|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 2** | |
| --- | --- |
| **Тема:** | |
| **«Эмпирический анализ сложности простых алгоритмов сортировки»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Кагарманов В.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 Описание выполнения алгоритма, доказательство корректности циклов и определение ситуаций лучшего, худшего и среднего случая. 6](#_3znysh7)

[2.2.1 Описание выполнения алгоритма 6](#_et731lhdmx49)

[2.2.2 Доказательство корректности циклов 7](#_tyjcwt)

[2.2.3 Ситуации лучшего, среднего и худшего случаев 8](#_3dy6vkm)

[2.3 Разработка программы алгоритма простым обменом 8](#)

[2.4 Тестирование 10](#_2s8eyo1)

[2.5 Построение графика 11](#_17dp8vu)

[2.6 Вывод 11](#_3rdcrjn)

[3 ЗАДАНИЕ №2 13](#_26in1rg)

[3.1 Формулировка задачи 13](#_lnxbz9)

[3.2 Тестирование программы при массиве упорядоченному по убыванию 13](#_35nkun2)

[3.3 Построение графика 15](#_2211lb4to67v)

[3.4 Тестирование программы при массиве упорядоченному по возрастанию 15](#_5u2yfrg0c0b)

[3.5 Построение графика 17](#_8mtkaqmhtdrb)

[3.6 Вывод 17](#_44sinio)

[4 ЗАДАНИЕ №3 18](#_z337ya)

[4.1 Формулировка задачи 18](#_3j2qqm3)

[4.2 Описание выполнения алгоритма, доказательство корректности циклов и определение ситуаций лучшего, худшего и среднего случая. 19](#_8i4amm7dzglx)

[4.2.1 Описание выполнения алгоритма 19](#_cembsd8kbcib)

[4.2.2 Доказательство корректности циклов 20](#_2xcytpi)

[4.2.3 Ситуации лучшего, среднего и худшего случаев 21](#_1ci93xb)

[4.3 Разработка программы алгоритма простой вставки 22](#)

[4.4 Тестирование 23](#_id69akuyqhoq)

[4.5 Построение графика 24](#_7xyrwnobilws)

[4.6 Тестирование программы при массиве упорядоченному по убыванию 25](#_7q3bnm1r512y)

[4.7 Построение графика 27](#_xoz5x0aw0wvn)

[4.8 Тестирование программы при массиве упорядоченному по возрастанию 28](#_1u4hjlondnj4)

[4.9 Построение графика 29](#_opzskzrjan7q)

[4.10 Сравнение графиков двух алгоритмов 30](#_3o7alnk)

[4.5 Вывод 31](#_23ckvvd)

[4 КОНТРОЛЬНЫЕ ВОПРОСЫ 33](#_ihv636)

[5 ВЫВОДЫ 34](#_32hioqz)

[6 ЛИТЕРАТУРА 35](#_1hmsyys)

# 1 ЦЕЛЬ

Актуализация знаний и приобретение практических умений по эмпирическому определению вычислительной сложности алгоритмов.

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи**

Вариант 2, в списке 14. Алгоритм 1 и 2 задания: простой обмен («пузырек», Exchange sort).

Оценить эмпирически вычислительную сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1. Составить функцию простой сортировки одномерного целочисленного массива A[n], используя алгоритм согласно варианту индивидуального задания. Провести тестирование программы на исходном массиве n=10.

2. Используя теоретический подход, определить для алгоритма:

a. Что будет ситуациями лучшего, среднего и худшего случаев.

b. Функции роста времени работы алгоритма от объёма входа для лучшего и худшего случаев.

3. Провести контрольные прогоны программы массивов случайных чисел при n = 100, 1000, 10000, 100000 и 1000000 элементов с вычислением времени выполнения T(n) – (в миллисекундах/секундах). Полученные результаты свести в сводную таблицу 2.

4. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операций Тп как сумму сравнений Сп и перемещений Мп. Полученные результаты вставить в сводную таблицу 2.

5. Построить график функции роста Тп этого алгоритма от размера массива n.

6. Определить ёмкостную сложность алгоритма.

7. Сделать вывод об эмпирической вычислительной сложности алгоритма на основе скорости роста функции роста.

## **2.2 Описание выполнения алгоритма, доказательство корректности циклов и определение ситуаций лучшего, худшего и среднего случая.**

### **2.2.1 Описание выполнения алгоритма**

Алгоритм простой сортировки обменом (или пузырьковая сортировка) - это простой алгоритм сортировки, который проходит по списку элементов многократно и меняет местами соседние элементы, если они находятся в неправильном порядке.Алгоритм проходит по всему массиву начиная с начала. Во время прохода, он сравнивает два соседних элемента. Если они стоят в неправильном порядке (меньший элемент стоит после большего), они меняются местами. Этот процесс повторяется для каждой пары соседних элементов в массиве. При каждом проходе самый большой элемент "всплывает" на правильное место в массиве. После каждого прохода самый большой элемент массива уже стоит на правильном месте, поэтому его можно исключить из следующих итераций.Таким образом, диапазон, по которому проходит сортировка, сужается на каждой итерации.

Алгоритм продолжает сравнивать и менять элементы до тех пор, пока все элементы не будут находиться в правильном порядке (т.е. массив будет отсортирован).

Преимущества пузырьковой сортировки включают ее простоту реализации и легкость понимания. Однако эта сортировка неэффективна на больших объемах данных из-за большого количества операций перестановок.

Представим данный алгоритм в виде блок-схемы(рис.1).

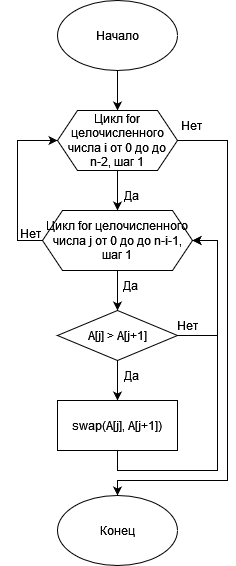


Рисунок 1 – Блок-схема алгоритма сортировки простыми вставками

### **2.2.2 Доказательство корректности циклов**

Найдём инварианты для внешнего и внутреннего цикла. Инвариантом внешнего цикла будет являться значение переменной i всегда меньше n-1. Инвариантом для внутреннего цикла будет являться значение переменной i всегда меньше n-i-1.

Докажем корректность циклов для чего нужно доказать их конечность. Внешний цикл проходит через все числа, начиная с первого и заканчивая предпоследним, а второй цикл идет на половину чисел. Значения проверяются попарно, и если первое число больше второго, то они меняются местами. После первого прохода самое большое число становится последним. Следовательно, циклы данного алгоритма конечны, а следовательно циклы алгоритма корректны.

### **2.2.3 Ситуации лучшего, среднего и худшего случаев**

Таблица 1-Псевдокод и анализ алгоритма сортировки обменом

| № | Алгоритм, записанный на псевдокоде | Количество выполнений оператора |
| --- | --- | --- |
| 1 | function ExchangeSort(a): |  |
| 2 | for i ← 0 to (n - 2) do | n |
| 3 | for j ← 1 to (n - 1 - i) do | tj = 𝑛 + 𝑛 − 1 + ⋯ + 2=0.5𝑛2 − 1.5𝑛 + 1 |
| 4 | if (a[j] > a[j + 1]) then | tj - 1= 0.5𝑛2 − 2.5𝑛 + 2 |
| 5 | swap(a[j], a[j + 1]) | 3\*(tj - 1) = 1.5𝑛2 − 7.5𝑛 + 6 |
| 6 | endif |  |
| 7 | оd |  |
| 8 | od |  |
| 9 | } |  |

В наилучшем случае, когда массив уже упорядочен по возрастанию, количество действий составит O(n). В среднем случае, когда массив содержит случайные числа, алгоритм будет иметь сложность O(n2). В худшем случае, когда массив упорядочен по убыванию, количество действий будет O(n2). Относительно времени выполнения: лучший случай - O(n), худший случай - O(n2). Ёмкостная сложность алгоритма оценивается как O(1).

## 

## **2.3 Разработка программы алгоритма простым обменом**

Разработаем программу алгоритма простым обменом на языке C++(рис.2,3). В данной программе используются следующие библиотеки и концепции:

<iostream>: для стандартного ввода-вывода данных.

<random>: для использования генератора случайных чисел и инструмента равномерного распределения.

<chrono>: для измерения времени выполнения сортировки.

using namespace std;: для удобства использования стандартного пространства имен.

Функция swap: для обмена значений двух элементов массива.

Функция ExchangeSort: основная функция сортировки обменом (метод пузырька), которая принимает массив, его размер и счетчик операций.

Использование генератора случайных чисел mt19937 и распределения uniform\_int\_distribution для заполнения массива случайными значениями.

Измерение времени начала и окончания сортировки с помощью <chrono>.

Подсчет времени выполнения сортировки в микросекундах.

Вывод отсортированного массива, количества операций и времени сортировки на экран.

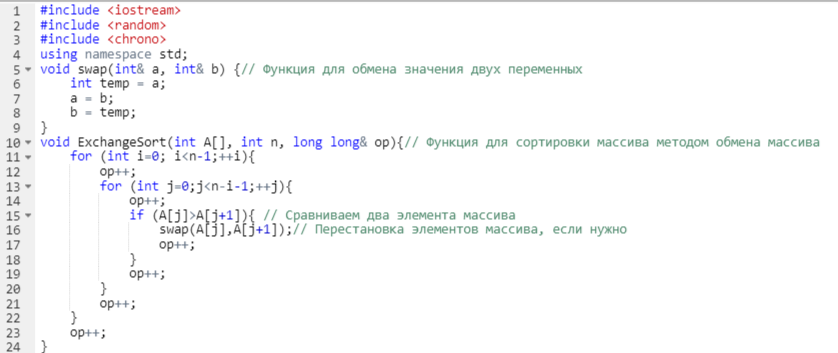


Рисунок 2 – Программа алгоритма сортировки простым обменом

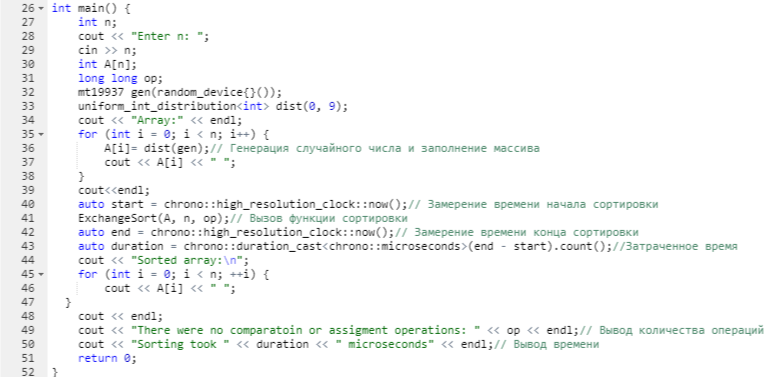


Рисунок 3 – Функция main для алгоритма сортировки простым обменом

## **2.4 Тестирование**

Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 2. При n=10 результат будет предоставлен на рисунке 4.

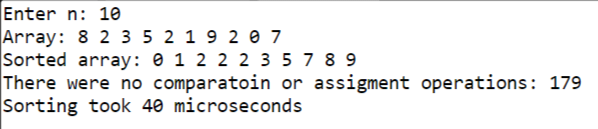


Рисунок 4 - Тестирование программы

Таблица 2. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.07 | - | 12351 |
| 1000 | 2.819 | - | 1222976 |
| 10000 | 444.271 | - | 122480570 |
| 100000 | 48554.832 | - | 12235451560 |
| 1000000 | 6331550.09 | - | 122249302750 |

### 

## **2.5 Построение графика**

Построим график функции роста Тп алгоритма простым обменом от размера массива на основе данных в таблице 2(рис.5).

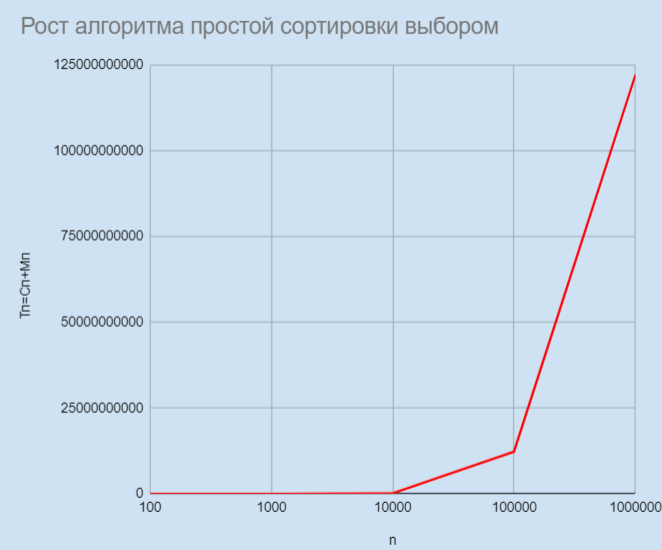


Рисунок 5 - График функции роста Тп алгоритма от размера массива n

## **2.6 Вывод**

Алгоритм простого выбора - это метод сортировки, который на каждом шаге находит наименьший элемент в оставшейся части массива и помещает его в начало. Этот процесс продолжается до тех пор, пока весь массив не будет отсортирован.

Анализ алгоритма простого выбора показывает, что его сложность в среднем составляет O(n2). Этот алгоритм неэффективен для больших массивов из-за квадратичной сложности, но может быть хорошим выбором для небольших массивов или в случаях, когда дополнительная память ограничена.

# 

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи**

Вариант 2, в списке 14. Алгоритм 1 и 2 задания: простой обмен («пузырек», Exchange sort).

Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

1. Провести дополнительные прогоны программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных:

a. строго в убывающем порядке значений, результаты представить в сводной таблице по формату Таблицы 2;

b. строго в возрастающем порядке значений, результаты представить в сводной таблице по формату Таблицы 2;

2. Сделать вывод о зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

## **3.2 Тестирование программы при массиве упорядоченному по убыванию**

При упорядоченном массиве по убыванию алгоритм обладает O(n2), так как это худший случай. Допишем в уже имеющуюся программу функцию сортировки по убыванию(рис.6). Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 3. При n=10 результат будет предоставлен на рисунке 7.

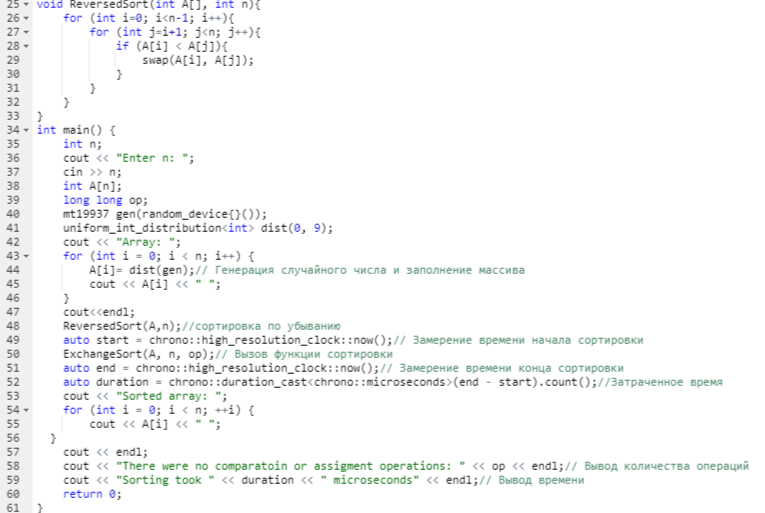


Рисунок 6 – Функция сортировки по убыванию и функция main

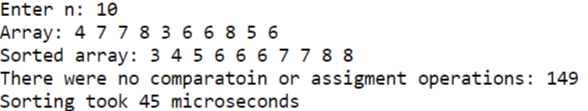


Рисунок 7 – Тестирование программы

Таблица 3. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.082 |  | 14563 |
| 1000 | 4.991 |  | 1450652 |
| 10000 | 394.781 |  | 145005478 |
| 100000 | 40754.652 |  | 14500029992 |
| 1000000 | 5061727.78 |  | 1450000299992 |

## **3.3 Построение графика**

Построим график функции роста Тп алгоритма простым обменом в худшем случае от размера массива на основе данных в таблице 3(рис.8).

### 

Рисунок 8 - График функции роста Тп алгоритма в худшем случае

## **3.4 Тестирование программы при массиве упорядоченному по возрастанию**

При упорядоченном массиве по возрастанию алгоритм обладает O(n), так как это лучший случай. Допишем в уже имеющуюся программу функцию сортировки по возрастанию(рис.9). Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 4. При n=10 результат будет предоставлен на рисунке 10.

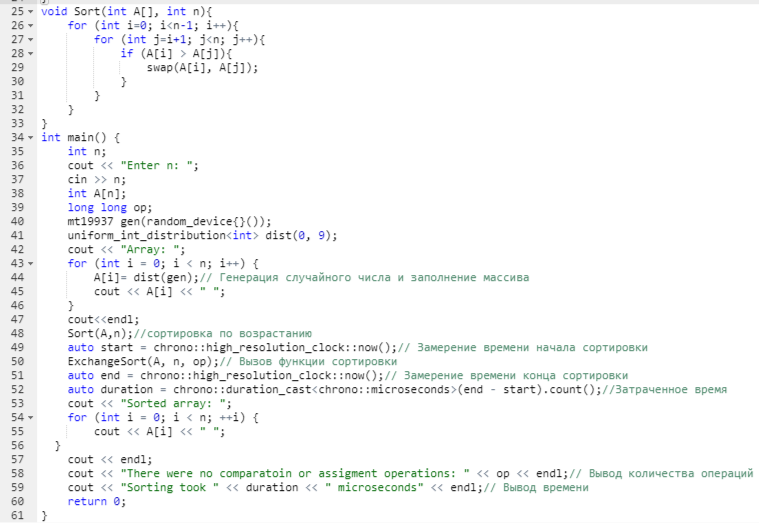


Рисунок 9 – Функция сортировки по возрастанию и функция main

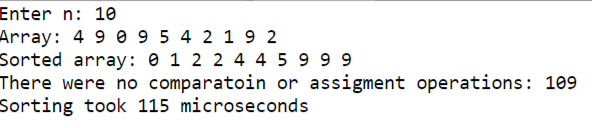


Рисунок 10 – Тестирование программы

Таблица 4. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.35 |  | 10099 |
| 1000 | 1.953 |  | 1000999 |
| 10000 | 229.273 |  | 100009999 |
| 100000 | 20276.833 |  | 10000099999 |
| 1000000 | 2305475.91 |  | 1000000999999 |

## **3.5 Построение графика**

Построим график функции роста Тп алгоритма простым обменом в худшем случае от размера массива на основе данных в таблице 4(рис.11).

### 

Рисунок 11 - График функции роста Тп алгоритма в лучшем случае

## 

## **3.6 Вывод**

Алгоритм сортировки простым обменом (также известный как пузырьковая сортировка) зависит от исходной упорядоченности массива. Если массив уже отсортирован, то время выполнения алгоритма будет минимальным, так как элементы не будут менять свои позиции. Однако, если массив отсортирован в обратном порядке, то алгоритм будет работать наихудшим образом, так как придется делать максимальное количество обменов.

Таким образом, можно сделать вывод, что алгоритм сортировки простым обменом зависит от исходной упорядоченности массива и его эффективность может значительно изменяться в зависимости от этого фактора.

# 

# 4 ЗАДАНИЕ №3

## **4.1 Формулировка задачи**

Вариант 2, в списке 14. Алгоритм 3 задания: простая вставка (Insertion sort).

Оценить эмпирически вычислительную сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1. Выполнить разработку и программную реализацию второго алгоритма согласно индивидуальному варианту.

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях в соответствии с форматом Таблицы 2 (на тех же массивах, что и в заданиях 1 и 2).

3. Определить ёмкостную сложность алгоритма от n.

4. На одном сравнительном графике отобразить функции Тп(n) двух алгоритмов сортировки в худшем случае.

5. Аналогично на другом общем графике отобразить функции Тп(n) двух алгоритмов сортировки для лучшего случая.

6. Выполнить сравнительный анализ полученных результатов для двух алгоритмов.

## **4.2 Описание выполнения алгоритма, доказательство корректности циклов и определение ситуаций лучшего, худшего и среднего случая.**

### **4.2.1 Описание выполнения алгоритма**

Алгоритм сортировки вставками работает следующим образом:

Начиная со второго элемента массива, сравниваем его со всеми предшествующими элементами и вставляем его на нужное место в уже отсортированной части массива.

Повторяем этот процесс для каждого элемента, поочередно перемещая неотсортированный элемент ближе к началу массива, пока не доходим до последнего элемента.

При сравнении и вставке элементов, мы сравниваем текущий элемент с предыдущими элементами, пока не найдем место для вставки.

Алгоритм завершен, когда весь массив станет отсортированным.

Этот алгоритм эффективно работает в случаях, когда массив уже частично отсортирован, что делает его хорошим выбором для многих реальных сценариев использования.

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

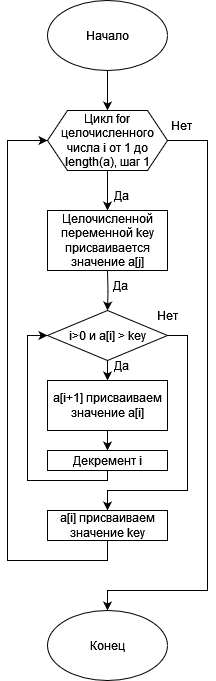


Рисунок 12 – Блок-схема алгоритма сортировки простыми вставками

### **4.2.2 Доказательство корректности циклов**

Найдём инварианты для внешнего и внутреннего цикла. Инвариантом внешнего цикла будет являться значение переменной i всегда меньше length(a).. Инвариантом для внутреннего цикла будет являться значение переменной i всегда больше 0 и a[i] больше key.

Докажем корректность циклов для чего нужно доказать их конечность. Цикл прохода через все элементы массива начинается со второго, сдвигая текущий элемент налево в уже упорядоченной части массива до тех пор, пока не наткнется на элемент, который меньше или равен ему, либо не достигнет начала массива. Вложенный цикл сдвигает все элементы отсортированной части, которые больше текущего элемента, вправо, пока не найдет подходящее место для него или не дойдет до начала массива. Следом текущий элемент помещается на свою позицию. Следовательно, циклы данного алгоритма конечны, а следовательно циклы алгоритма корректны.

### **4.2.3 Ситуации лучшего, среднего и худшего случаев**

Таблица 5-Псевдокод и анализ алгоритма сортировки вставками

| № | Алгоритм, записанный на псевдокоде | Количество выполнений оператора |
| --- | --- | --- |
| 1 | InsertionSort(a,n){ |  |
| 2 | for i←2 to length(a) do | n |
| 3 | key←a[j] | n-1 |
| 4 | while i>0 и a[i]>key do | n-1 |
| 5 | a[i+1]←a[i] | (tj - 1) |
| 6 | i←i-1 | (tj - 1) |
| 7 | оd |  |
| 8 | a[i]←key | n-1 |
| 9 | od |  |
| 10 | } |  |

В наилучшем случае, когда массив уже упорядочен по возрастанию, количество действий составит O(n). В среднем случае, когда массив содержит случайные числа, алгоритм будет иметь сложность O(n2). В худшем случае, когда массив упорядочен по убыванию, количество действий будет O(n2). Относительно времени выполнения: лучший случай - O(n), худший случай - O(n2). Ёмкостная сложность алгоритма оценивается как O(1).

## 

## **4.3 Разработка программы алгоритма простой вставки**

Разработаем программу алгоритма простым обменом на языке C++(рис.13,14). В данной программе используются следующие библиотеки и концепции:

<iostream>: для стандартного ввода-вывода данных.

<random>: для использования генератора случайных чисел и инструмента равномерного распределения.

<chrono>: для измерения времени выполнения сортировки.

<vector>: для динамического массива

using namespace std;: для удобства использования стандартного пространства имен.

Функция InsertionSort: основная функция сортировки вставками.

Использование генератора случайных чисел mt19937 и распределения uniform\_int\_distribution для заполнения массива случайными значениями.

Измерение времени начала и окончания сортировки с помощью <chrono>.

Подсчет времени выполнения сортировки в микросекундах.

Вывод отсортированного массива, количества операций и времени сортировки на экран.

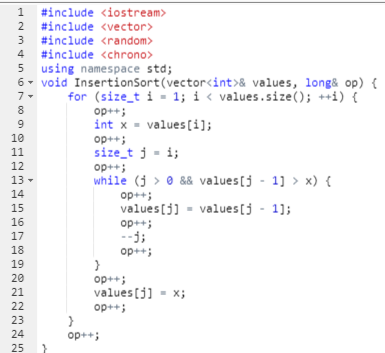


Рисунок 13 – Программа алгоритма сортировки простыми вставками



Рисунок 14 – Функция main для алгоритма сортировки простыми вставками

## **4.4 Тестирование**

Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 6. При n=10 результат будет предоставлен на рисунке 15.

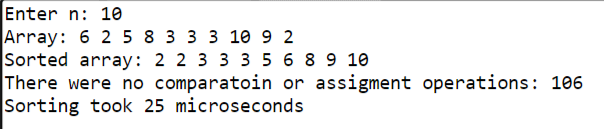


Рисунок 15 - Тестирование программы

Таблица 6. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.053 |  | 8263 |
| 1000 | 3.067 |  | 675037 |
| 10000 | 246.425 |  | 67332418 |
| 100000 | 22596.106 |  | 6759100756 |
| 1000000 | 2331918.14 |  | 678375901836 |

### 

## **4.5 Построение графика**

Построим график функции роста Тп алгоритма простым обменом от размера массива на основе данных в таблице 3(рис.16).



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

## **4.6 Тестирование программы при массиве упорядоченному по убыванию**

При упорядоченном массиве по убыванию алгоритм обладает O(n2), так как это худший случай. Допишем в уже имеющуюся программу библиотеку вектор для сортировки по убыванию(рис.17). Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 7. При n=10 результат будет предоставлен на рисунке 18.



Рисунок 17 – Функция main с сортировкой по убыванию

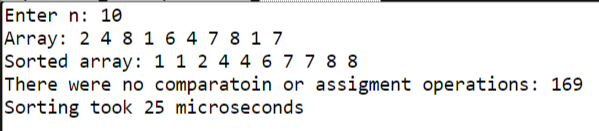


Рисунок 18 – Тестирование программы

Таблица 7. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.036 |  | 13810 |
| 1000 | 5.739 |  | 1353442 |
| 10000 | 334.059 |  | 135046111 |
| 100000 | 32347.719 |  | 13500396481 |
| 1000000 | 3393305.29 |  | 1350006437261 |

## **4.7 Построение графика**

Построим график функции роста Тп алгоритма простым обменом в худшем случае от размера массива на основе данных в таблице 7(рис.19).

### 

Рисунок 19 - График функции роста Тп алгоритма в худшем случае

## **4.8 Тестирование программы при массиве упорядоченному по возрастанию**

При упорядоченном массиве по возрастанию алгоритм обладает O(n), так как это лучший случай. Допишем в уже имеющуюся программу библиотеку вектор для сортировки по возрастанию(рис.20). Проверим работоспособность программы с разными размерами массивов (n=10, 100, 1000, 10000, 100000, 1000000). Отчет о результатах тестирования для n от 100 до 1000000 будет представлен в таблице 8. При n=10 результат будет предоставлен на рисунке 21.

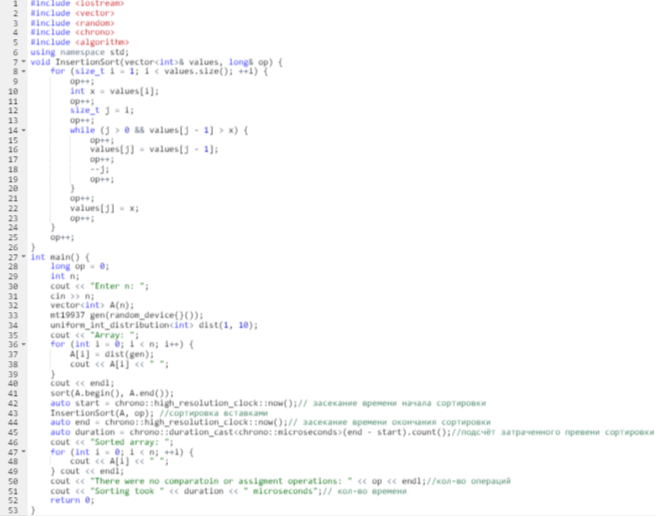


Рисунок 20 –Функция main с сортировкой по возрастанию

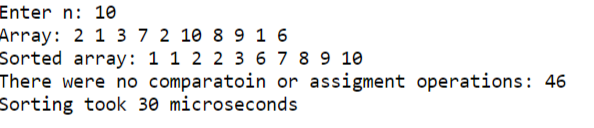


Рисунок 21 – Тестирование программы

Таблица 8. Сводная таблица результатов

| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| --- | --- | --- | --- |
| 100 | 0.001 |  | 496 |
| 1000 | 0.023 |  | 4996 |
| 10000 | 0.148 |  | 49996 |
| 100000 | 1.113 |  | 499996 |
| 1000000 | 13.127 |  | 4999996 |

## **4.9 Построение графика**

Построим график функции роста Тп алгоритма простым обменом в худшем случае от размера массива на основе данных в таблице 8(рис.22).

### 

Рисунок 22 - График функции роста Тп алгоритма в лучшем случае

## **4.10 Сравнение графиков двух алгоритмов**

**4.10.1 Сравнение двух алгоритмов сортировки в худшем случае**

Построим график сравнения простых сортировок обменом и вставкой в худшем случае(рис.23).

****

Рисунок 23 – График двух алгоритмов в худшем случае

Алгоритм простой вставкой более эффективный, чем алгоритм простым обменом, что можно увидеть на рисунке

**4.4.2 Сравнение двух алгоритмов сортировки в лучшем случае**

Построим график сравнения простых сортировок обменом и вставкой в лучшем случае(рис.24).



Рисунок 24 – График двух алгоритмов в лучшем случае

Алгоритм простой вставкой более эффективный, чем алгоритм простым обменом, что можно увидеть на рисунке

## **4.5 Вывод**

Простые сортировки обменом и вставкой являются двумя из самых простых алгоритмов сортировки. Оба алгоритма имеют время выполнения O(n2), что делает их не самым быстрым способом сортировки. Однако они легко понимаемы и реализуемы, что делает их удобными для использования в небольших наборах данных или обучающих целях.

Сортировка обменом (или пузырьковая сортировка) проста в реализации, но неэффективна на практике из-за большого количества обменов элементов. Сортировка вставкой, в свою очередь, эффективнее по сравнению с сортировкой обменом, так как требует меньше операций для сортировки элементов.

В целом, выбор между сортировкой обменом и вставкой зависит от конкретной ситуации. Если необходимо отсортировать небольшой массив данных и простота реализации является приоритетом, то можно использовать сортировку обменом. Однако, если необходимо улучшить производительность и эффективность сортировки, то стоит выбрать сортировку вставкой.

# 4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Простыми сортировками называют сортировки, которые базируются на простых операциях сравнения и обмена элементов, такие как сортировка пузырьком, сортировка вставками и сортировка выбором.
2. Понятие "внутренняя сортировка" означает, что вся сортировка массива происходит внутри этого массива, без использования дополнительной памяти для хранения других структур данных.
3. Основные операции при оценке сложности алгоритма сортировки - это число сравнений и число перемещений (или обменов) элементов.
4. При оценке эффективности алгоритма сортировки используются характеристики временной сложности (время выполнения) и емкостной сложности (используемая память).
5. Вычислительная сложность:

* Простого обмена: O(n2)
* Простой вставки: В среднем O(n2), в лучшем случае O(n)
* Простого выбора: O(n2) Емкостная сложность:
* Простого обмена, простой вставки, простого выбора: O(1) (так как они используют постоянное количество дополнительной памяти)

1. Условие Айверсона в сортировке обменом предполагает, что если на каком-то проходе не было ни одной пары элементов, требующих обмена, то сортировка завершена.
2. Алгоритмом, рассмотренным в данном задании, сортировался исходный массив 5 6 1 2 3 по сортировке пузырьком.
3. Вычислительная теоретическая сложность алгоритма сортировки пузырьком - O(n2), где n - количество элементов в массиве.

# 5 ВЫВОДЫ

В ходе практической работы были выполнены следующие задачи:

- Актуализированы знания и приобретены умения по эмпирическому определению вычислительной сложности;

- Проведён анализ алгоритмов простой сортировки обменом и вставками;

- Были реализованы программы для алгоритмов простой сортировки обменом и вставками;

- Проведённое тестирование программ для алгоритмов простой сортировки обменом и вставками;

- Построены графики функции роста Тп алгоритмов простой сортировки обменом и вставками от размера массива n.

- Произведено сравнение алгоритмов простой сортировки обменом и вставками на основе анализа, результатов тестирования и графиков.

Таким образом, главную цель практической работы, а именно актуализация знаний и приобретение практических умений по эмпирическому определению вычислительной сложности алгоритмов, 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. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

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

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

7. 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).