|  |
| --- |
| 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: Синицин Д.Н. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_gjdgxs)

[2 ЗАДАНИЕ №1 5](#_30j0zll)

[2.1 Формулировка задачи (Вариант 9) 5](#_1fob9te)

[2.2 Описание выполнения, блок-схемы и случаи 6](#_2et92p0)

[2.2.1 Описание выполнения и блок-схема алгоритма сортировки обменами с условием Айверсона 6](#_tyjcwt)

[2.2.2 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки обменами с условием Айверсона 8](#_3dy6vkm)

[2.3 Реализация алгоритма на языке C++, проведение тестирования и построение графика 9](#)

[2.3.1 Реализация алгоритма сортировки обменами с условием Айверсона на языке C++ 9](#_1t3h5sf)

[2.3.2 Тестирование 10](#_4d34og8)

[2.4 Описание выполнения, блок-схема и случаи 11](#_2s8eyo1)

[2.4.1 Описание выполнения и блок-схема алгоритма быстрой сортировки (Хоара) 11](#_17dp8vu)

[2.4.2 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма быстрой сортировки (Хоара) 13](#_26in1rg)

[2.5 Реализация алгоритма на языке C++, проведение тестирования и построение графика 14](#_lnxbz9)

[2.5.1 Реализация алгоритма быстрой сортировки (Хоара) на языке C++ 14](#_35nkun2)

[2.5.2 Тестирование 15](#_1ksv4uv)

[2.6 Сортировка простым выбором 16](#_44sinio)

[2.7 Сравнение трёх алгоритмов на графике 16](#_2jxsxqh)

[2.8 Тестирование программ для алгоритмов сортировки обменами с условием Айверсона и быстрой сортировки (Хоара) 19](#_z337ya)

[2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма сортировки обменами с условием Айверсона 19](#_3j2qqm3)

[2.8.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма сортировки обменами с условием Айверсона 20](#)

[2.8.3 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма быстрой сортировки (Хоара) 22](#_1y810tw)

[2.8.4 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма быстрой сортировки (Хоара) 23](#)

[2.9 Вывод по заданию №1 25](#_4i7ojhp)

[3 ЗАДАНИЕ №2 26](#_1ci93xb)

[3.1 Формулировка задачи (Вариант 9) 26](#_3whwml4)

[3.2 Формулы функции роста алгоритма сортировки простым выбором в худшем и лучшем случае 26](#_2bn6wsx)

[3.3 Асимптотическая оценка вычислительной сложности простого алгоритма сортировки выбором 27](#)

[3.4 Графическое представление функции роста и полученных асимптотических оценок сверху и снизу 27](#_qsh70q)

[3.5 Справочная информация о вычислительной сложности алгоритмов сортировки обменами с условием Айверсона и быстрой сортировки (Хоара) 28](#_3as4poj)

[3.5.1 Справочная информация о вычислительной сложности алгоритма сортировки обменами с условием Айверсона 28](#_1pxezwc)

[3.5.2 Справочная информация о вычислительной сложности алгоритма быстрой сортировки (Хоара) 29](#_49x2ik5)

[3.6 Таблица асимптотической сложности трёх алгоритмов 29](#_2p2csry)

[3.7 Выводы по заданию №2 30](#_mlzyypah0ilv)

[4 КОНТРОЛЬНЫЕ ВОПРОСЫ 31](#_147n2zr)

[5 ВЫВОДЫ 37](#_3o7alnk)

[6 ЛИТЕРАТУРА 38](#_23ckvvd)

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи (Вариант 9)**

Эмпирическая оценка эффективности алгоритмов.

1. Разработать алгоритм алгоритма сортировки обменами с условием Айверсона, 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 сортировки обменами с условием Айверсона.

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 Описание выполнения, блок-схемы и случаи**

### **2.2.1 Описание выполнения и блок-схема алгоритма сортировки обменами с условием Айверсона**

Алгоритм сортировки обменами с условием Айверсона (или шейкерная сортировка) представляет собой улучшенную версию классической сортировки обменами (пузырьковой сортировки). Вместо того, чтобы проходить по массиву слева направо и перемещать максимальный элемент в конец массива на каждом проходе, алгоритм сначала проходит по массиву слева направо, а затем справа налево, обменивая соседние элементы, если они находятся в неправильном порядке. Этот процесс повторяется до тех пор, пока массив не будет отсортирован.

Основные шаги алгоритма сортировки обменами с условием Айверсона:

1. Установка начальных значений переменных swapped (логическая переменная, указывающая на наличие обменов на текущей итерации), start (индекс начала массива) и end (индекс конца массива).

2. Выполнение двух проходов по массиву: слева направо и справа налево.

3. На каждом проходе проверка пар соседних элементов. Если текущий элемент больше следующего, они обмениваются местами, и переменная swapped устанавливается в true.

4. После завершения прохода по всем элементам проверяется значение переменной swapped. Если на текущем проходе не было обменов, сортировка завершается, так как массив уже отсортирован.

5. Если были произведены обмены, обновляются значения переменных start и end, чтобы сузить диапазон элементов, по которым будет выполнен следующий проход.

6. Шаги 2-5 повторяются до тех пор, пока массив полностью не отсортирован.

Алгоритм сортировки обменами с условием Айверсона обладает некоторой эффективностью из-за того, что он прерывается, если на каком-то проходе не было обменов. Это позволяет избежать избыточных итераций, когда массив уже отсортирован.

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

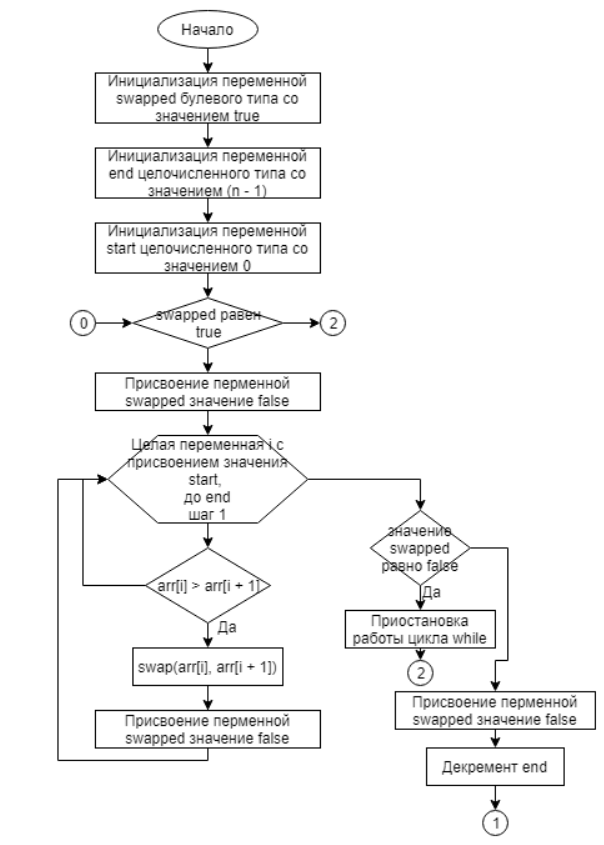


Рисунок 1 – Блок-схема алгоритма сортировки обменами с условием Айверсона

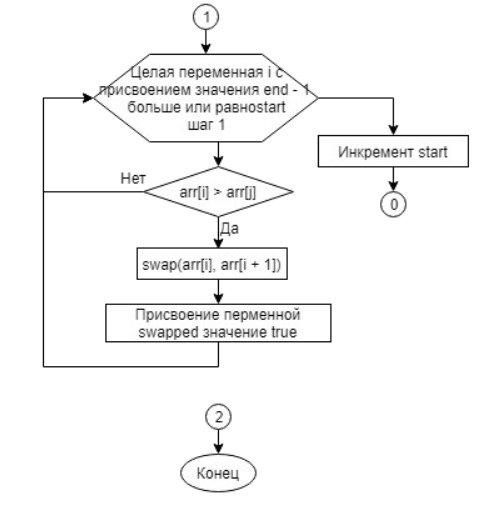


Рисунок 2 – Блок-схема алгоритма сортировки обменами с условием Айверсона

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

a. Лучший случай - массив уже отсортирован. В этом случае количество операций сравнения и перемещения будет минимальным и будет составлять O(n2).

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(n2).

Худший случай: O(n2).

Ёмкостная сложность алгоритма будет равна O(1).

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

### **2.3.1 Реализация алгоритма сортировки обменами с условием Айверсона на языке C++**

Реализуем данный алгоритм на языке C++(рис.3,4). Для реализации используем 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 позволит посчитать затраченное время. Для подсчёта количество операций введём переменную oper типа long.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Программа алгоритма сортировка обменами с условием Айверсона

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 – Функция main для алгоритма сортировка обменами с условием Айверсона

### **2.3.2 Тестирование**

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.

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Рисунок 5 - Тестирование программы при n=10

Таблица 1.1. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.04 | 12238 |
| 1000 | 5.144 | 1206616 |
| 10000 | 327.619 | 120427810 |
| 100000 | 48874.043 | 12036591944 |
| 1000000 | 4418647.28 | 1201495346447 |

## 

## **2.4 Описание выполнения, блок-схема и случаи**

### **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.6,7,8).

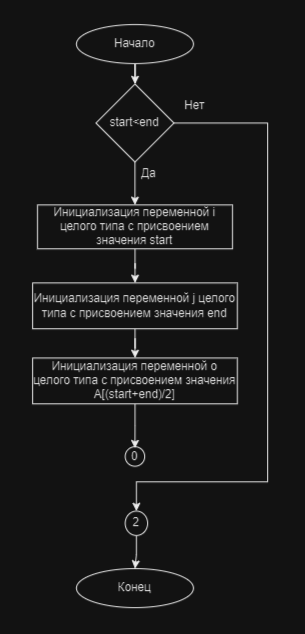


Рисунок 6 – Блок-схема алгоритма быстрой сортировки (Хоара)

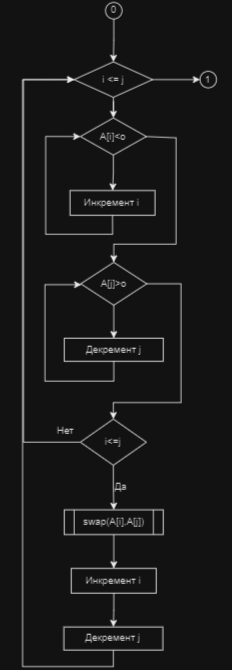


Рисунок 7 – Блок-схема алгоритма быстрой сортировки (Хоара)

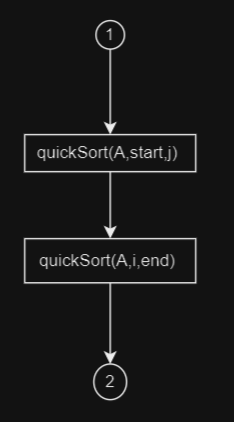


Рисунок 8 – Блок-схема алгоритма быстрой сортировки (Хоара)

### 

### **2.4.2 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(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).

b. 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.9,10). Для реализации используем 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. Для подсчёта количество операций присваивания или сравнения введём переменную oper типа long.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 9 – Программа алгоритма быстрой сортировки (Хоара)

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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

### **2.5.2 Тестирование**

Стоит задача протестировать программу с заданным размером массива n=10 (рис.11), 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.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 11 - Тестирование программы при n=10

Таблица 1.2. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.009 | 2057 |
| 1000 | 0.028 | 28018 |
| 10000 | 1.325 | 389007 |
| 100000 | 11.837 | 4454183 |
| 1000000 | 181.166 | 54840224 |

## **2.6 Сортировка простым выбором**

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

Таблица 1.3. Сводная таблица результатов

| **n** | **T(n), мс** | **Тп(n)=Cф+Mф** |
| --- | --- | --- |
| 100 | 0.036 | 6784 |
| 1000 | 1.535 | 673615 |
| 10000 | 201.064 | 68189125 |
| 100000 | 19696.345 | 6739321753 |
| 1000000 | 2160795.56 | 675445729131 |

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

На основе полученных данных, продемонстрированных в таблицах 1.1, 1.2 и 1.3, построим график функции роста Тп алгоритма сортировки обменами с условием Айверсона, 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).

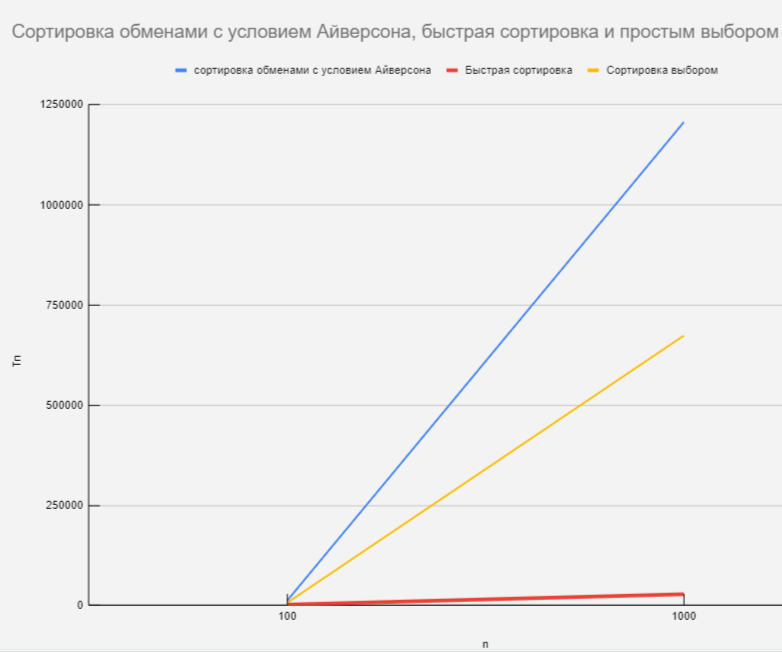


Рисунок 12 - График сравнения трёх сортировок в среднем случае при n до 1000

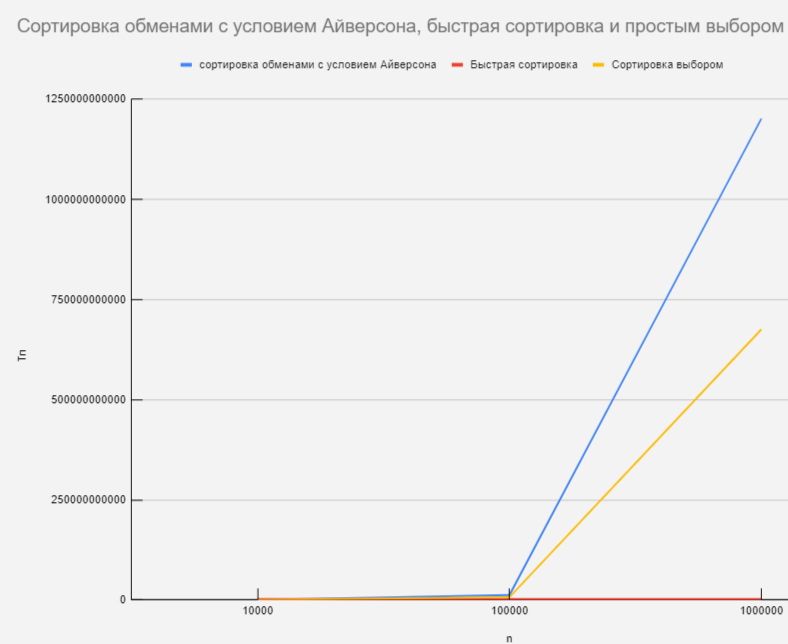


Рисунок 13 - График сравнения трёх сортировок в среднем случае при n от 10000 до 1000000

На основе таблиц 1.1, 1.2 и 1.3 и графиков(рис.12,13), можно сделать вывод, что в среднем случае алгоритм сортировка обменами с условием Айверсона сортировки самый неэффективный, алгоритм простого выбора второй по эффективности, а алгоритм быстрой сортировки (Хоара) самый эффективный.

## **2.8 Тестирование программ для алгоритмов сортировки обменами с условием Айверсона и быстрой сортировки (Хоара)**

### **2.8.1 Тестирование при упорядоченном по убыванию элементов массива и построение графика для алгоритма сортировки обменами с условием Айверсона**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго убывающем порядке. В функцию main добавим вызов функции сортировки по убыванию (рис.14) и продемонстрируем работу программы при n=10 (рис.15). Алгоритм сортировки обменами с условием Айверсона не изменяется и соответствует продемонстрированному на рисунке 3.

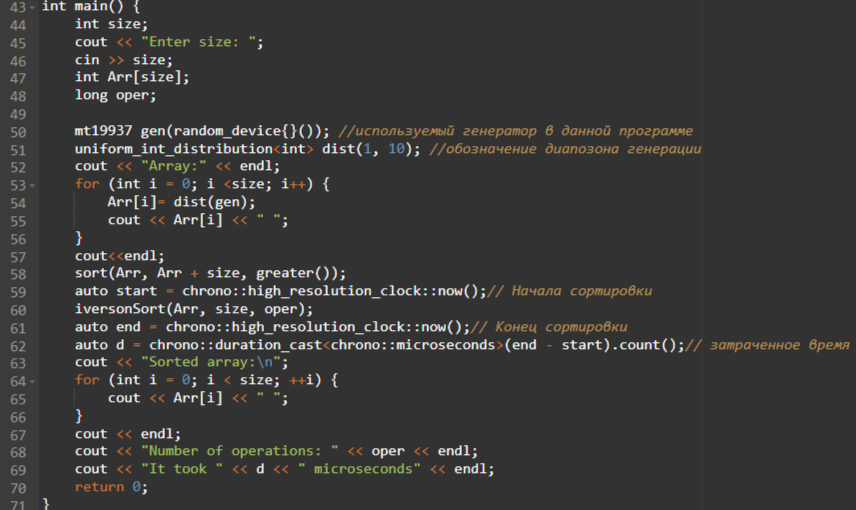


Рисунок 14 – Функция main при n=10 и с отсортированными значениями по убыванию

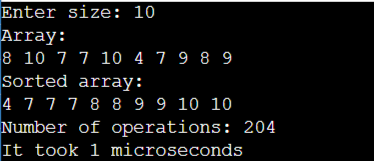


Рисунок 15 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

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 | 0.063 | 19214 |
| 1000 | 5.115 | 1901540 |
| 10000 | 463.822 | 190009774 |
| 100000 | 47190.543 | 19000176798 |
| 1000000 | 4374573.742 | 190000563452 |

### **2.8.2 Тестирование при упорядоченном по возрастанию элементов массива и построение графика для алгоритма сортировки обменами с условием Айверсона**

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго возрастающем порядке. В функцию main добавим вызов функции сортировки по возрастанию(рис.16) и продемонстрируем работу программы при n=10 (рис.17). Алгоритм сортировки обменами с условием Айверсона не изменяется и соответствует продемонстрированному на рисунке 3.

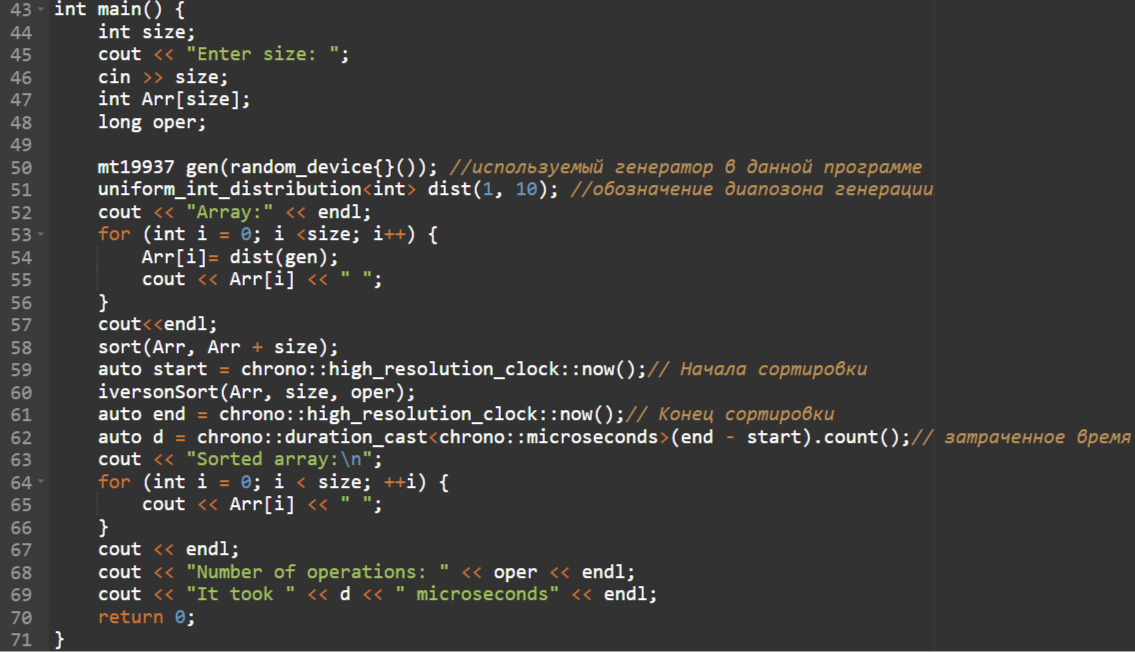


Рисунок 16 – Функция main при n=10 и с отсортированными значениями по возрастанию

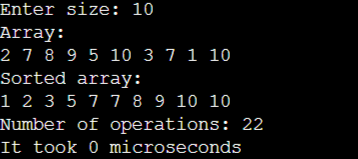


Рисунок 17 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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 | 0.001 | 202 |
| 1000 | 0.003 | 2002 |
| 10000 | 0.028 | 20002 |
| 100000 | 0.048 | 200002 |
| 1000000 | 0.283 | 2000002 |

### **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. В функцию main добавим вызов функции сортировки по убыванию (рис.18) и продемонстрируем работу программы при n=10 (рис.19). Алгоритм быстрой сортировки (Хоара) не изменяется и соответствует продемонстрированному на рисунке 9.

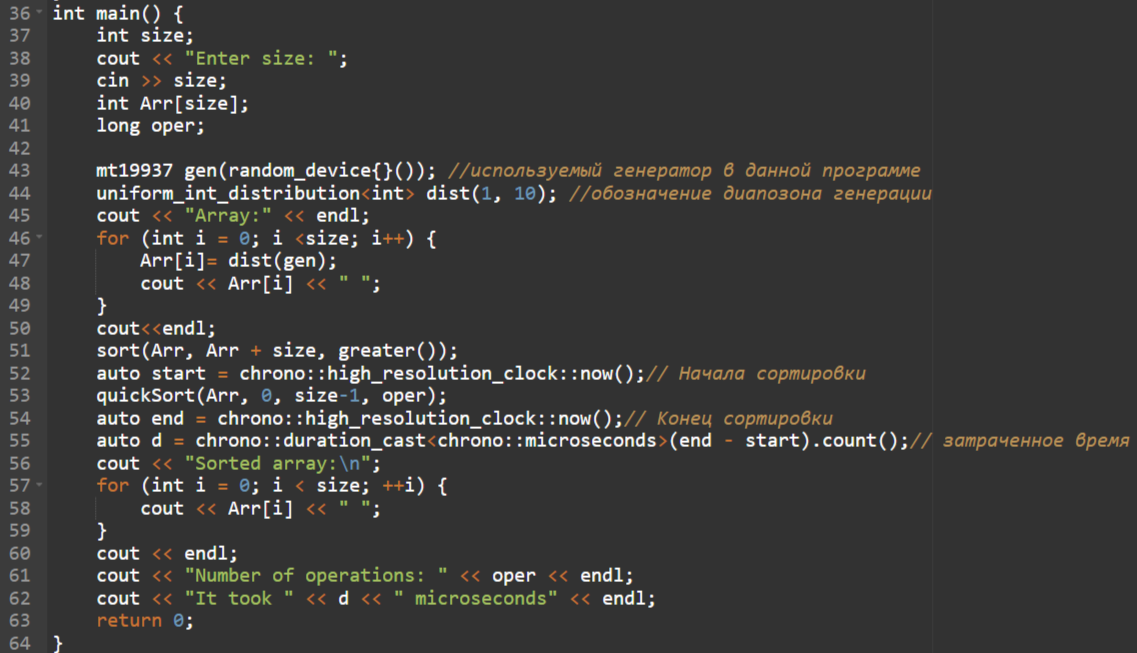


Рисунок 18 –Функция main при n=10 и с отсортированными значениями по убыванию

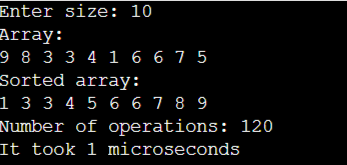


Рисунок 19 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

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 | 0.007 | 1999 |
| 1000 | 0.078 | 27362 |
| 10000 | 0.777 | 354829 |
| 100000 | 13.498 | 4475096 |
| 1000000 | 112.369 | 52665868 |

### **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. В функцию main добавим вызов функции сортировки по возрастанию(рис.20) и продемонстрируем работу программы при n=10 (рис.21). Алгоритм быстрой сортировки (Хоара) не изменяется и соответствует продемонстрированному на рисунке 9.

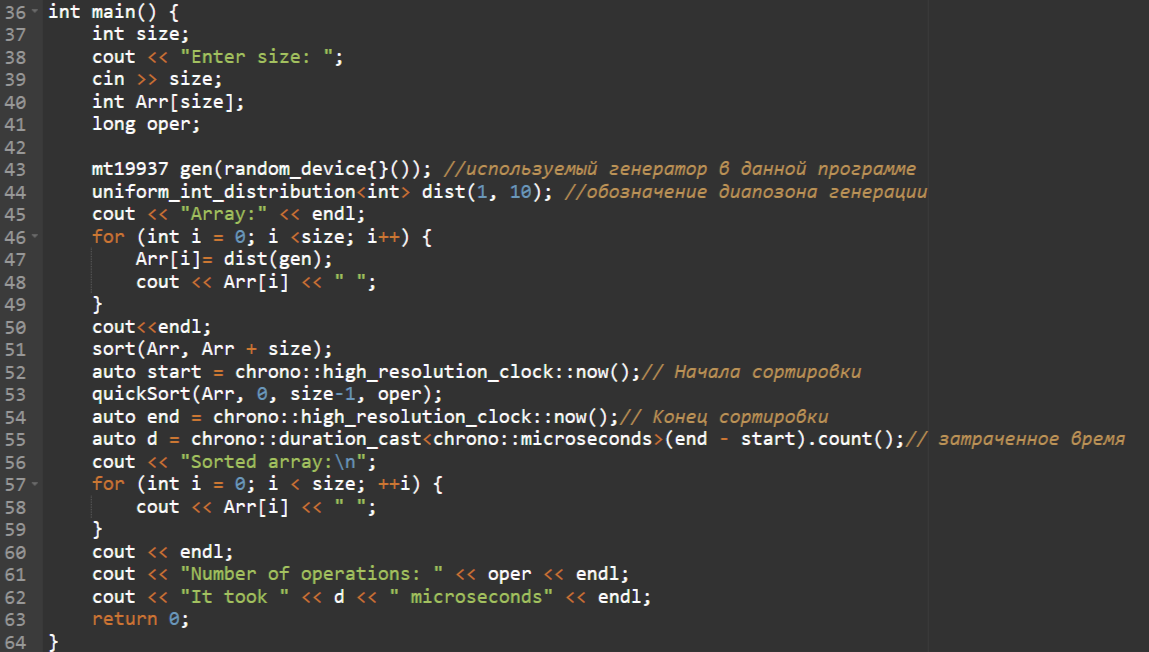


Рисунок 20 – Функция main при n=10 и с отсортированными значениями по возрастанию

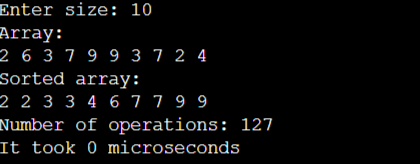


Рисунок 21 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

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 | 0.007 | 1996 |
| 1000 | 0.068 | 26445 |
| 10000 | 0.723 | 345335 |
| 100000 | 8.907 | 4432197 |
| 1000000 | 485.133 | 52260518 |

## **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 сортировки обменами с условием Айверсона и быстрой сортировки (Хоара) от исходной упорядоченности массива. На практике алгоритмы сортировки обменами с условием Айверсона и быстрая сортировки (Хоара) обе показывают эффективные результаты независимо от исходной упорядоченности массива. Однако, быстрая сортировка обычно более эффективна и быстрая на больших объемах данных, чем алгоритм сортировки обменами.

И алгоритм сортировки обменами и быстрая сортировка работают быстро на случайных или отсортированных по возрастанию массивах. Алгоритм сортировки обменами с условием Айверсона может быть немного менее эффективным на обратно отсортированных данных, в то время как быстрая сортировка сохраняет свою эффективность.

Таким образом, как правило, исходная упорядоченность массива не сильно влияет на скорость работы быстрой сортировки, в то время как алгоритм сортировки обменами может показывать небольшое снижение эффективности на уже отсортированных данных.

# 

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи (Вариант 9)**

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 выбором в лучшем и худшем случае.

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

- 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 сортировки обменами с условием Айверсона 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 выбором 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 -2.

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

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

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

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

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

Ёмкостная сложность алгоритма простой сортировки выбором 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.22).

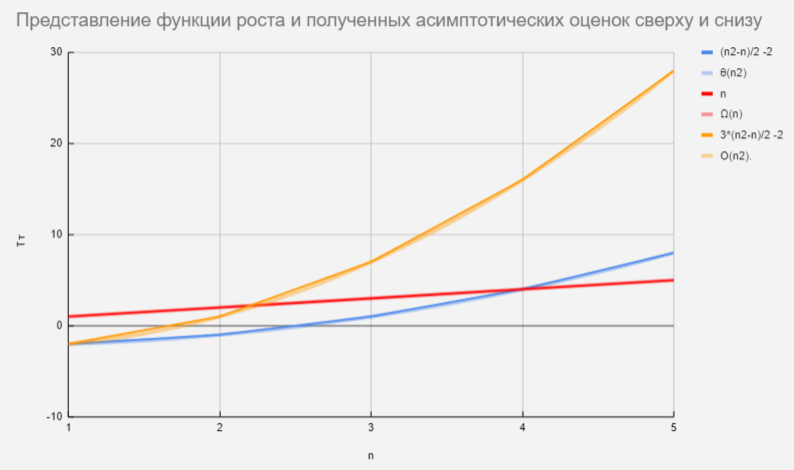


Рисунок 22 - Графическое представление функции роста и полученных асимптотических оценок сверху и снизу

## **3.5 Справочная информация о вычислительной сложности алгоритмов сортировки обменами с условием Айверсона и быстрой сортировки (Хоара)**

### **3.5.1 Справочная информация о вычислительной сложности алгоритма сортировки обменами с условием Айверсона**

Асимптотическая оценка вычислительной сложности алгоритма сортировка обменами с условием Айверсона для худшего случая в О-нотации (оценка сверху) будет равна О(n2).

Асимптотическая оценка вычислительной сложности алгоритма сортировка обменами с условием Айверсона для лучшего случая в Ω-нотации (оценка снизу) будет равна Ω(n).

Асимптотическая оценка вычислительной сложности алгоритма сортировка обменами с условием Айверсона для среднего случая в θ-нотации будет равна θ(n2).

Ёмкостная сложность алгоритма сортировка обменами с условием Айверсона 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 быстрой сортировки (Хоара) 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 выбора, сортировки обменами с условием Айверсона 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 выбор | О(n2) | Ω(n) | θ(n2) | О(1) |
| Сортировка обменами с условием Айверсона | О(n2) | Ω(n) | θ(n2) | О(1) |
| Быстрая сортировка (Хоара) | О(n2) | Ω(nlog2n) | θ(nlog2n) | О(log2n) |

## **3.7 Выводы по заданию №2**

Анализ таблицы 2 показывает, что наиболее эффективным алгоритмом сортировки является быстрая сортировка (Хоара). В среднем случае её асимптотическая сложность составляет θ(nlog2n), что является наилучшим результатом среди представленных алгоритмов. В лучшем же случае быстрая сортировка (Хоара) имеет асимптотическую сложность Ω(nlog2n), что также очень хорошо. В таких ситуациях алгоритм быстрой сортировки работает оптимально и эффективно. Кроме того, быстрая сортировка обладает лучшей ёмкостной сложностью O(log2n), что также является её преимуществом. Однако стоит отметить, что в худшем случае алгоритм быстрой сортировки имеет сложность O(n2), что может оказаться 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.

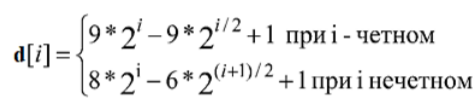
6. От чего зависит эффективность сортировки Шелла в среднем случае?

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(nlog2n).

Асимптотическая оценка вычислительной сложности алгоритма простого слияния для лучшего случая в Ω-нотации (оценка снизу) будет равна Ω(nlog2n).

Асимптотическая оценка вычислительной простого слияния для среднего случая в θ-нотации будет равна θ(nlog2n).

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(nlog2n). 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 сортировки обменами с условием Айверсона и быстрой сортировки (Хоара);

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

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

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

- Произведено сравнение алгоритмов сортировки обменами с условием Айверсона и быстрой сортировки (Хоара);

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

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

- Проведено определение наиболее эффективного алгоритма.

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

# 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).