|  |
| --- |
|  |
| 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: Зернов Н.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](#_2et92p0)

[2.2.1 Описание выполнения и блок-схема алгоритма Шелла со смещениями Д. Кнута. Способ 2 6](#_tyjcwt)

[2.2.2 Доказательство корректности циклов алгоритма Шелла со смещениями Д. Кнута. Способ 2 7](#_3dy6vkm)

[2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма Шелла со смещениями Д. Кнута. Способ 2 8](#_1t3h5sf)

2.3 Реализация алгоритма на языке C++, проведение тестирования и построение графика 9

[2.3.1 Реализация алгоритма Шелла со смещениями Д. Кнута. Способ 2 сортировки на языке C++ 9](#_4d34og8)

[2.3.2 Тестирование и построение графика 11](#_2s8eyo1)

[2.4 Математическая модель решения алгоритма 13](#_17dp8vu)

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

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

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

[2.5 Реализация алгоритма на языке C++, проведение тестирования и построение графика 17](#_35nkun2)

[2.5.1 Реализация алгоритма быстрой сортировки(Хоара) на языке C++ 17](#_1ksv4uv)

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

[2.6 Сортировка простым обменом 20](#_2jxsxqh)

[2.7 Сравнение трёх алгоритмов на графике 21](#_z337ya)

[2.8 Тестирование программ для алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 22](#_3j2qqm3)

[2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма шейкерной сортировки 22](#_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 36](#_1ci93xb)

[3 ЗАДАНИЕ №2 38](#_2bn6wsx)

[3.1 Формулировка задачи 38](#_qsh70q)

[3.2 Формулы функции роста алгоритма сортировки простым обменом в худшем и лучшем случае 38](#_3as4poj)

3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки обменом 39

[3.4 Графическое представление функции роста и полученных асимптотических оценок сверху и снизу 39](#_1pxezwc)

[3.5 Справочная информация о вычислительной сложности алгоритмов шейкерной сортировки и быстрой сортировки(Хоара) 40](#_49x2ik5)

[3.5.1 Справочная информация о вычислительной сложности алгоритма шейкерной сортировки 40](#_2p2csry)

[3.5.2 Справочная информация о вычислительной сложности алгоритма быстрой сортировки(Хоара) 40](#_147n2zr)

[3.6 Таблица асимптотической сложности трёх алгоритмов 41](#_3o7alnk)

[4 К ВЫВОДЫ 42](#_23ckvvd)

[5 ЛИТЕРАТУРА 43](#_32hioqz)

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи (Вариант 13)**

Эмпирическая оценка эффективности алгоритмов.

1. Разработать алгоритм Шелла со смещениями Д. Кнута. Способ 2, 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 Д. Кнута. Способ 2.

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 Шелла со смещениями Д. Кнута. Способ 2**

Алгоритм сортировки Шелла со смещениями Дональда Кнута Способ 2, представляет собой модификацию сортировки Шелла, где смещения для сортировки последовательности определяются особым образом.

Описание алгоритма:

1. Генерация последовательности смещений Дональда Кнута:

- Начнем с выбора начального смещения (шага) h, равного единице.

- Далее, на каждой итерации увеличиваем h по формуле h = 3 \* h + 1 пока h не превысит длину сортируемого массива.

- Этот способ генерации последовательности смещений обеспечивает оптимальные результаты в сортировке Шелла.

2. Сортировка массива:

- Начинаем сортировку, используя сгенерированные смещения.

- Для каждого смещения h, начиная с максимального значения и заканчивая единицей:

- Применяем вставочную сортировку к подмассивам элементов, расположенным на расстоянии h друг от друга.

- При этом сравниваем и переставляем элементы таким образом, чтобы каждый элемент был на своем месте относительно элементов на h позиций влево и вправо от него.

- Этот процесс выполняется для всех подмассивов с одним и тем же смещением h.

- Повторяем этот процесс для каждого уменьшающегося значения h, пока h не станет равным единице.

3. Завершение сортировки после выполнения сортировки для каждого смещения h, весь массив будет упорядочен.

Этот алгоритм сочетает в себе преимущества сортировки Шелла с оптимальным выбором смещений, что позволяет достичь более эффективной сортировки по сравнению с классической сортировкой Шелла.

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

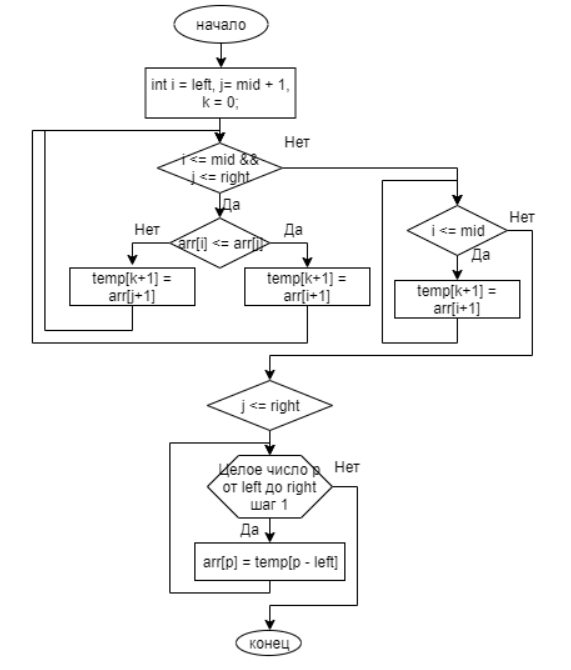


Рисунок 1 – Блок-схема алгоритма Шелла со смещениями Д. Кнута. Способ 2

### **2.2.2 Доказательство корректности циклов алгоритма Шелла со смещениями Д. Кнута. Способ 2**

Доказательство корректности алгоритма сортировки Шелла со смещениями Д. Кнута. Способ 2 требует доказательства двух ключевых аспектов: сначала, что массив действительно становится отсортированным после прохождения всех циклов, и второго, что выбор смещений Дональда Кнута обеспечивает эффективную сортировку.

1. Корректность сортировки. Для доказательства корректности сортировки Шелла со смещениями Д. Кнута, можно применить индукцию по количеству элементов массива. Базовый случай: при одном элементе массив уже отсортирован. Предположение индукции: пусть алгоритм корректно сортирует массивы длиной до k. Шаг индукции: покажем, что алгоритм сортировки Шелла справляется с массивом длиной k + 1. После прохождения всех циклов сортировки, каждый элемент будет находиться на своем месте относительно элементов, находящихся на расстоянии смещения друг от друга. После последнего прохода с наименьшим смещением, алгоритм выполняет сортировку вставками, завершая сортировку всего массива. Таким образом, по индукции доказывается корректность сортировки.

2. Эффективность смещений Д. Кнута. Для доказательства эффективности выбора смещений Дональда Кнута, можно использовать теоретический анализ. Смещения Д. Кнута h = 1, 4, 13, 40, ... обладают свойством, что они взаимно просты между собой. Это позволяет алгоритму перемещать элементы массива на короткие расстояния в начале сортировки, а затем на более длинные расстояния, приближая элементы к своим итоговым позициям.

Экспериментальные и теоретические исследования подтверждают, что смещения Дональда Кнута обеспечивают достаточно эффективную сортировку в среднем случае.

Таким образом, доказательство корректности алгоритма Шелла со смещениями Дональда Кнута (способ 2) включает в себя доказательство корректности сортировки и эффективности выбора смещений Дональда Кнута.

### **2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма Шелла со смещениями Д. Кнута. Способ 2**

1. Лучший случай возникает, когда массив уже отсортирован. В этом случае, используемые смещения Д. Кнута обеспечивают эффективную сортировку, расставляя элементы в правильном порядке. Время выполнения в лучшем случае составляет примерно O(n\* log2n), где n - количество элементов в массиве.

Средний случай сортировки Шелла со смещениями Д. Кнута сложно определить точно, так как он зависит от конкретных данных и выбора смещений. В среднем, алгоритм обычно имеет временную сложность от O(n^1.3) до O(n^2). Это может быть оптимальный выбор, если последовательность смещений правильно подобрана.

2. Худший случай наступает, когда массив уже отсортирован в обратном порядке. В этом случае, смещения Д. Кнута не обеспечивают эффективное уменьшение количества инверсий. Время выполнения будет приблизительно O(n^2), что делает алгоритм менее эффективным.

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 Шелла со смещениями Д. Кнута. Способ 2 на языке C++**

Реализуем данный алгоритм на языке C++(рис.2,3,4). Для реализации понадобятся такие библиотеки, как iostream, vector и random, chrono. Iostream — это заголовочный файл с классами, функциями и переменными для организации потока ввода и вывода в языке программирования C++. Vector — это шаблон класса для контейнеров последовательности. Вектор хранит элементы заданного типа в линейном расположении и обеспечивает быстрый случайный доступ к любому элементу. 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\_counter, которая представляет собой целое число в диапазоне от -9223372036854775807 до 9223372036854775807 и занимает 8 байта в памяти.

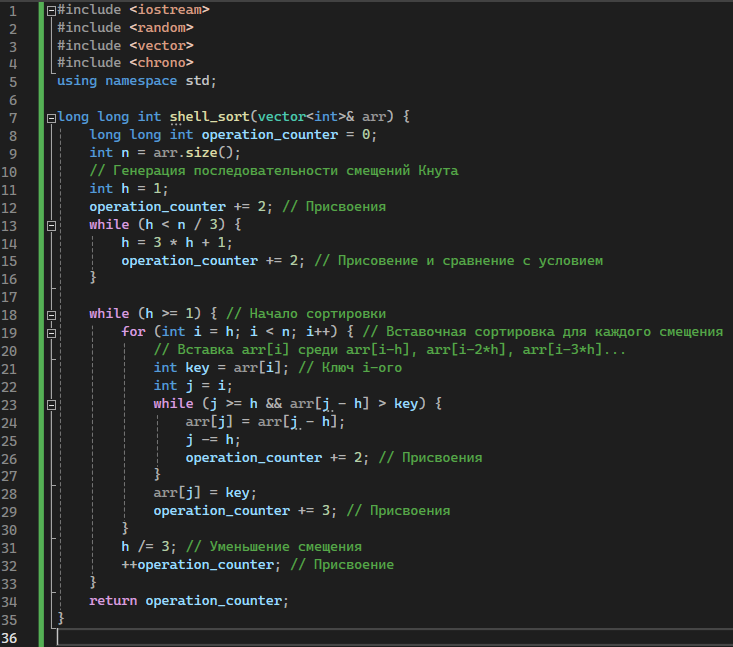


Рисунок 2 – Программа алгоритма Шелла со смещениями Д. Кнута. Способ 2

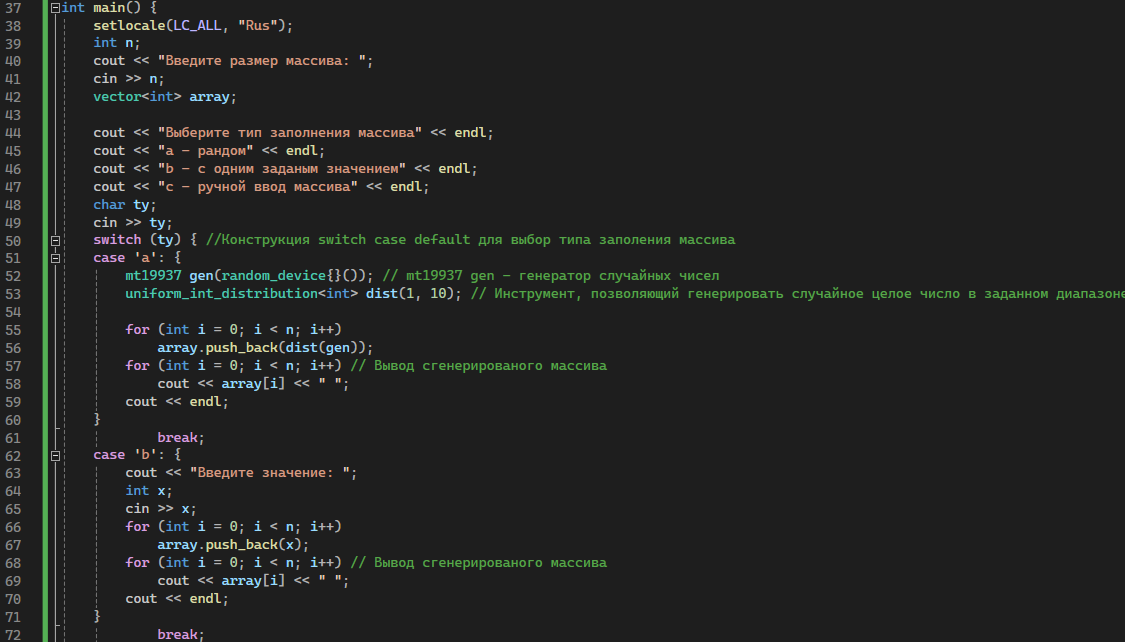


Рисунок 3 – Функция main для алгоритма Шелла со смещениями Д. Кнута. Способ 2

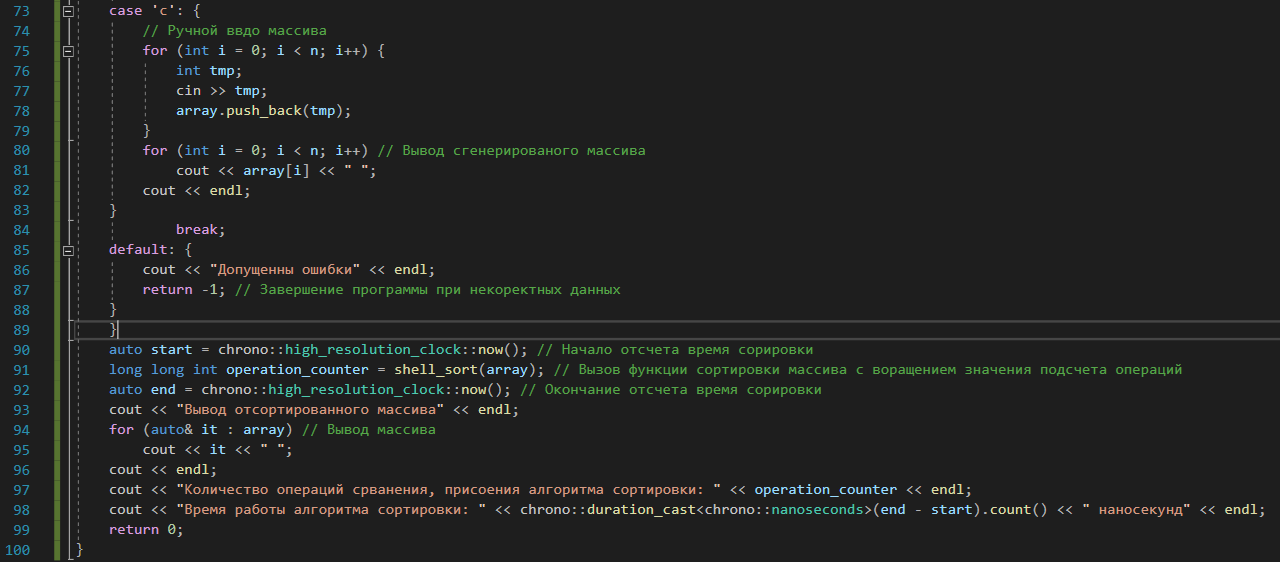


Рисунок 4 – Функция main для алгоритма Шелла со смещениями Д. Кнута. Способ 2

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

Стоит задача протестировать программу с заданным размером массива n = 10 (рис.5), 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 наносекунды, которые в дальнейшем, для заполнения таблицы, переведем в миллисекунды.

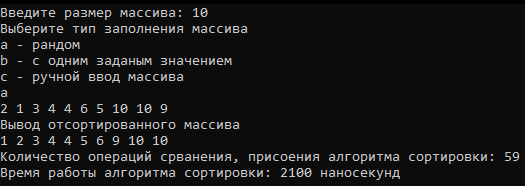


Рисунок 5 - Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.0455 | 1628 |
| 1000 | 0.6453 | 23549 |
| 10000 | 7.9326 | 299694 |
| 100000 | 705,5074 | 3662329 |
| 1000000 | 1973,6825 | 43151612 |

На основе полученных данных, продемонстрированных в таблице 1.1, построим график функции роста Тп алгоритма Шелла со смещениями Д. Кнута. Способ 2 в среднем случае от размера массива n (рис.6).



Рисунок 6 - График функции роста Тп алгоритма Шелла со смещениями Д. Кнута. Способ 2 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. Краткая суть алгоритма 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. Из массива выбирается опорный элемент. Этот элемент будет использоваться для разделения массива на две части.

2. (Разделение массива) массив переупорядочивается таким образом, что все элементы, меньшие опорного, помещаются перед ним, а все элементы, большие опорного, помещаются после него. В результате этого шага опорный элемент находится на своём окончательном месте в отсортированном массиве.

3. После разделения массива на две части, рекурсивно вызывается алгоритм быстрой сортировки для каждой из них.

4. Когда размер подмассивов становится равным 0 или 1, рекурсия завершается, и массив считается отсортированным. В этом случае дополнительных действий не требуется.

Важно правильно выбирать опорный элемент для эффективной работы алгоритма. Обычно выбирают либо первый, либо последний элемент массива, либо элемент из середины массива. Иногда применяются более сложные методы выбора опорного элемента, такие как случайный выбор или выбор медианы из трёх.

Время работы алгоритма быстрой сортировки в среднем составляет O(n log2n), что делает его очень эффективным для сортировки больших массивов данных. Однако, в худшем случае (когда опорный элемент плохо выбирается и каждый разделенный массив оказывается почти пустым или почти полным), время работы может составлять O(n^2), что делает его менее эффективным. Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.6).

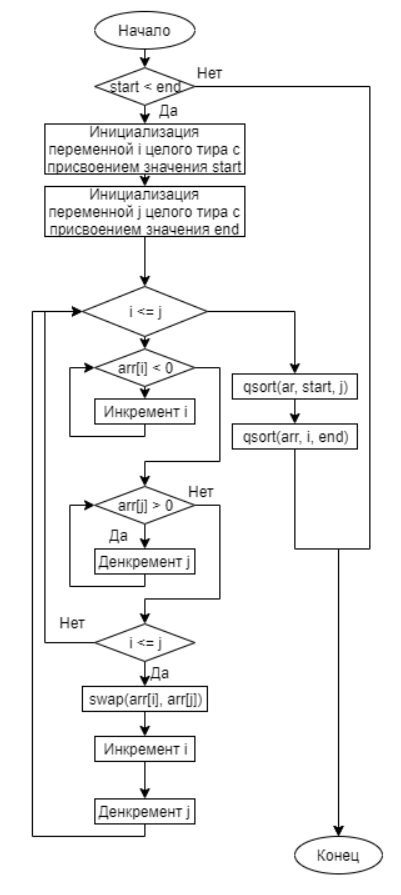


Рисунок 6 – Блок-схема подпрограммы обмена элементов

### **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 arr[i] всегда меньше 0. Инвариант для внутреннего цикла: значение переменной arr[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 arr, start, j), либо i увеличивается (в случае рекурсии с параметрами arr, 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)**

1. Лучший случай - массив уже отсортирован. В этом случае количество операций сравнения и перемещения будет минимальным и будет составлять O(nlog2n).

Средний случай - массив заполнен случайными числами. В этом случае алгоритм будет иметь сложность O(nlog2n).

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

2. Функции роста времени:

Лучший случай: O(nlog2n).

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(log2n).

## **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.7). Для реализации понадобятся такие библиотеки, как iostream, vector и random, chrono. Iostream — это заголовочный файл с классами, функциями и переменными для организации потока ввода и вывода в языке программирования C++. Vector — это шаблон класса для контейнеров последовательности. Вектор хранит элементы заданного типа в линейном расположении и обеспечивает быстрый случайный доступ к любому элементу. 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\_counter, которая представляет собой целое число в диапазоне от -9223372036854775807 до 9223372036854775807 и занимает 8 байта в памяти.

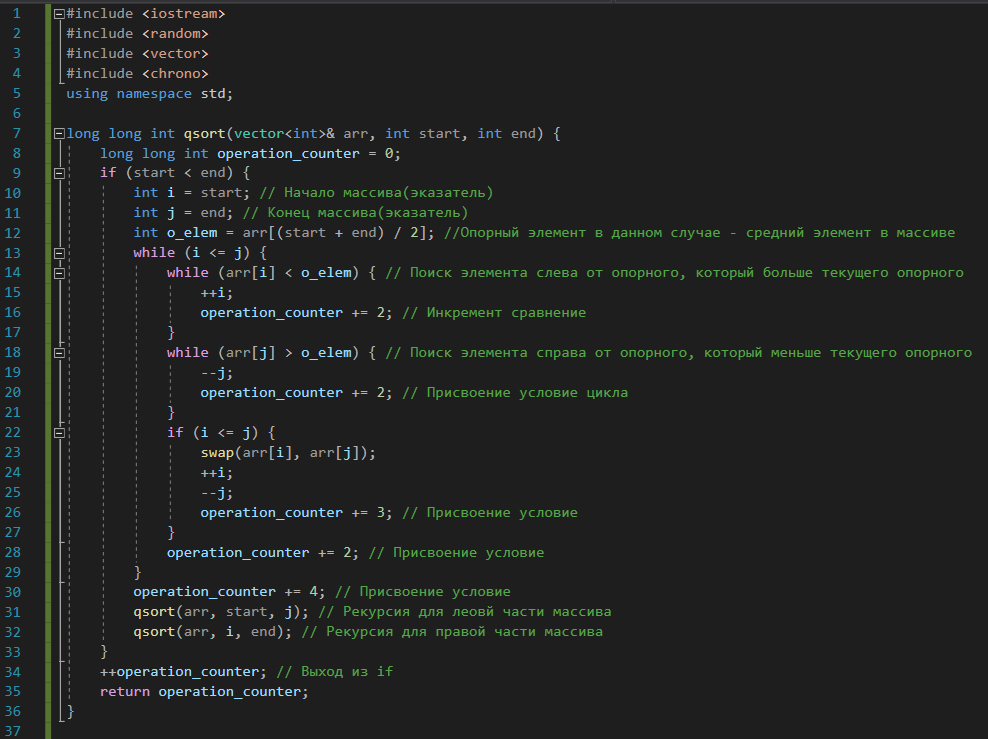


Рисунок 7 – Программа алгоритма быстрой сортировки (Хоара)

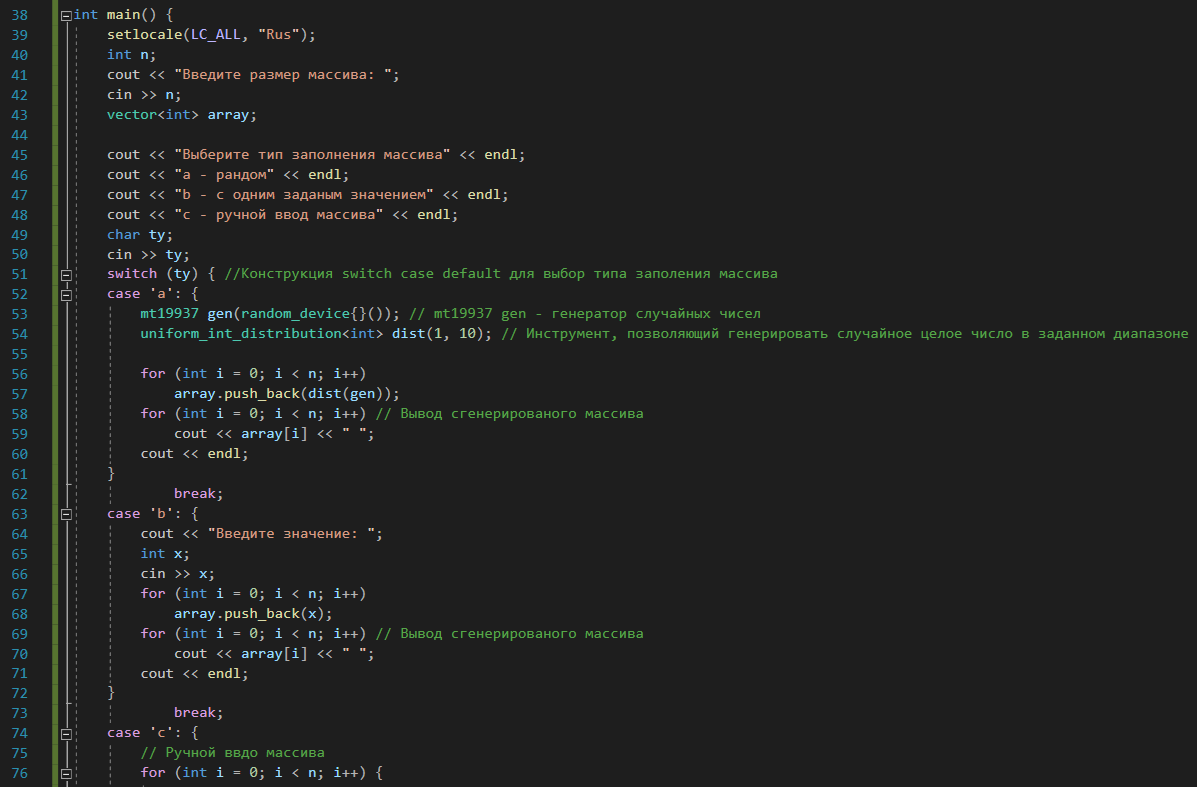


Рисунок 8 – Функция main для алгоритма быстрой сортировки (Хоара)

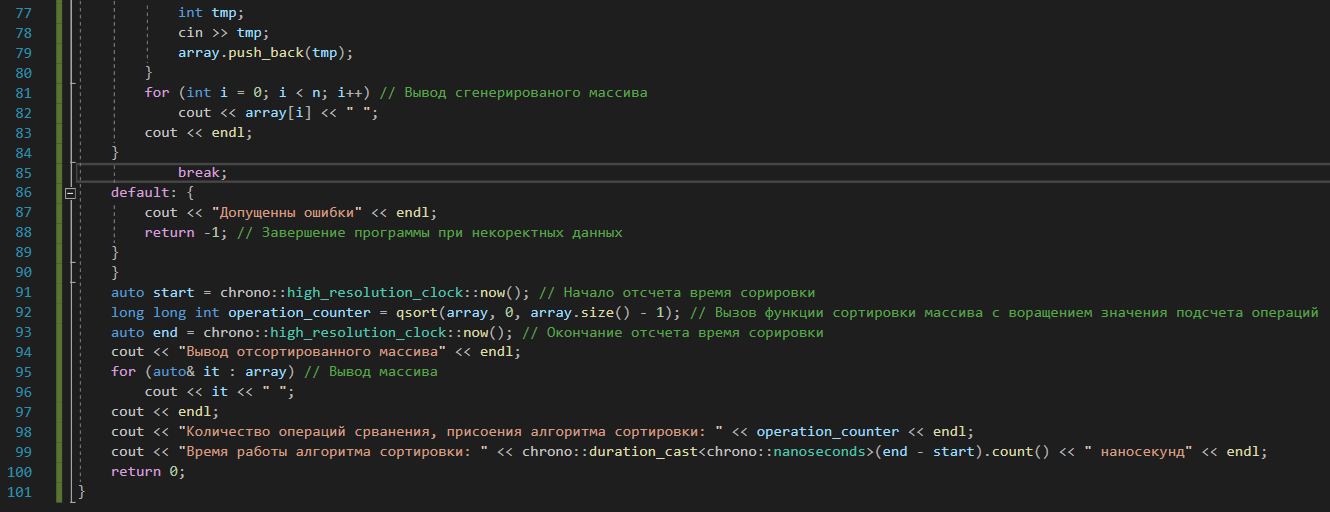


Рисунок 9 – Функция main для алгоритма быстрой сортировки (Хоара)

### **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.

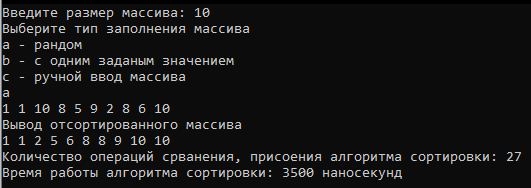


Рисунок 10 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| 100 | 0.0539 | 232 |
| 1000 | 0.8187 | 2224 |
| 10000 | 10.8336 | 22722 |
| 100000 | 137.5897 | 221876 |
| 1000000 | 1653.4884 | 2091171 |

На основе данных из таблицы 1.2 построим график функции роста Тп алгоритма быстрой сортировки (Хоара) в среднем случае от размера массива n(рис.11).



Рисунок 11 - График функции роста Тп алгоритма быстрой сортировки от размера массива n

## **2.6 Сортировка простым выбором**

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 | 0.3425 | 5519 |
| 1000 | 33.2711 | 505401 |
| 10000 | 3287.6379 | 50054360 |
| 100000 | 36577.144653 | 5000542269 |
| 1000000 | 3953527.144653 | 50000543619 |

## **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 Шелла со смещениями Д. Кнута. Способ 2, быстрой сортировки (Хоара) и сортировки простым выбором в среднем случае от размера массива n. Для наглядности сравнения построим два графика. Первый будет построен на значениях до 1000(рис.12), а второй от 10000 и до 1000000(рис.13). Это позволит нам сделать более точное сравнение.

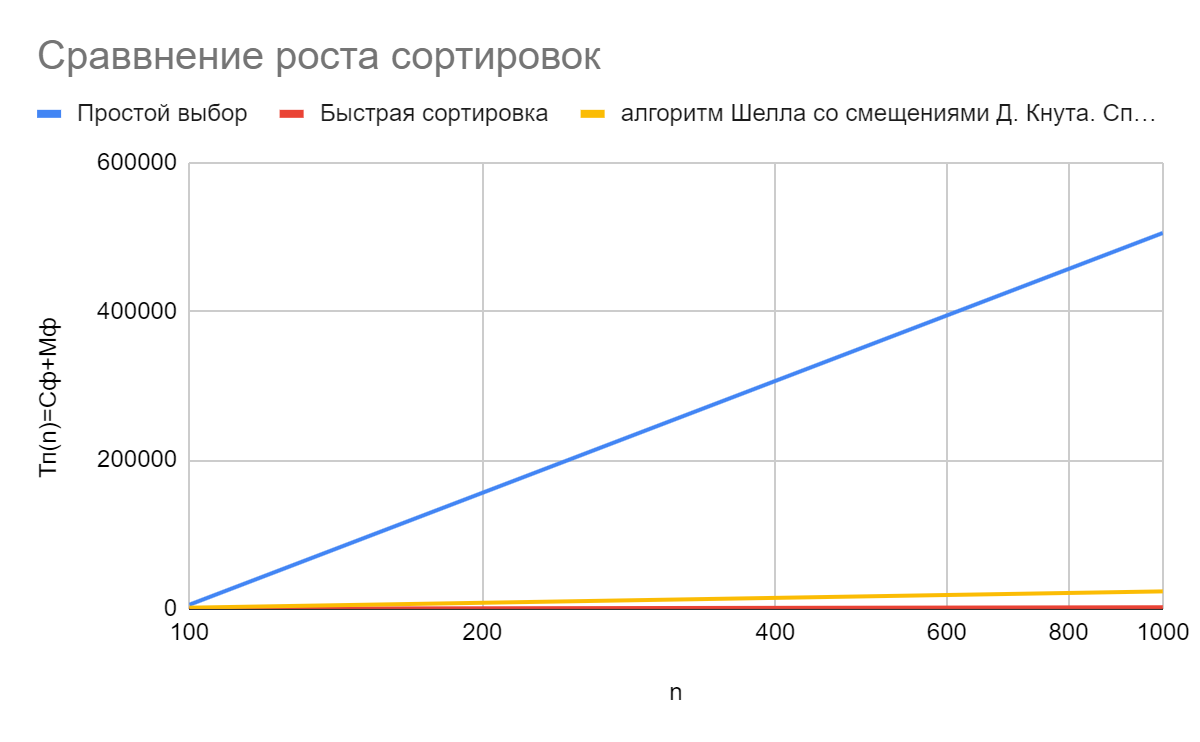


Рисунок 12 - График сравнения трёх сортировок в среднем случае при n до 1000

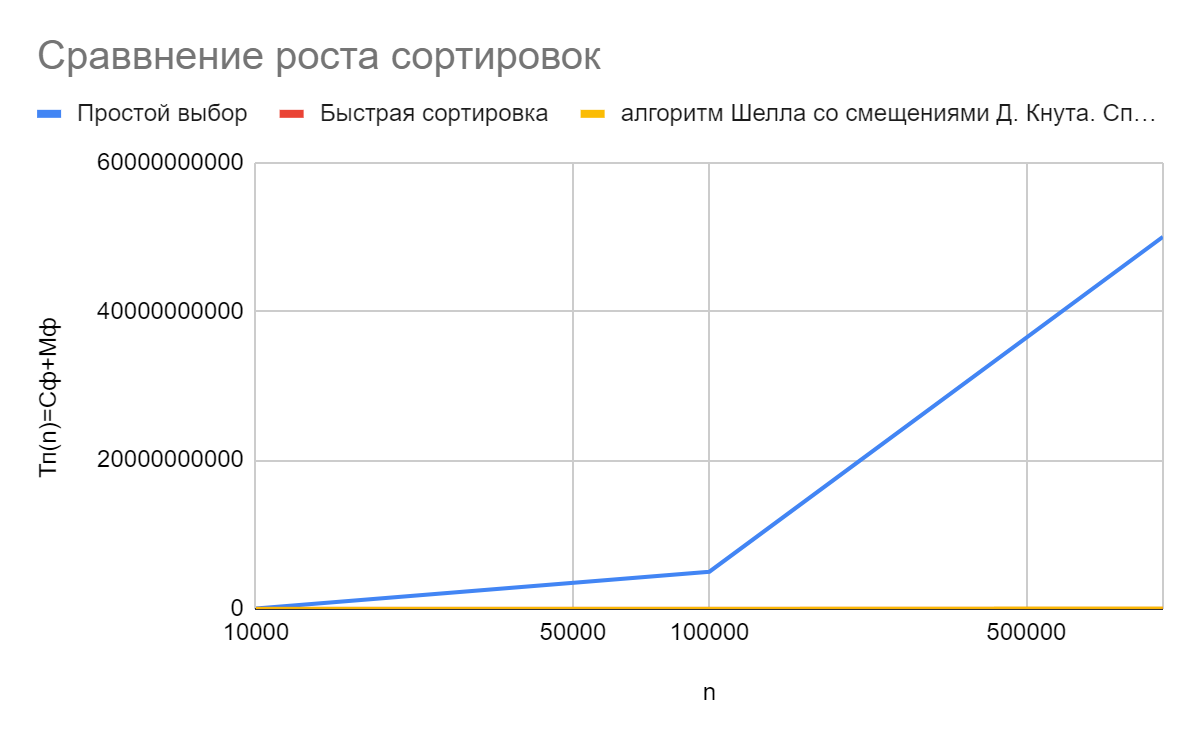


Рисунок 13 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

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 выбора самый неэффективный, алгоритм Шелла со смещениями Д. Кнута. Способ 2 сортировки второй по эффективности, а алгоритм быстрой сортировки (Хоара) самый эффективный.

## **2.8 Тестирование программ для алгоритмов Шелла со смещениями Д. Кнута, Способ 2 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 Шелла со смещениями Д. Кнута, Способ 2 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.14). В функцию main добавим вызов функции сортировки по убыванию (рис.15) и продемонстрируем работу программы при n=10 (рис.17). Алгоритм Шелла со смещениями Д. Кнута. Способ 2 сортировки не изменяется и соответствует продемонстрированному на рисунке 5.

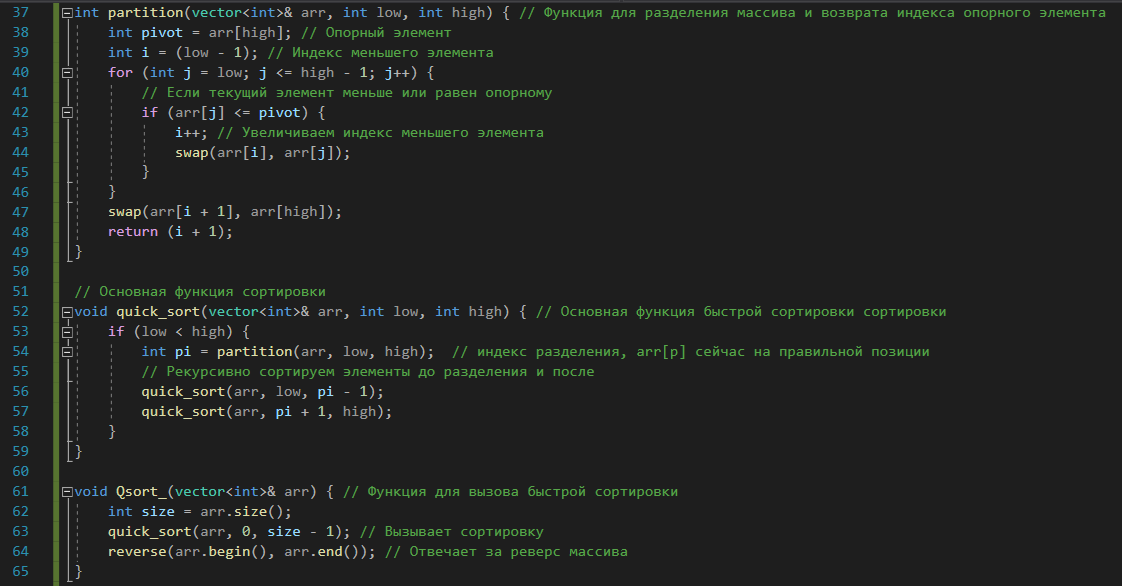


Рисунок 14 – Алгоритм сортировки по убыванию

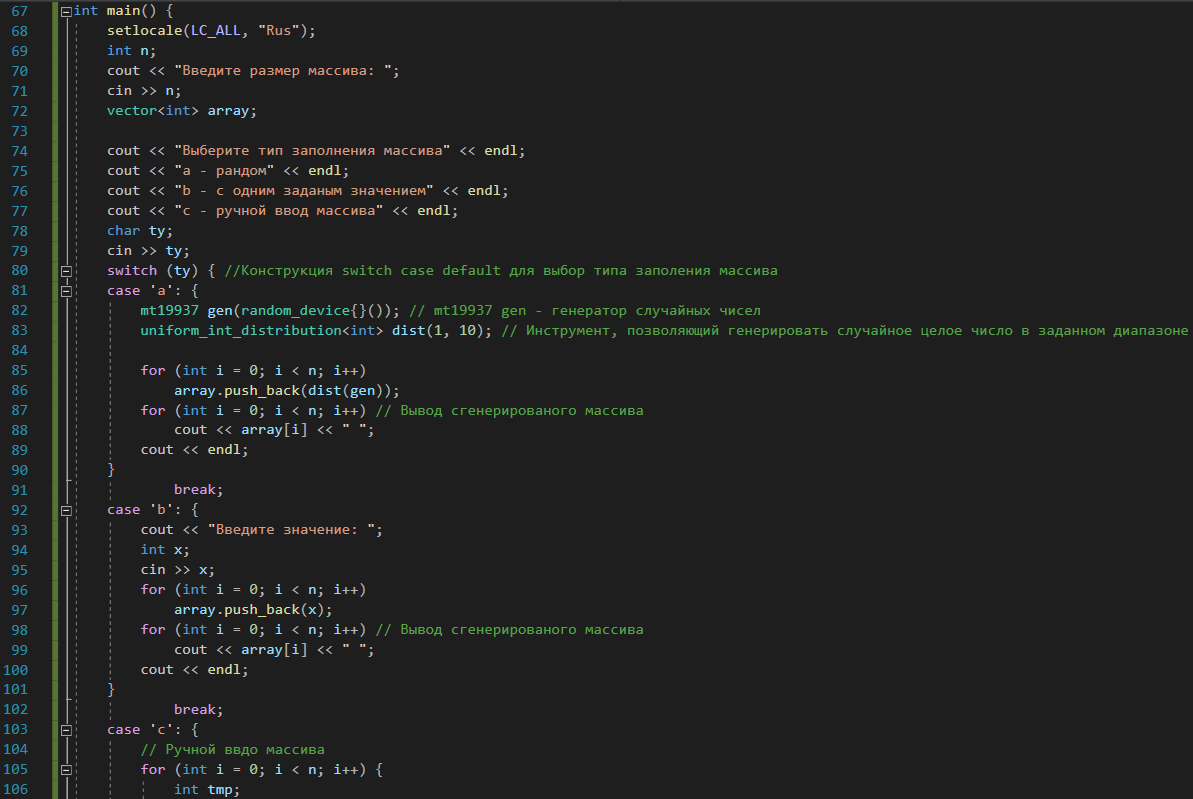


Рисунок 15 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

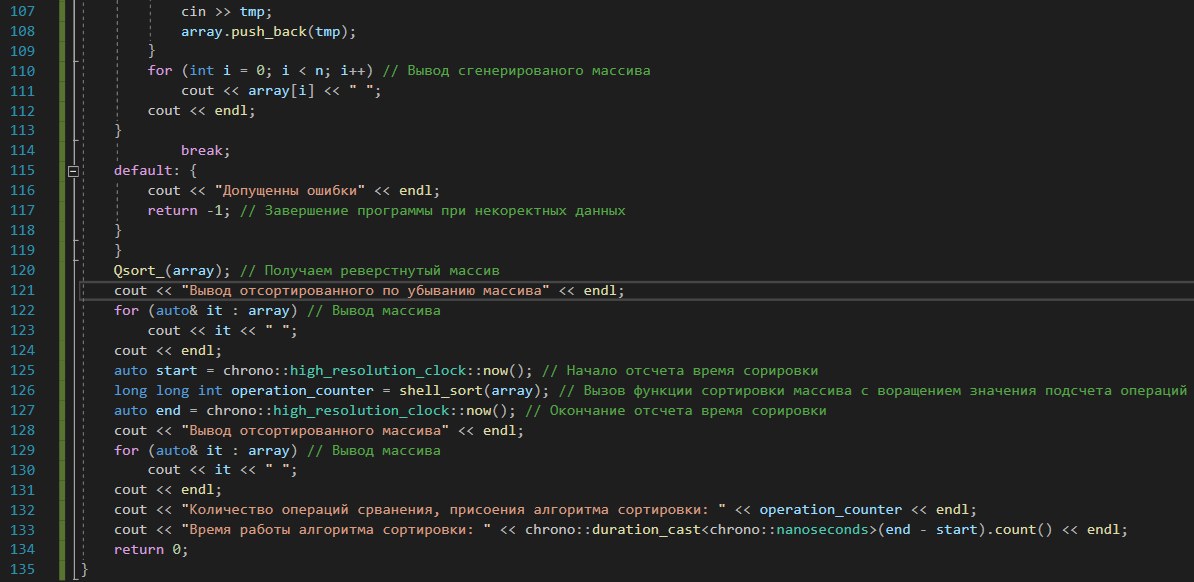


Рисунок 16 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

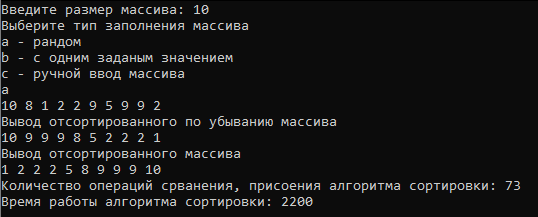


Рисунок 17 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

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 | 0.032 | 1304 |
| 1000 | 0.4919 | 20023 |
| 10000 | 6,5457 | 272194 |
| 100000 | 791.668737 | 3127319 |
| 1000000 | 2144083.13 | 35721310 |

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

### 

Рисунок 18 - График функции роста Тп алгоритма Шелла со смещениями Д. Кнута. Способ 2 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 алгоритма Шелла со смещениями Д. Кнута, Способ 2**

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.19). В функцию main добавим вызов функции сортировки по возрастанию(рис.20,21) и продемонстрируем работу программы при n=10 (рис.22). Алгоритм Шелла со смещениями Д. Кнута. Способ 2 сортировки не изменяется и соответствует продемонстрированному на рисунке 2.

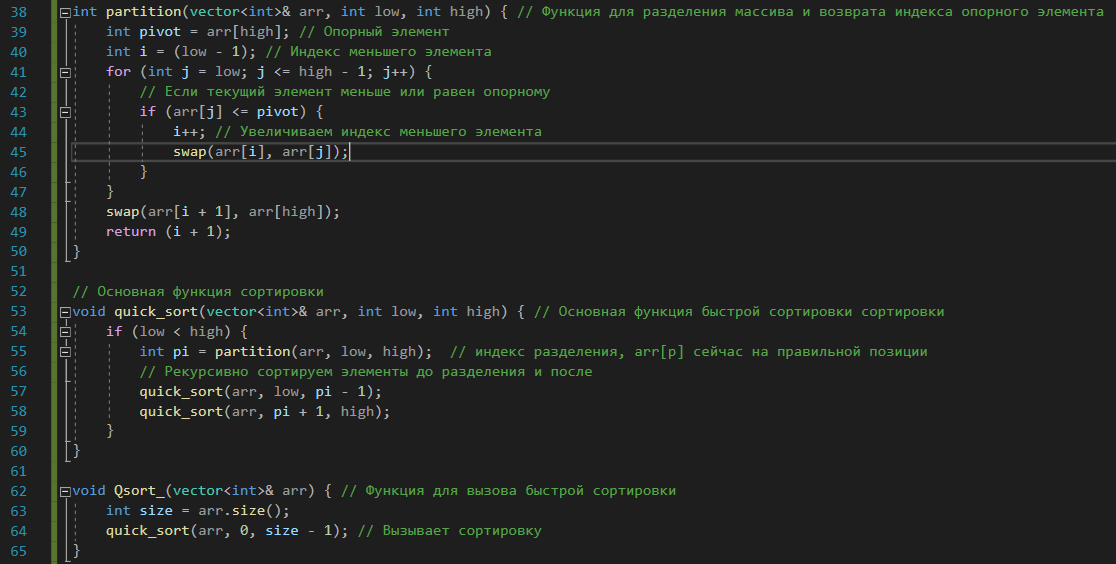


Рисунок 19 – Функция сортировки по возрастанию

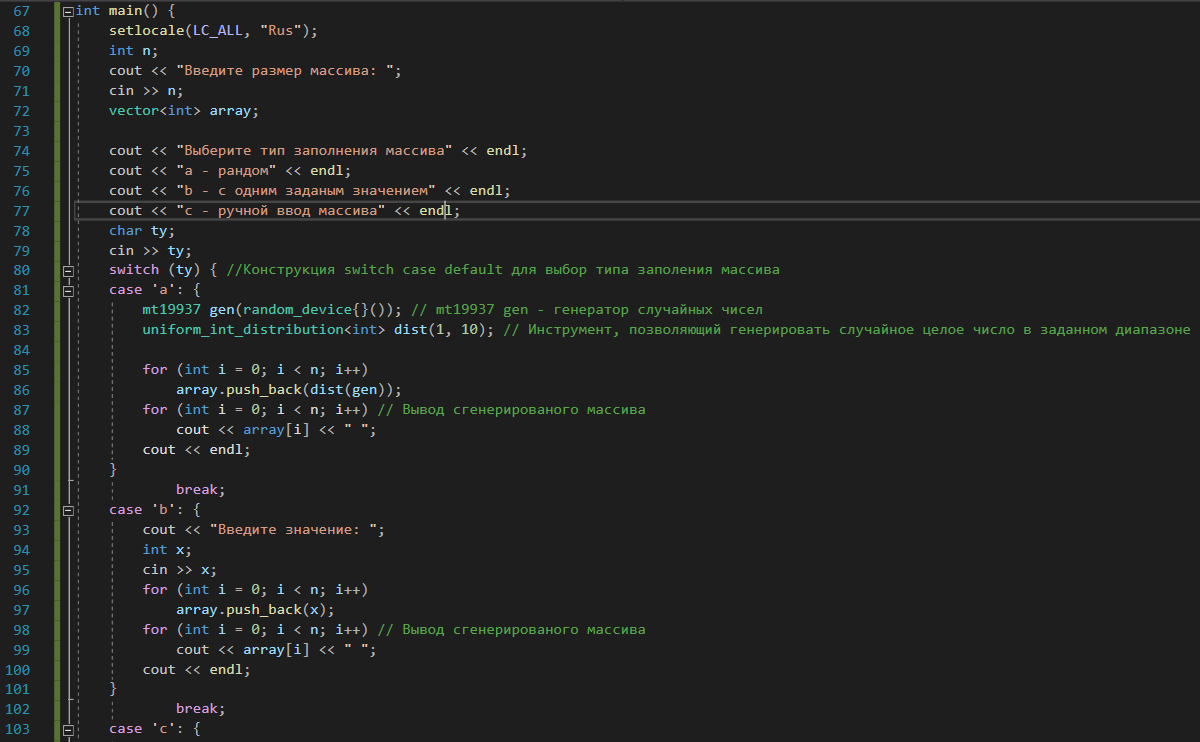


Рисунок 20 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

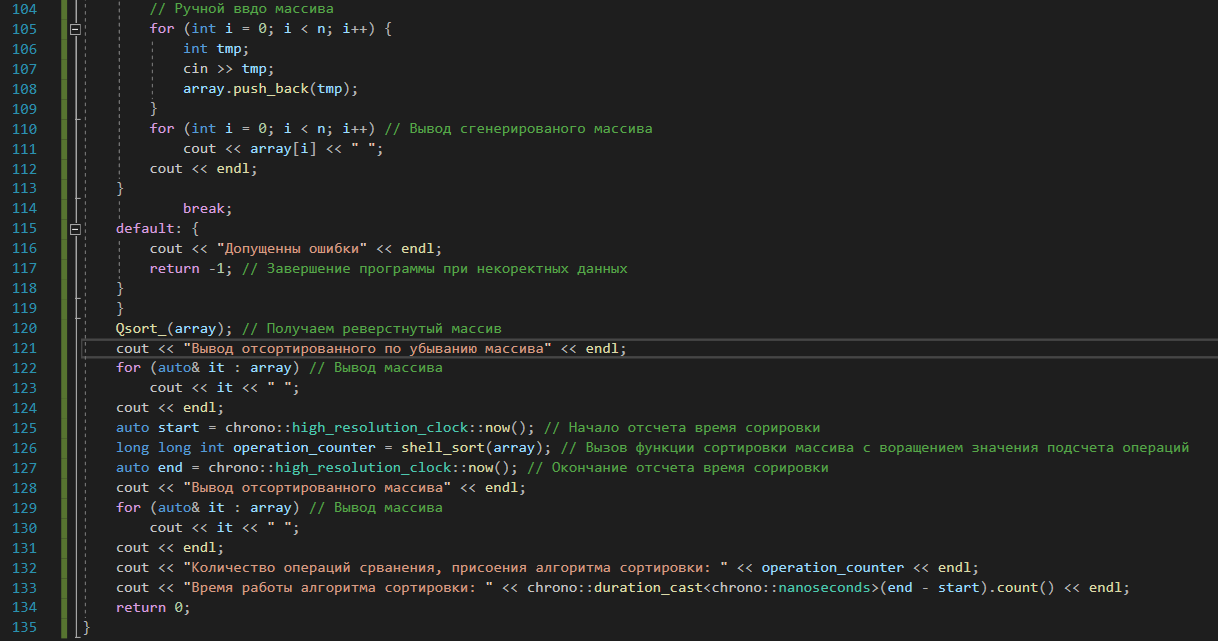


Рисунок 21 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

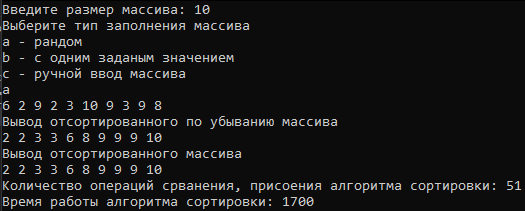


Рисунок 22 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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 | 0.000599 | 138 |
| 1000 | 0.003 | 1038 |
| 10000 | 0.023 | 10038 |
| 100000 | 0.29 | 100038 |
| 1000000 | 0.81 | 1000038 |

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.23).

### 

Рисунок 23 - График функции роста Тп алгоритм Шелла со смещениями Д. Кнута. Способ 2 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.24). В функцию main добавим вызов функции сортировки по убыванию (рис.25,26) и продемонстрируем работу программы при n=10 (рис.27). Алгоритм быстрой сортировки (Хоара) не изменяется и соответствует продемонстрированному на рисунке 7.

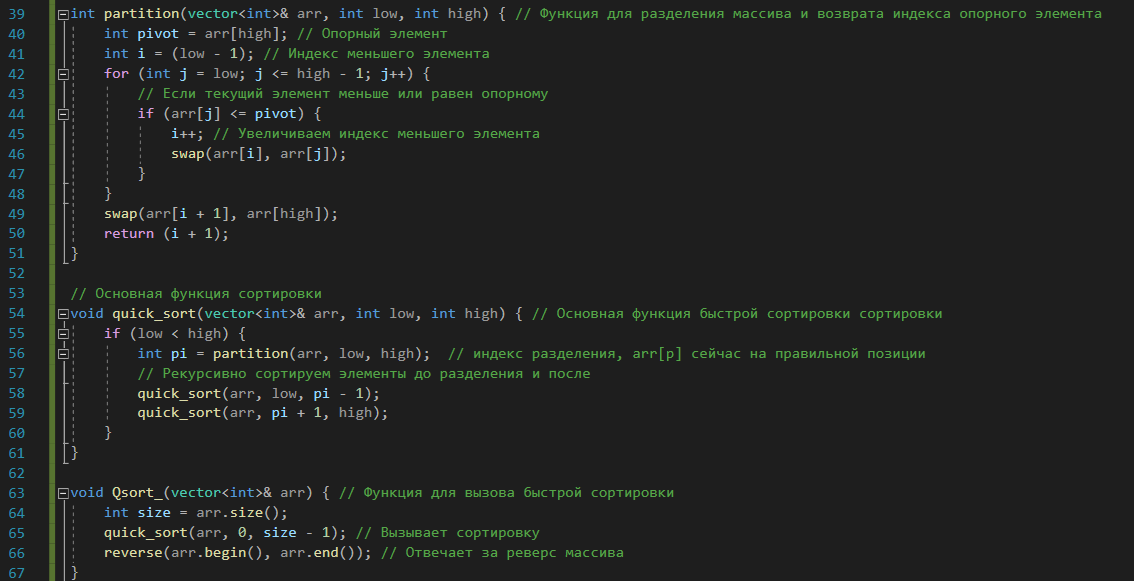


Рисунок 24 – Алгоритм сортировки по убыванию

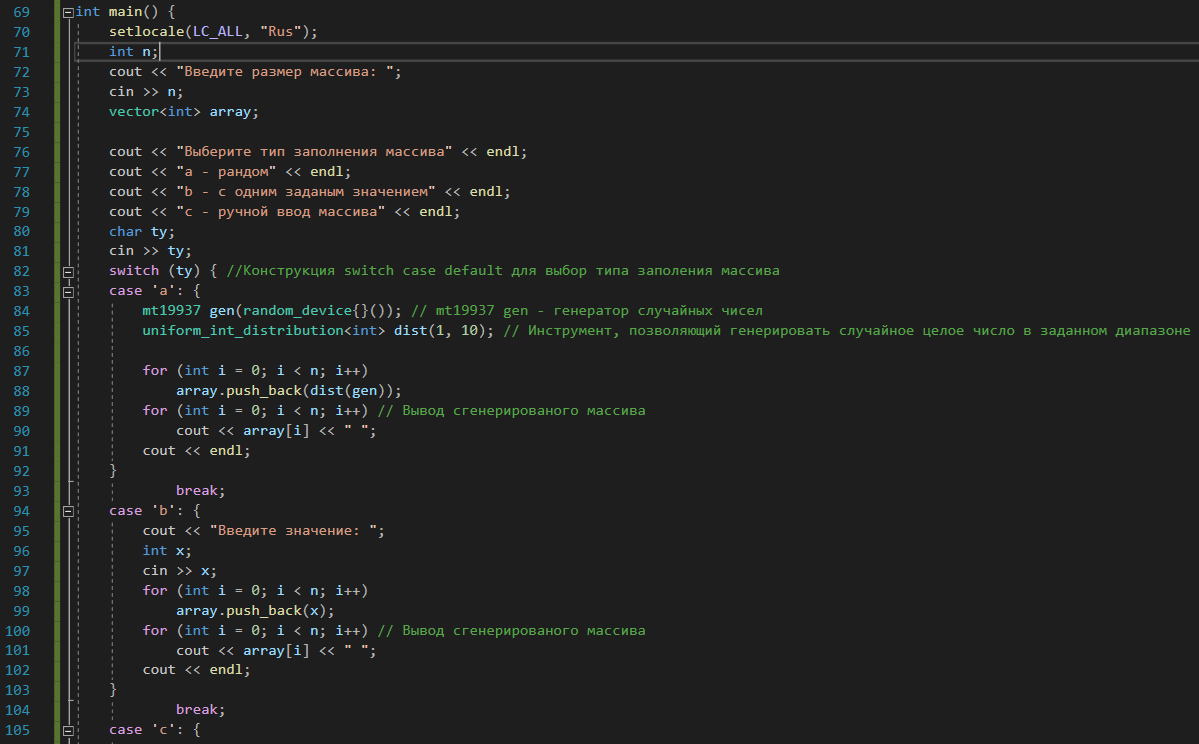


Рисунок 25 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

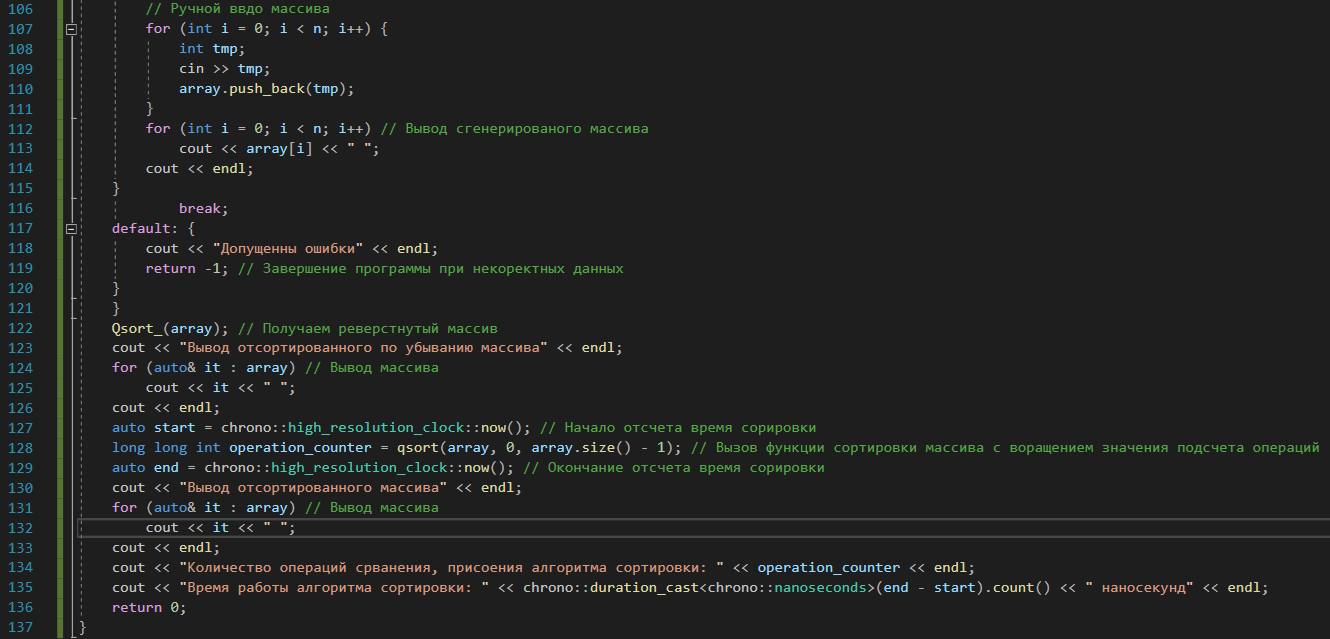


Рисунок 26 – Тестирование программы при n=10 и с отсортированными значениями по убыванию

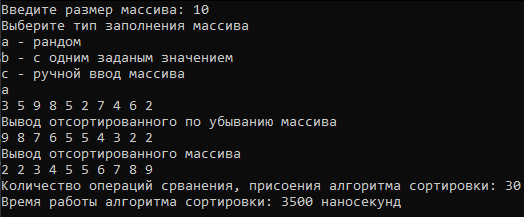


Рисунок 27 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

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 | 0.0048 | 255 |
| 1000 | 0.7312 | 2505 |
| 10000 | 10.1911 | 25005 |
| 100000 | 50.12 | 250005 |
| 1000000 | 132.3 | 2500005 |

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.28).

### 

Рисунок 28 - 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.29). В функцию main добавим вызов функции сортировки по возрастанию(рис.30,31) и продемонстрируем работу программы при n=10 (рис.32). Алгоритм быстрой сортировки (Хоара) не изменяется и соответствует продемонстрированному на рисунке 13.

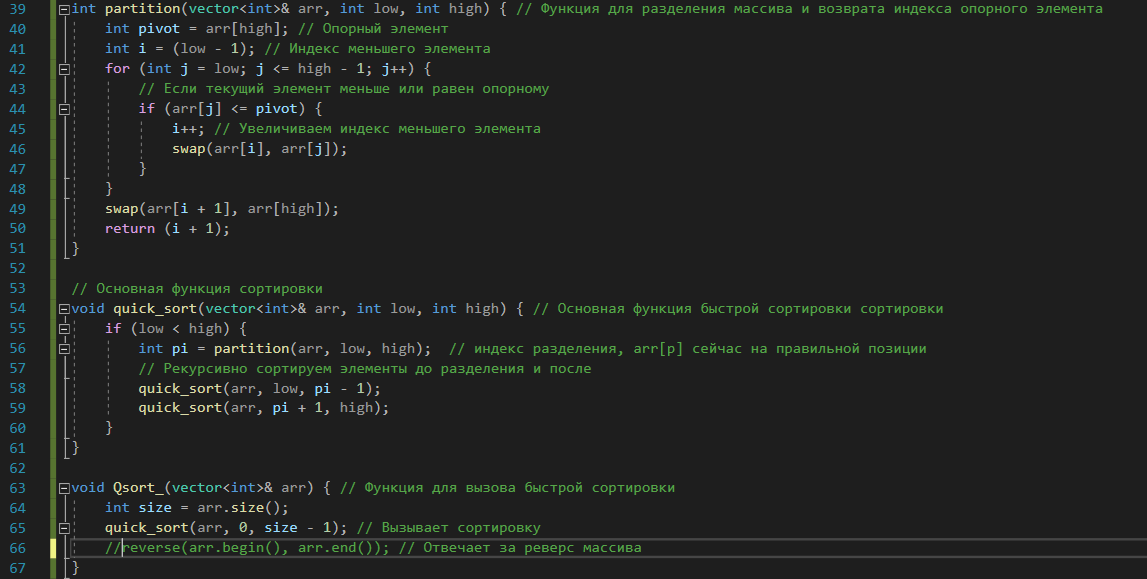


Рисунок 29 – Алгоритм сортировки по возрастанию

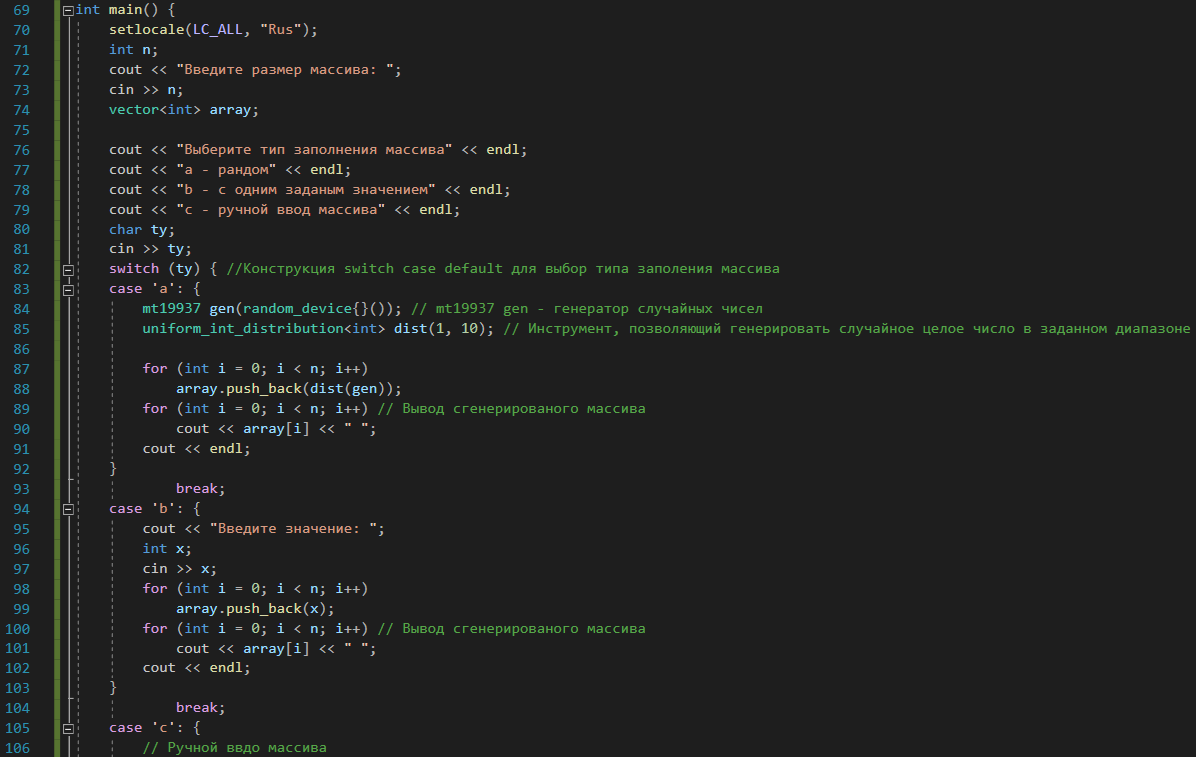


Рисунок 30 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

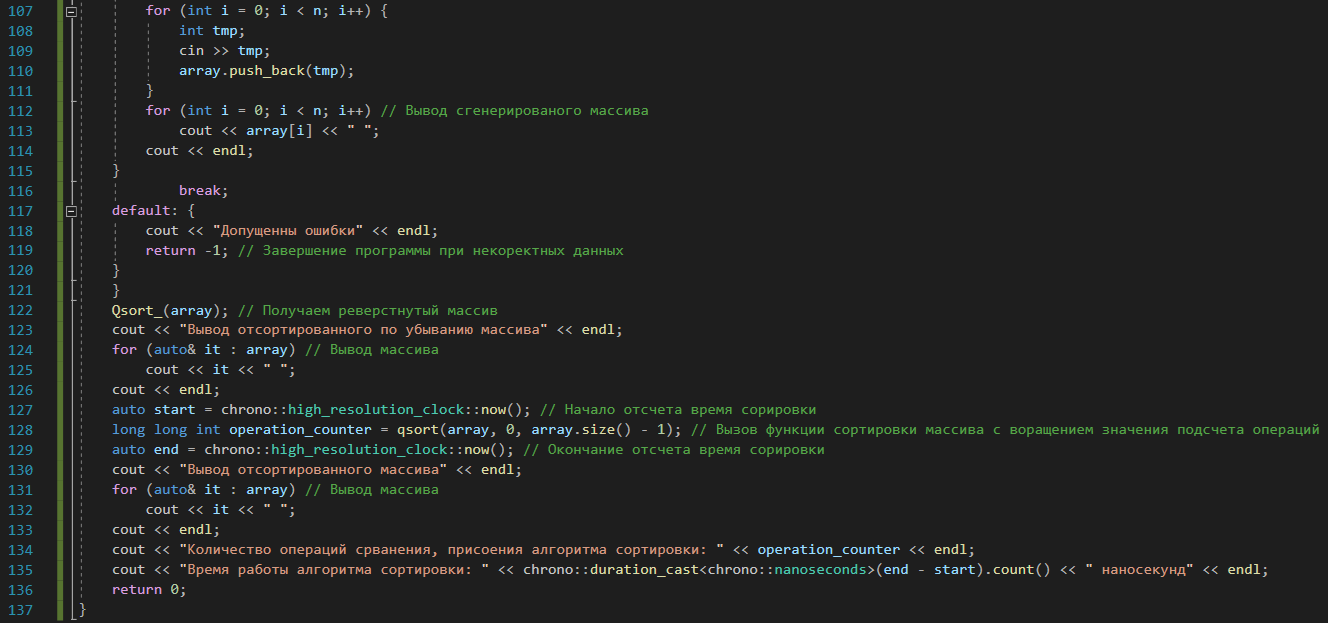


Рисунок 31 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

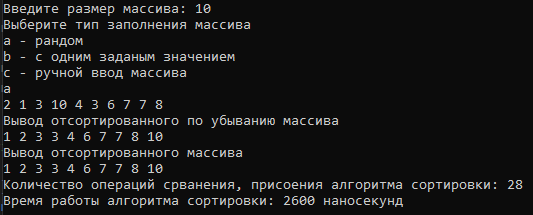


Рисунок 32 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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(nlog2n). 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 | 0.00418 | 212 |
| 1000 | 0.046862 | 2062 |
| 10000 | 0.879907 | 20520 |
| 100000 | 11.466231 | 205031 |
| 1000000 | 293.541 | 2050810 |

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.33).

### 

Рисунок 33 - 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, представленного в таблицах 1.1,1.2,1.4,1.5,1.6,1.7, можно сделать вывод о зависимости алгоритмов алгоритма Шелла со смещениями Д. Кнута. Способ 2 сортировки и быстрой сортировки (Хоара) от исходной упорядоченности массива. алгоритма Шелла со смещениями Д. Кнута. Способ 2 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, алгоритма Шелла со смещениями Д. Кнута. Способ 2 сортировка показывает стабильность и работает эффективно в любом случае. Быстрая сортировка (Хоара), в отличие от алгоритма Шелла со смещениями Д. Кнута. Способ 2 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 алгоритма Шелла со смещениями Д. Кнута. Способ 2 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 (Вариант 13)**

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 алгоритма Шелла со смещениями Д. Кнута. Способ 2 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.34).

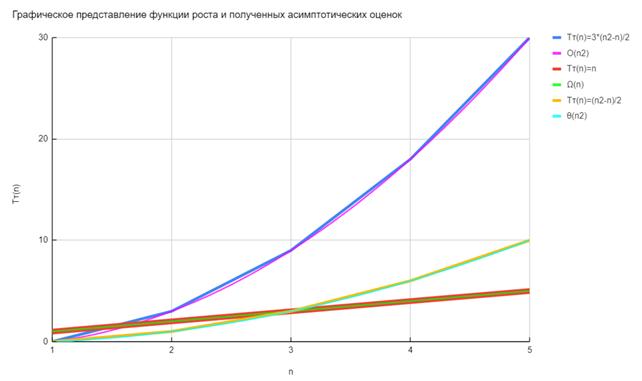


Рисунок 34 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритмов Шелла со смещениями Д. Кнута Способ 2сортировки и быстрой сортировки (Хоара)**

### **3.5.1 Справочная информация о вычислительной сложности алгоритма Шелла со смещениями Д. Кнута Способ 2 сортировки**

Асимптотическая оценка вычислительной сложности алгоритма Шелла со смещениями Д. Кнута. Способ 2 сортировки для худшего случая в О-нотации (оценка сверху) будет равна О(n2).

Асимптотическая оценка вычислительной сложности алгоритма Шелла со смещениями Д. Кнута. Способ 2сортировки для лучшего случая в Ω-нотации (оценка снизу) будет равна Ω(n).

Асимптотическая оценка вычислительной сложности алгоритма Шелла со смещениями Д. Кнута. Способ 2сортировки для среднего случая в θ-нотации будет равна θ(nlog2n).

Ёмкостная сложность алгоритма Шелла со смещениями Д. Кнута. Способ 2сортировки 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(nlog2n).

Асимптотическая оценка вычислительной быстрой сортировки (Хоара) для среднего случая в θ-нотации будет равна θ(nlog2n).

Ёмкостная сложность алгоритма Шелла со смещениями Д. Кнута. Способ 2сортировки 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 обмана, алгоритма Шелла со смещениями Д. Кнута. Способ 2 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) |
| алгоритма Шелла со смещениями Д. Кнута. Способ 2 сортировка | О(n2) | Ω(n logn) | θ(nlog2n) | О(1) |
| Быстрая сортировка (Хоара) | О(n2) | Ω(nlog2n) | θ(nlog2n) | О(log2n) |

**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(nlog2n), что является лучшим показателем среди представленных алгоритмов. В лучшем случае быстрая сортировка (Хоара) имеет асимптотическую сложность Ω(nlog2n), что также является очень хорошим показателем. В этом случае алгоритм быстрой сортировки работает оптимально и эффективно. Также быстрая сортировка имеет лучшую ёмкостную сложность O(log2n), 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 ВЫВОДЫ

В ходе практической работы были выполнены следующие этапы:

1. Освоение навыков анализа вычислительной сложности алгоритмов сортировки с целью определения наиболее эффективного метода.

2. Проведение анализа алгоритмов сортировки, таких как алгоритм Шелла с использованием смещений Д. Кнута, способ 2 сортировки и быстрая сортировка (Хоара).

3. Реализация программных решений для алгоритмов сортировки, включая алгоритм Шелла с использованием смещений Д. Кнута, способ 2 сортировки и быструю сортировку (Хоара).

4. Проведение тестирования программных решений для алгоритмов сортировки, чтобы проверить их корректность и эффективность.

5. Построение графиков функции роста времени выполнения алгоритмов сортировки, что позволило визуально сравнить их производительность.

6. Сравнение производительности алгоритмов простой сортировки обменом с алгоритмами Шелла с использованием смещений Д. Кнута, способом 2 сортировки и быстрой сортировкой (Хоара).

7. Проведение анализа асимптотической сложности алгоритмов сортировки, включая алгоритм простого выбора, алгоритм Шелла с использованием смещений Д. Кнута, способ 2 сортировки и быструю сортировку (Хоара).

8. Сравнение асимптотической сложности алгоритмов сортировки для определения наиболее эффективного метода.

Таким образом, основная цель практической работы, а именно приобретение навыков анализа вычислительной сложности алгоритмов сортировки и определение наиболее эффективного из них, была успешно достигнута.

# 5 ЛИТЕРАТУРА

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

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

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

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

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

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

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

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

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

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

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

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