шейкерной сортировки:

* Просмотр массива осуществляется 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.
* Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

Реализуем описание выполнения алгоритма в виде блок-схемы (рис.1).

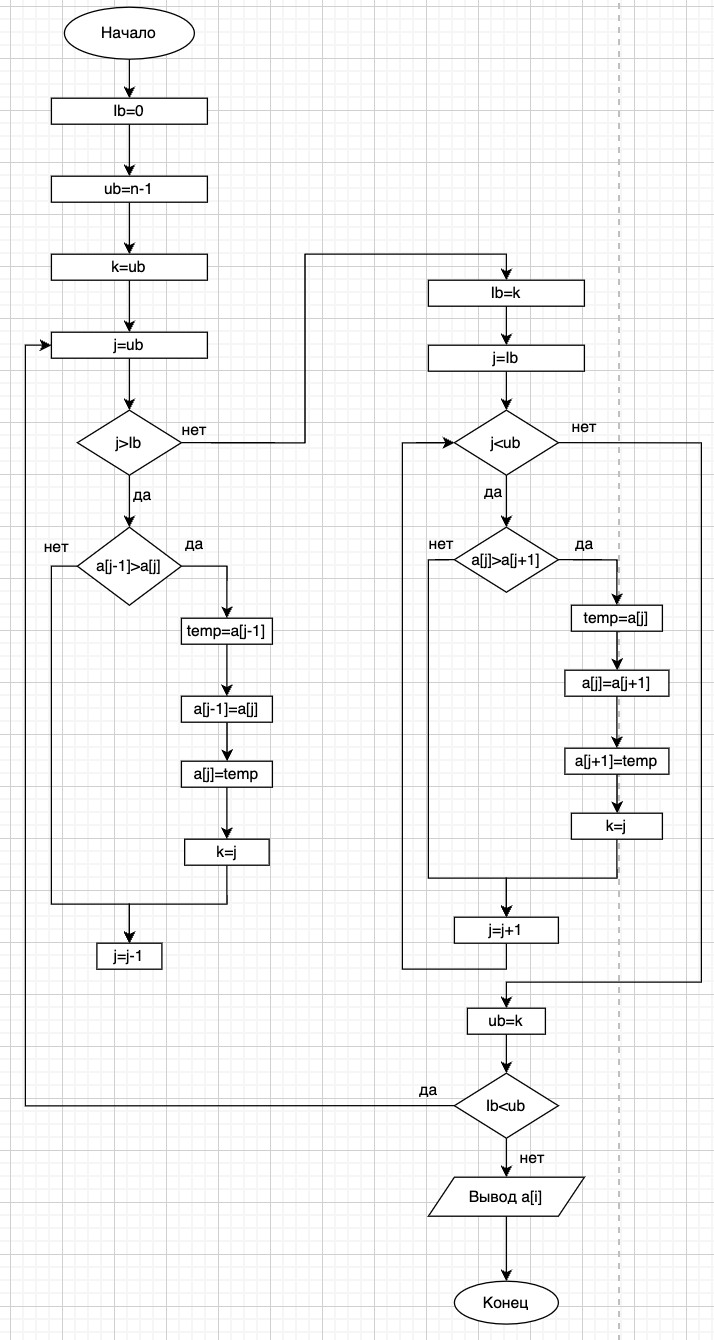


Рисунок 1 – Блок-схема алгоритма шейкерной сортировки

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

В лучшем случае алгоритма сортировки массива это, когда массив уже отсортирован. В таком случае минимальное количество операций сравнения и перемещения равно O(n). В среднем случае, когда массив заполнен случайными числами, сложность алгоритма будет O(n2). В худшем случае, когда массив отсортирован в обратном порядке, количество операций также будет O(n2).

b. Функции роста времени для данного метода сортировки:

Лучший случай: O(n)

Худший случай: O(n2)

При увеличении размера входного массива время исполнения в худшем случае увеличивается квадратично. Можно использовать квадратичную функцию для описания функции роста данного метода сортировки. Время исполнения в лучшем случае увеличивается линейно. Емкостная сложность алгоритма равна O(1).

## **2.**4 **Реализация алгоритма на языке C++ и тестирование**

Для реализации данного алгоритма на языке C++ (рис.2,3) понадобятся библиотеки iostream, random и chrono. Для подсчета количества операций присваивания или сравнения введена переменная o типа long. Проведём тестирование программы с заданным размером массива n=10 (рис.4), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование генерация случайных чисел в массиве. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.1. Воспользуемся библиотекой chrono для засекания времени. Для более точных результатов в программе будем рассматривать наносекунды.

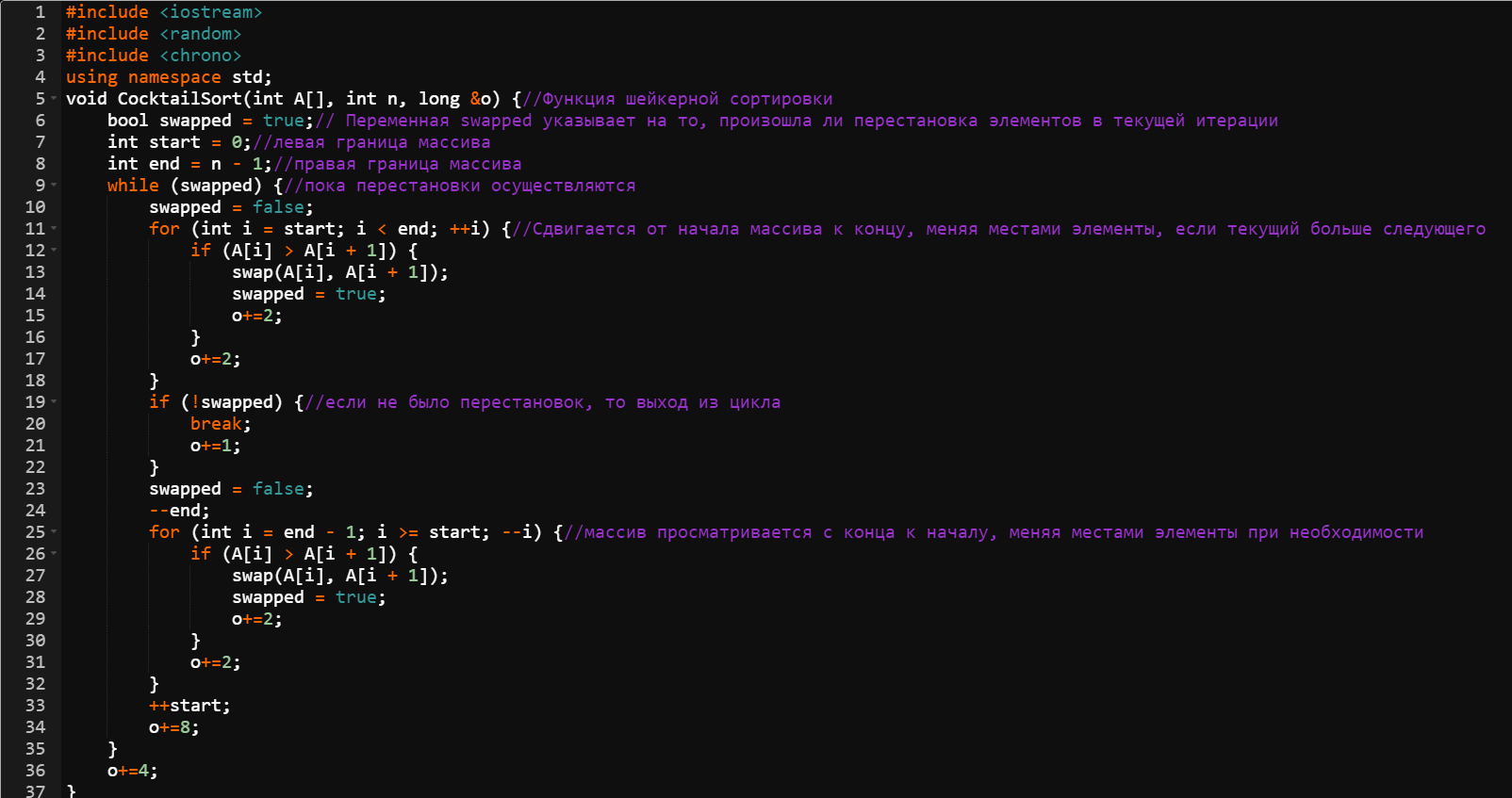


Рисунок 2 – Программа алгоритма шейкерной сортировки

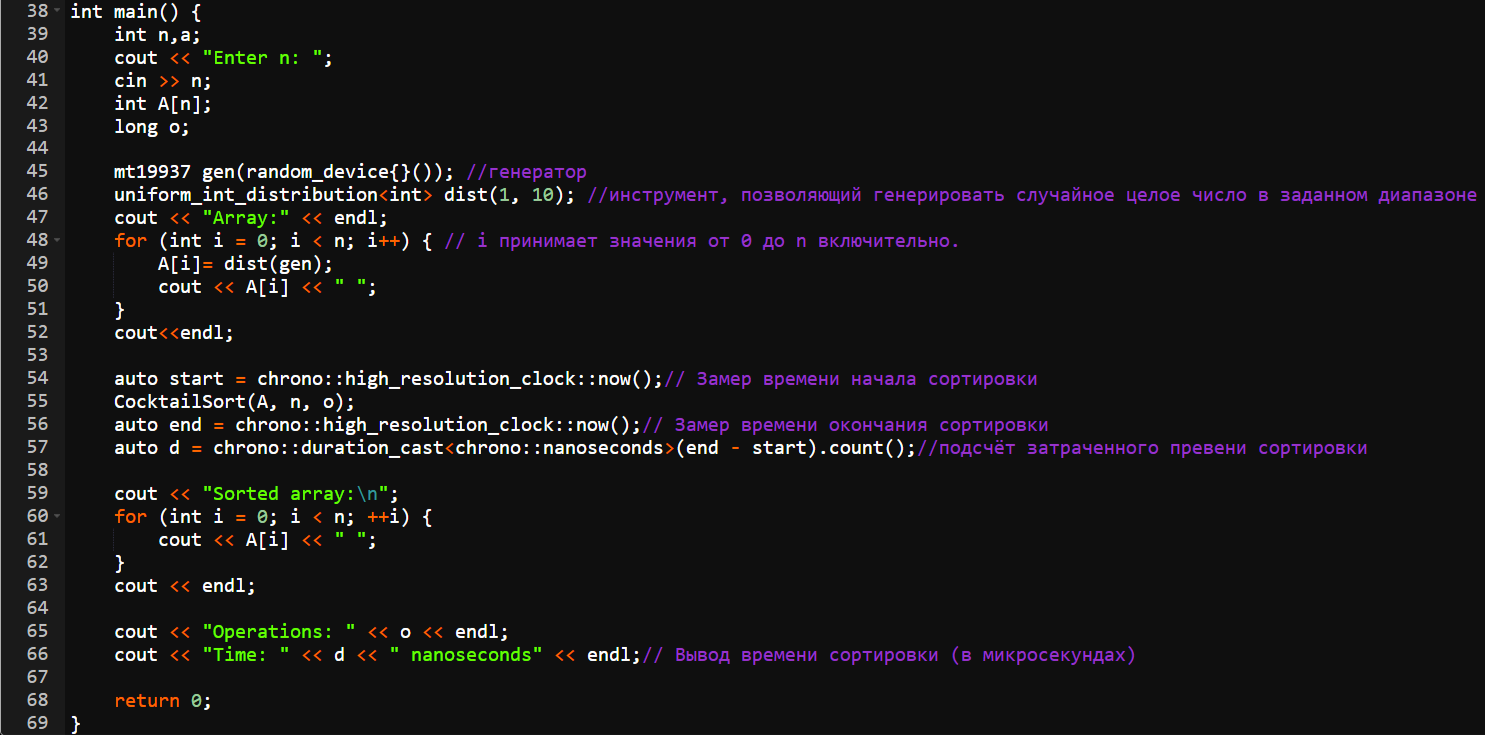


Рисунок 3 – Функция main для алгоритма шейкерной сортировки

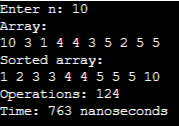


Рисунок 4 – Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.061507 | 12100 |
| 1000 | 7.128173 | 1205538 |
| 10000 | 592.648026 | 120231098 |
| 100000 | 37193.364029 | 12010200916 |
| 1000000 | 3997405.379460 | 120462190275 |

## **2.**5 **Описание выполнения и блок-схема алгоритма простого слияния**

Сортировка простым слиянием (Merge Sort) - это эффективный алгоритм сортировки, который использует принцип "разделяй и властвуй". Он разбивает входной массив на две равные части, рекурсивно сортирует каждую из них, а затем объединяет отсортированные части в одну целую отсортированную последовательность.

Процесс сортировки простым слиянием может быть описан следующим образом:

1. Разделение: Исходный массив делится пополам на две части.
2. Рекурсивная сортировка: Каждая из половинок рекурсивно сортируется с применением того же алгоритма сортировки простым слиянием.
3. Слияние: Отсортированные половинки объединяются в отсортированную целую последовательность.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.5).

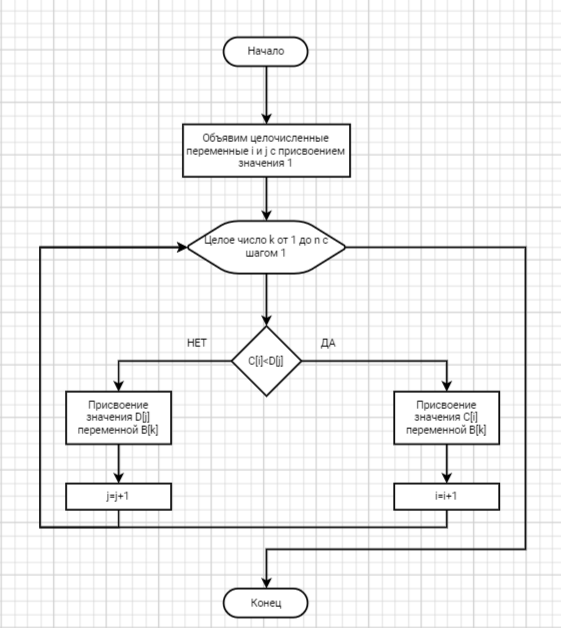


Рисунок 5 – Блок-схема алгоритма простого слияния

## 2.6 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма простого слияния

a. Лучший случай - когда массив уже отсортирован. В этом случае минимальное количество сравнений и перемещений составит O(nlog2n). В среднем случае, когда массив заполнен случайными числами, сложность алгоритма также будет O(nlog2n). В худшем случае, когда массив отсортирован в обратном порядке, количество операций также будет O(n2).

b. Зависимость времени выполнения: Лучший случай: O(nlog2n). Худший случай: O(n2). Для данного метода сортировки время выполнения в худшем случае увеличивается квадратично с увеличением размера входного массива. Поэтому можно использовать квадратичную функцию для описания зависимости времени выполнения данного метода сортировки. Время выполнения в лучшем случае увеличивается почти линейно с ростом размера входного массива. Ёмкостная сложность алгоритма будет составлять O(log2n).

## **2.**7 **Реализация алгоритма на языке C++ и тестирования**

Для реализации данного алгоритма на языке C++ (рис.6,7) понадобятся библиотеки iostream, random и chrono. Для подсчета количества операций присваивания или сравнения введена переменная o типа long. Проведём тестирование программы с заданным размером массива n=10 (рис.8), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование генерация случайных чисел в массиве. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.2. Воспользуемся библиотекой chrono для засекания времени. Для более точных результатов в программе будем рассматривать наносекунды.

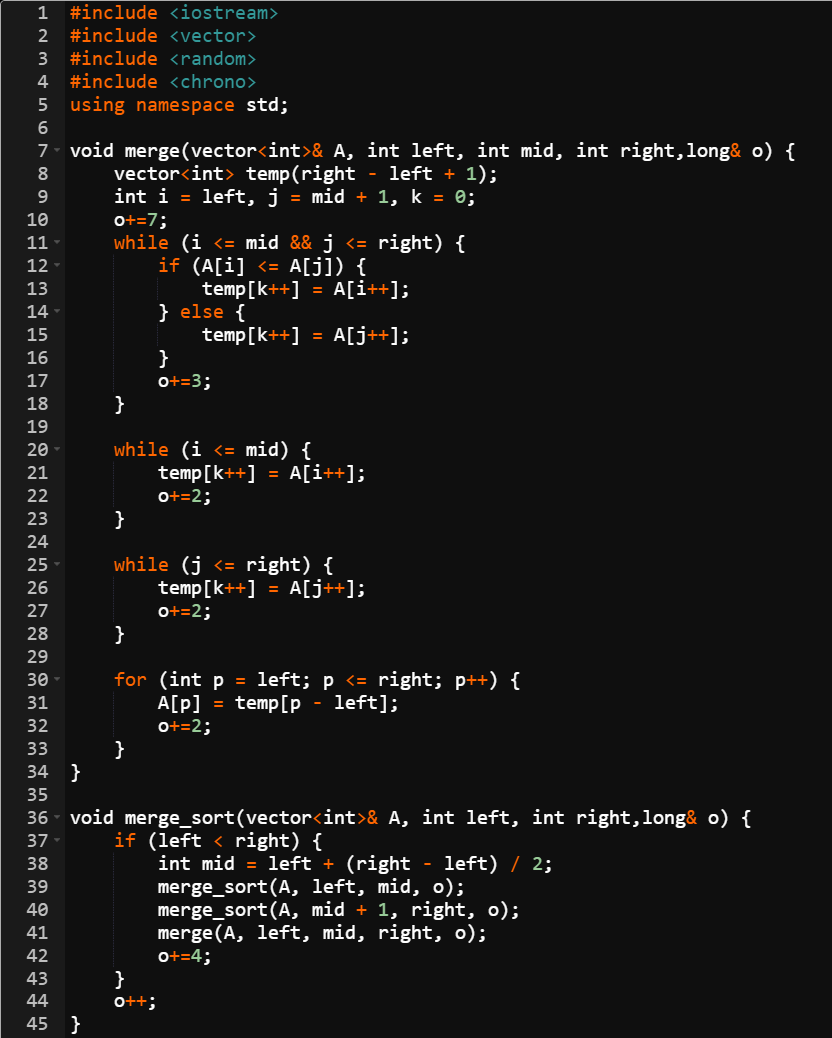


Рисунок 6 – Программа алгоритма простого слияния

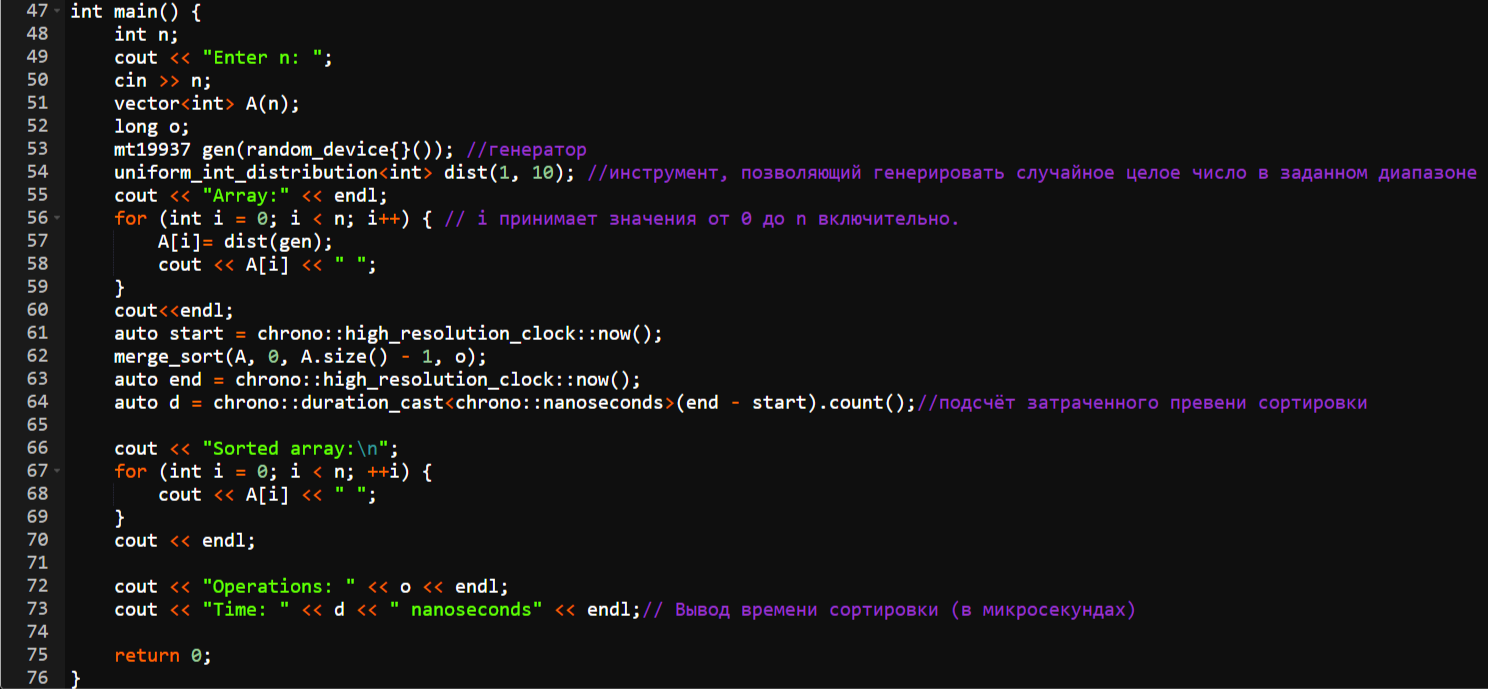


Рисунок 7 – Функция main для алгоритма простого слияния

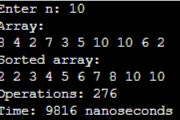


Рисунок 8 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.053726 | 4519 |
| 1000 | 0.683419 | 61399 |
| 10000 | 4.324830 | 781570 |
| 100000 | 44.151263 | 9460234 |
| 1000000 | 473.254131 | 110770871 |

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

Построим два графика для сравнения трёх алгоритмов сортировки. Первый будет построен на значениях до 1000(рис.9), а второй от 10000 и до 1000000(рис.10). Данные для заполнения таблицы будут браться из таблиц 1.1, 1.2, 1.3.

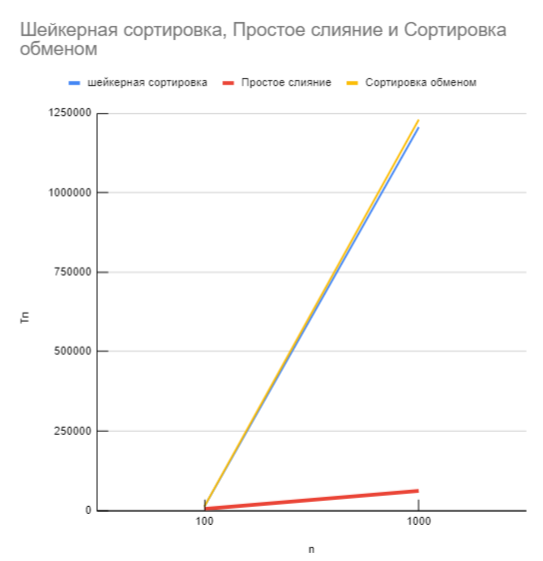


Рисунок 9 - График сравнения трёх сортировок в среднем случае при n до 1000

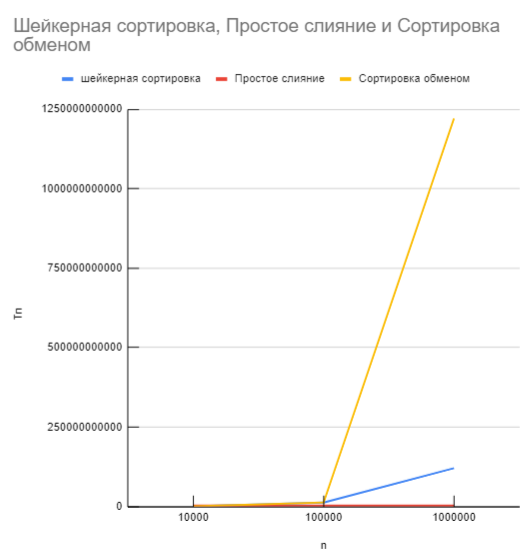


Рисунок 10 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

На основе таблиц 1.1, 1.2 и 1.3, можно сделать вывод, что в среднем случае алгоритм сортировки простого обмена самый неэффективный, алгоритм шейкерной сортировки второй по эффективности, а алгоритм простого слияния самый эффективный.

## **2.**10 **Тестирование программ для алгоритмов шейкерной сортировки и простого слияния**

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

Проведём тестирование программы с заданным размером массива n=10 (рис.12), n=100, n=1000, n=10000, n=100000, n=1000000, которые отсортированы в строго убывающем порядке. Чтобы провести данной тестирование генерация случайных чисел в массиве. Добавим библиотеку algorithm для использования функции sort. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.4. Воспользуемся библиотекой chrono для засекания времени. Алгоритм шейкерной сортировки не изменяется и соответствует продемонстрированному на рисунке 2. Алгоритм функции main продемонстирован на рисунке 11.

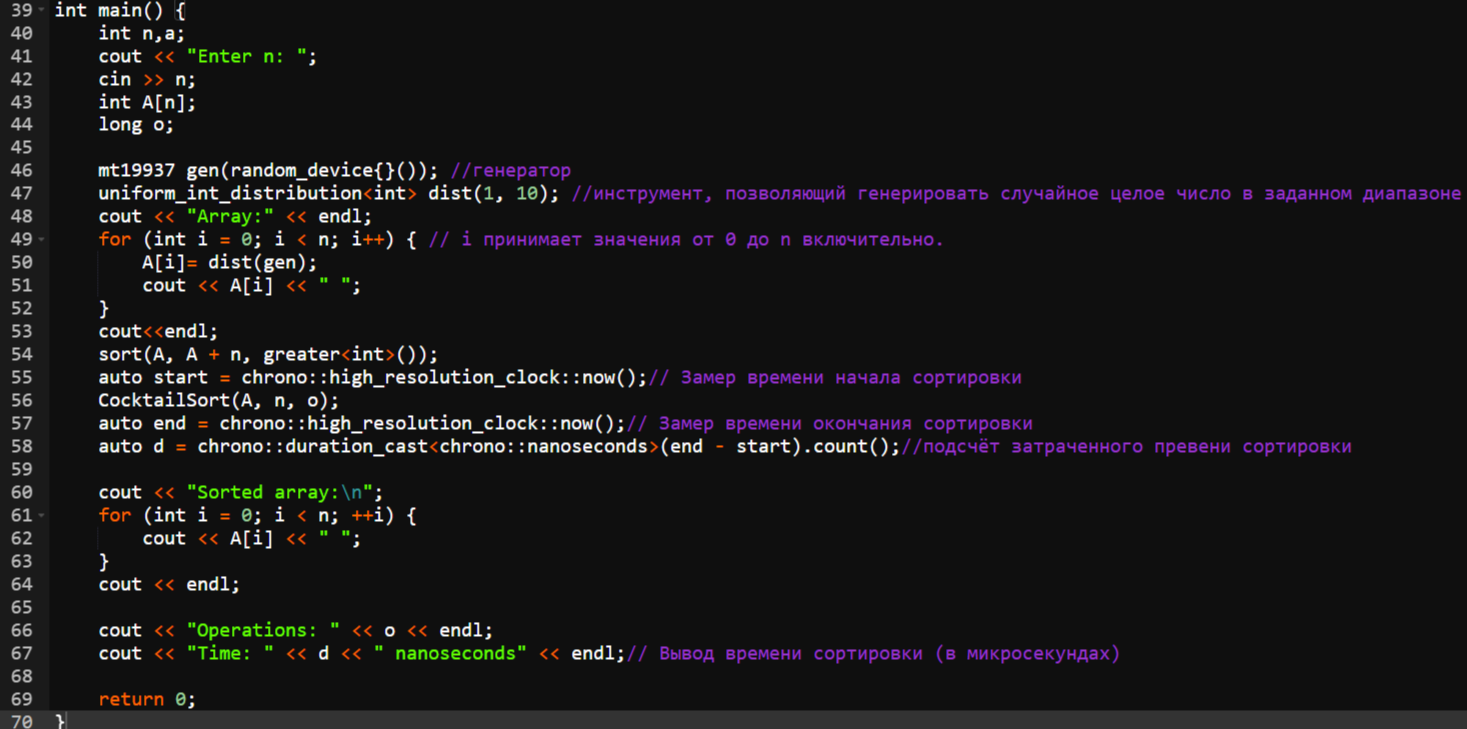


Рисунок 11 – Функция main с отсортированными значениями по убыванию

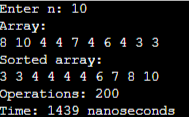


Рисунок 12 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Таблица 1.4. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.094581 | 19122 |
| 1000 | 7.406486 | 1901732 |
| 10000 | 655.830437 | 189964010 |
| 100000 | 48364.435346 | 18999250380 |
| 1000000 | 4604290.880946 | 189999857430 |

### **2.10.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма шейкерной сортировки**

Проведём тестирование программы с заданным размером массива n=10 (рис.14), n=100, n=1000, n=10000, n=100000, n=1000000, которые отсортированы в строго возрастающем порядке. Чтобы провести данной тестирование генерация случайных чисел в массиве. Добавим библиотеку algorithm для использования функции sort. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.4. Воспользуемся библиотекой chrono для засекания времени. Алгоритм шейкерной сортировки не изменяется и соответствует продемонстрированному на рисунке 2. Алгоритм функции main продемонстирован на рисунке 13.

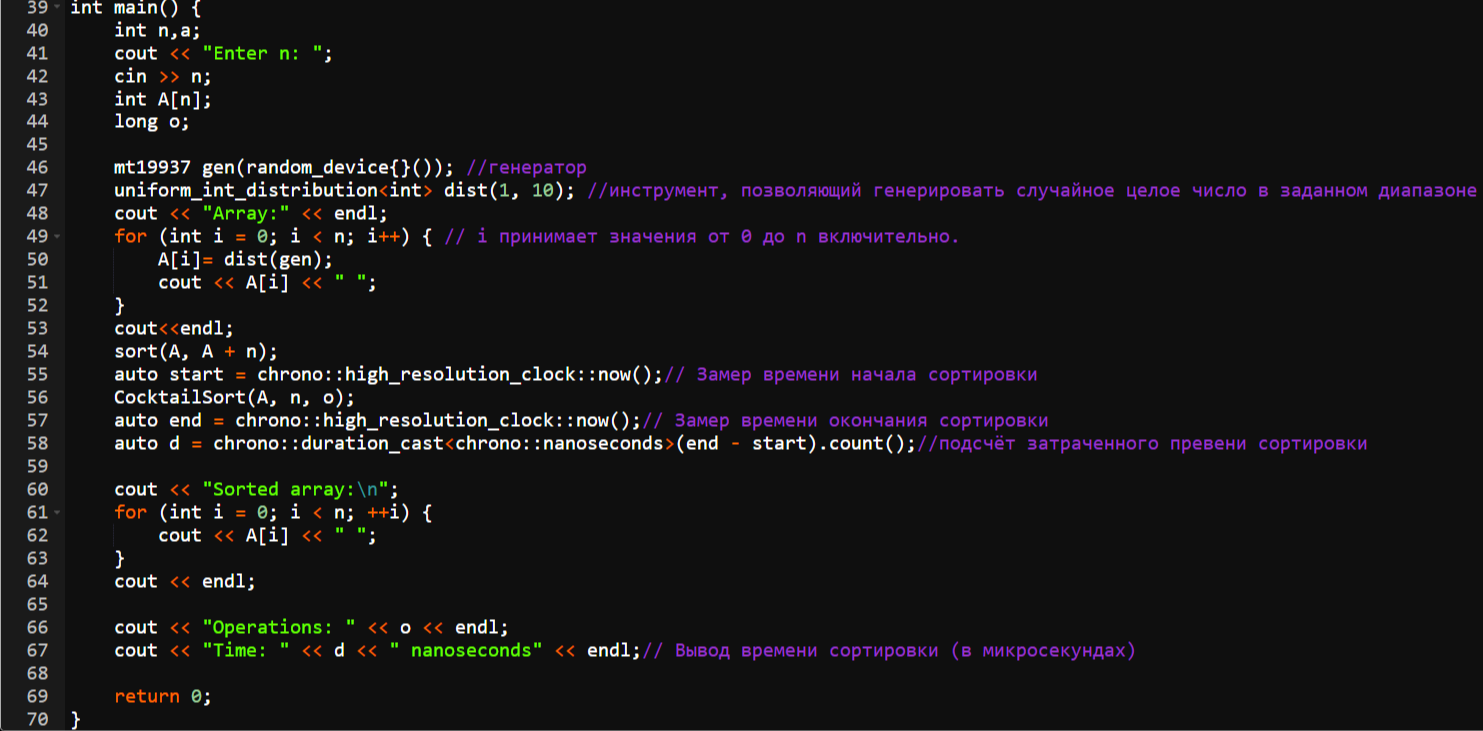


Рисунок 13 – Функция main с отсортированными значениями по возрастанию

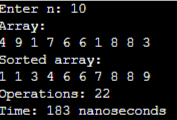


Рисунок 14 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Таблица 1.5. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп=Cп+Mп** |
| --- | --- | --- |
| 100 | 0.001081 | 202 |
| 1000 | 0.003923 | 2002 |
| 10000 | 0.028488 | 20002 |
| 100000 | 0.283075 | 200002 |
| 1000000 | 2.480385 | 2000002 |

### 

### **2.10.3 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма простого слияния**

Проведём тестирование программы с заданным размером массива n=10 (рис.16), n=100, n=1000, n=10000, n=100000, n=1000000, которые отсортированы в строго убывающем порядке. Чтобы провести данной тестирование генерация случайных чисел в массиве. Добавим библиотеку algorithm для использования функции sort. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.4. Воспользуемся библиотекой chrono для засекания времени. Алгоритм простого слияния не изменяется и соответствует продемонстрированному на рисунке 5. Алгоритм функции main продемонстирован на рисунке 15.

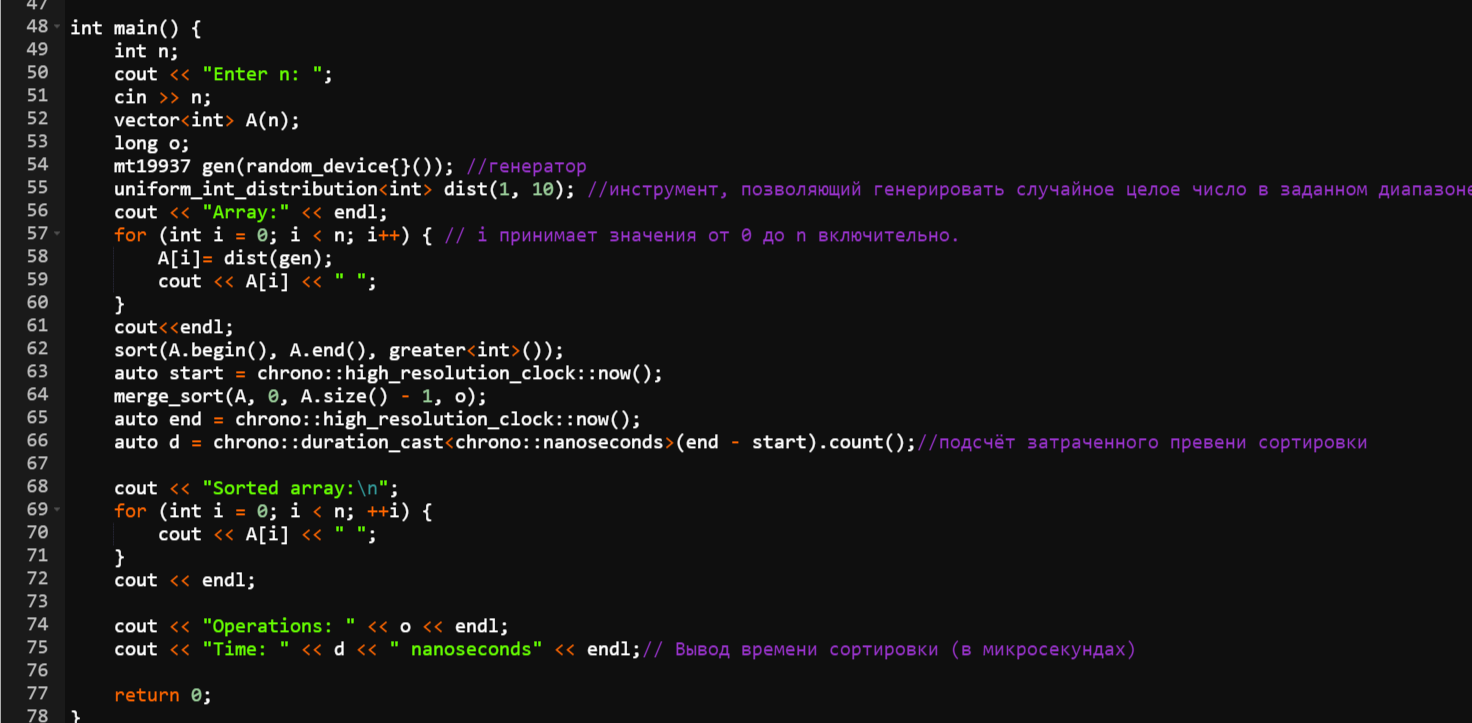


Рисунок 15 – Функция main с отсортированными значениями по убыванию

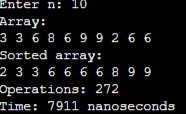


Рисунок 16 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Таблица 1.6. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.0497 | 4399 |
| 1000 | 0.503286 | 58860 |
| 10000 | 3.327009 | 742970 |
| 100000 | 55.254188 | 8927523 |
| 1000000 | 403.46864 | 103684951 |

### **2.10.4 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма простого слияния**

Проведём тестирование программы с заданным размером массива n=10 (рис.18), n=100, n=1000, n=10000, n=100000, n=1000000, которые отсортированы в строго возрастающем порядке. Чтобы провести данной тестирование генерация случайных чисел в массиве. Добавим библиотеку algorithm для использования функции sort. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1.4. Воспользуемся библиотекой chrono для засекания времени. Алгоритм простого слияния не изменяется и соответствует продемонстрированному на рисунке 5. Алгоритм функции main продемонстирован на рисунке 17.

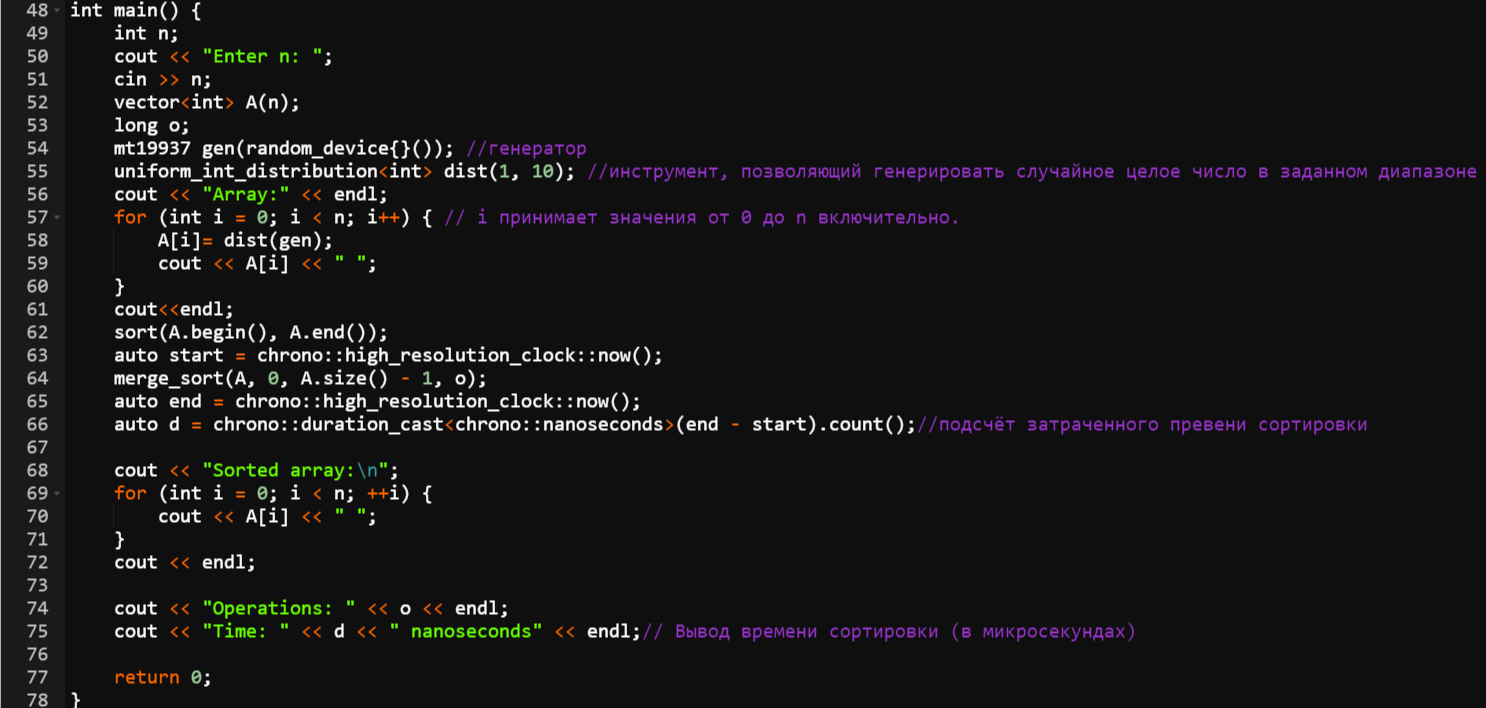


Рисунок 17 –Функция main с отсортированными значениями по возрастанию

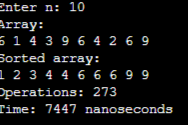


Рисунок 18 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Таблица 1.7. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.046407 | 4332 |
| 1000 | 0.287016 | 57936 |
| 10000 | 6.181374 | 733460 |
| 100000 | 35.985037 | 8829604 |
| 1000000 | 415.762337 | 102872116 |

## **2.**11 **Вывод по заданию №1**

Основываясь на результатах тестирования, представленных в таблицах 1.1,1.2,1.4,1.5,1.6,1.7, можно сделать вывод о зависимости алгоритмов шейкерной сортировки и простого слияния от исходной упорядоченности массива. Алгоритм сортировки слиянием и шейкерная сортировка могут вести себя по-разному в зависимости от упорядоченности исходного массива.

Если исходный массив уже отсортирован, оба алгоритма отработают за оптимальное время. Сортировка слиянием потребует O(nlogn) операций, а шейкерная сортировка (также известная как коктейльная сортировка) потребует O(n) операций, где n - количество элементов в массиве.

Однако, если исходный массив в обратном порядке, то для сортировки слиянием понадобится также O(nlogn) операций, а шейкерная сортировка будет более эффективной и потребует O(n) операций.

Таким образом, можно сделать вывод, что шейкерная сортировка более эффективна для частично упорядоченных массивов, в то время как сортировка слиянием обычно более предпочтительна для сортировки больших массивов.

# 

# 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 простого слияния.

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 - когда массив уже отсортирован, в этом случае количество операций сравнения и перемещения будет минимальным и составит Тт(n)=n. Средний случай - когда массив заполнен случайными числами, в этом случае алгоритм будет иметь сложность Тт(n)=(n2-n)/2. Худший случай - когда массив отсортирован в обратном порядке, количество операций в этом случае также будет Тт(n)=3\*(n2-n)/2.

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

Оценка вычислительной сложности для худшего случая алгоритма сортировки обменом в О-нотации составляет O(n2). Для лучшего случая оценка в Ω-нотации равна Ω(n). Алгоритм сортировки обменом имеет оценку вычислительной сложности в среднем случае равную θ(n2). Ёмкостная сложность алгоритма сортировки обменом составляет 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.19).

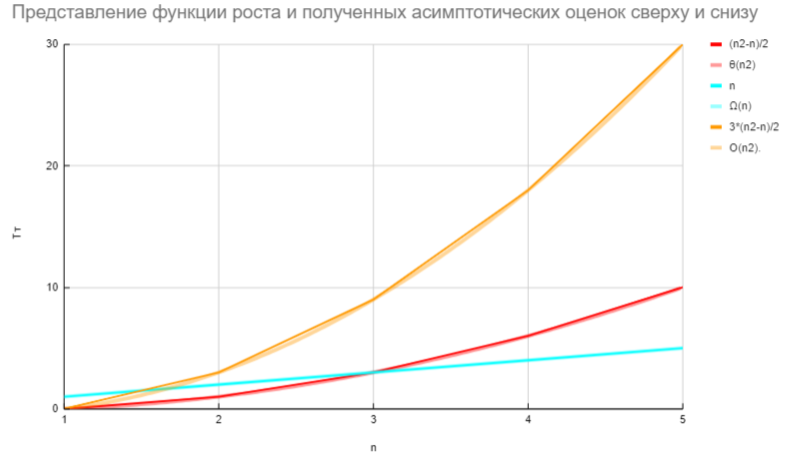


Рисунок 19 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и простого слияния**

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

Алгоритм шейкерной сортировки имеет квадратичную вычислительную сложность в худшем случае (O(n2)) и линейную сложность в лучшем случае (Ω(n)). Для среднего случая сложность алгоритма составляет θ(n2). Ёмкостная сложность алгоритма шейкерной сортировки равна O(1).

### **3.5.2 Справочная информация о вычислительной сложности алгоритма простого слияния**

Оценка вычислительной сложности алгоритма простого слияния для наихудшего случая составляет O(nlog2n), для лучшего случая - Ω(nlog2n), а для среднего случая - θ(nlog2n). Ёмкостная сложность алгоритма простого слияния O(log2n).

## **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 выбором, шейкерной сортировки и простой сортировки слиянием.

Таблица 2. Сводная таблица результатов

| Алгоритм | Асимптотическая сложность алгоритма | | | |
| --- | --- | --- | --- | --- |
| Наихудший случай (сверху) | Наилучший случай (снизу) | Средний случай (точная оценка) | Ёмкостная сложность |
| Простой выбор | О(n2) | Ω(n) | θ(n2) | О(1) |
| Шейкерная сортировка | О(n2) | Ω(n) | θ(n2) | О(1) |
| Сортировка простым слиянием | О(nlog2n) | Ω(nlog2n) | θ(nlog2n) | О(n) |

**3.7 Выводы по заданию №2**

На основе таблицы 2, можно сделать вывод, что наиболее эффективным алгоритмом в лучшем случае является шейкерная сортировка и простым обменом. В среднем и худшем случае наиболее эффективным алгоритмом является простое слияние, асимптотическая сложность которого составляет θ(nlog2n), что является лучшим показателем среди представленных алгоритмов. Шейкерная сортировка и сортировка простым обменом имеют худшую асимптотическую сложность в худшем и среднем случае, поэтому они могут быть менее эффективными на больших объемах данных. Таким образом, при выборе наиболее эффективного алгоритма сортировки стоит отдавать предпочтение сортировке простым слиянием.

# 5 ВЫВОДЫ

В ходе выполнения работы были достигнуты следующие результаты:

* Овладение навыками анализа вычислительной сложности алгоритмов сортировки и выбора наиболее эффективного варианта;
* Исследование алгоритмов шейкерной сортировки и простого слияния;
* Разработка программных реализаций алгоритмов шейкерной сортировки и простого слияния;
* Проведение тестирования программ для указанных алгоритмов;
* Построение графиков функции роста временной сложности алгоритмов;
* Сравнение алгоритмов простой сортировки обменом, шейкерной сортировки и простого слияния;
* Анализ асимптотической сложности указанных алгоритмов;
* Сравнение асимптотической сложности алгоритмов сортировки;
* Выявление наиболее эффективного алгоритма сортировки.

Таким образом, основная цель работы, заключающаяся в получении навыков анализа вычислительной сложности алгоритмов сортировки и выбора оптимального метода, была успешно достигнута.

# 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).