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

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_gjdgxs)

[2 ЗАДАНИЕ №1 5](#_30j0zll)

[2.1 Формулировка задачи (Вариант 3) 5](#_1fob9te)

[2.2 Математическая модель решения алгоритма 6](#_3znysh7)

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

[2.2.2 Доказательство корректности циклов алгоритма сортировки простым выбором 7](#_tyjcwt)

2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки простого выбора 8

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

[2.3.1 Реализация алгоритма сортировки простого выбора на языке C++ 9](#_4d34og8)

[2.3.2 Тестирование 11](#_2s8eyo1)

2.3.3 Построение графика 11

[2.4 Вывод по заданию №1 12](#_3rdcrjn)

3 ЗАДАНИЕ №2 13

[3.1 Формулировка задачи 13](#_26in1rg)

[3.2 Тестирование программы 13](#_lnxbz9)

[3.2.1 Массив упорядоченный по убыванию 13](#_35nkun2)

3.2.2 Массив упорядоченный по возрастанию 16

[3.3 Вывод по заданию №2 19](#_1ksv4uv)

[4 ЗАДАНИЕ №3 20](#_44sinio)

[4.1 Формулировка задания 21](#_2jxsxqh)

[4.2 Математическая модель решения алгоритма 21](#_z337ya)

[4.2.1 Описание выполнения и блок-схема алгоритма сортировки простым выбором 21](#_3j2qqm3)

[4.2.2 Доказательство корректности циклов алгоритма сортировки простым обменом 22](#_1y810tw)

[4.2.3 Определение ёмкостной сложности, ситуации лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки простым обменом 22](#_4i7ojhp)

[4.3 Реализация алгоритма на языке C++, проведение тестирования и построение графика 24](#_2xcytpi)

[4.3.1 Реализация алгоритма сортировки простым обменом на языке C++ 24](#_1ci93xb)

[4.3.2 Тестирование при случайном заполнении массива 25](#_3whwml4)

[4.3.3 Построение графика 26](#_2bn6wsx)

[4.3.4 Тестирование при упорядоченном по убыванию элементов массива и построение графика 27](#_qsh70q)

4.3.5 Массив упорядоченный по возрастанию 30

[4.4 Сравнение графиков двух алгоритмов сортировки из задания 1 и 3 34](#_2p2csry)

[4.5 Выводы по заданию №3 36](#_147n2zr)

[5 КОНТРОЛЬНЫЕ ВОПРОСЫ 38](#_3o7alnk)

[6 ВЫВОДЫ 41](#_23ckvvd)

[7 ЛИТЕРАТУРА 42](#_ihv636)

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи**

(Вариант 3, в списке 18)

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 простой вставки. 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 сортировки простым выбором**

Имеется входной массив длиной n. Начинающаяся с пустой упорядоченной части, в неупорядоченной части выбирается элемент с минимальным значением. Этот минимальный элемент обменивается местами с первым элементом неупорядоченной части и добавляется в упорядоченную часть. Затем процесс повторяется для оставшихся n-1 элементов: выбирается минимальный элемент из неупорядоченной части, обменивается с начальным элементом неупорядоченной части и добавляется в упорядоченную часть. Процесс продолжается до тех пор, пока неупорядоченная часть не будет содержать только один элемент.

Таким образом, алгоритм сортировки выбором постепенно упорядочивает весь массив, перемещая минимальные элементы из неупорядоченной части в упорядоченную.

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

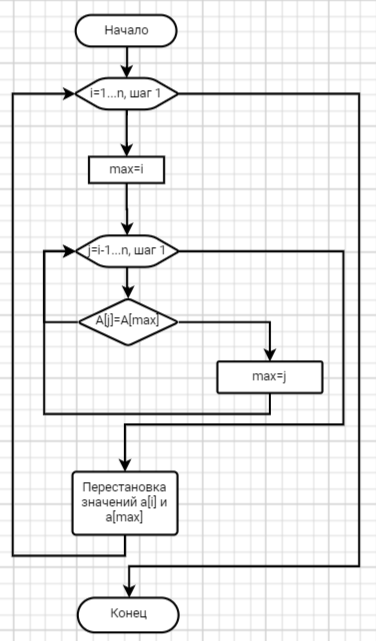


Рисунок 1 – Блок-схема алгоритма сортировки простым выбором

### **2.2.2 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки простого выбора**

Таблица 1-Псевдокод и анализ алгоритма сортировки выбором

|  |  |  |
| --- | --- | --- |
| № | Алгоритм, записанный на псевдокоде | Количество выполнений оператора |
| 1 | SelectionSort(a,n){ |  |
| 2 | for i←1 to n-1 do | n |
| 3 | imax←i | n-1 |
| 4 | for j←i+1 to n do | (n2-n)/2 |
| 5 | if a[j]←a[imax] then | (n2-n)/2-(n-1) |
| 6 | imax←j | (n2-n)/2-(n-1) |
| 7 | Endlf  оd |  |
| 8 | swap(a[i],a[imax]) | 3\*(n-1) |
| 9 | od |  |
| 10 | } |  |

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

b. Функции роста времени для данного метода сортировки:

Лучший случай: O(n2)

Худший случай: O(n2)

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

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

### **2.3.1 Реализация алгоритма сортировки простого выбора на языке C++**

Для реализации данного алгоритма на языке C++ (рис.2,3) понадобятся библиотеки iostream, random и chrono. Для подсчета количества операций присваивания или сравнения введена переменная o типа long. Проведём тестирование программы с заданным размером массива n=10 (рис.4), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование генерация случайных чисел в массиве. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 1. Воспользуемся библиотекой chrono для засекания времени. Для более точных результатов в программе будем рассматривать наносекунды.

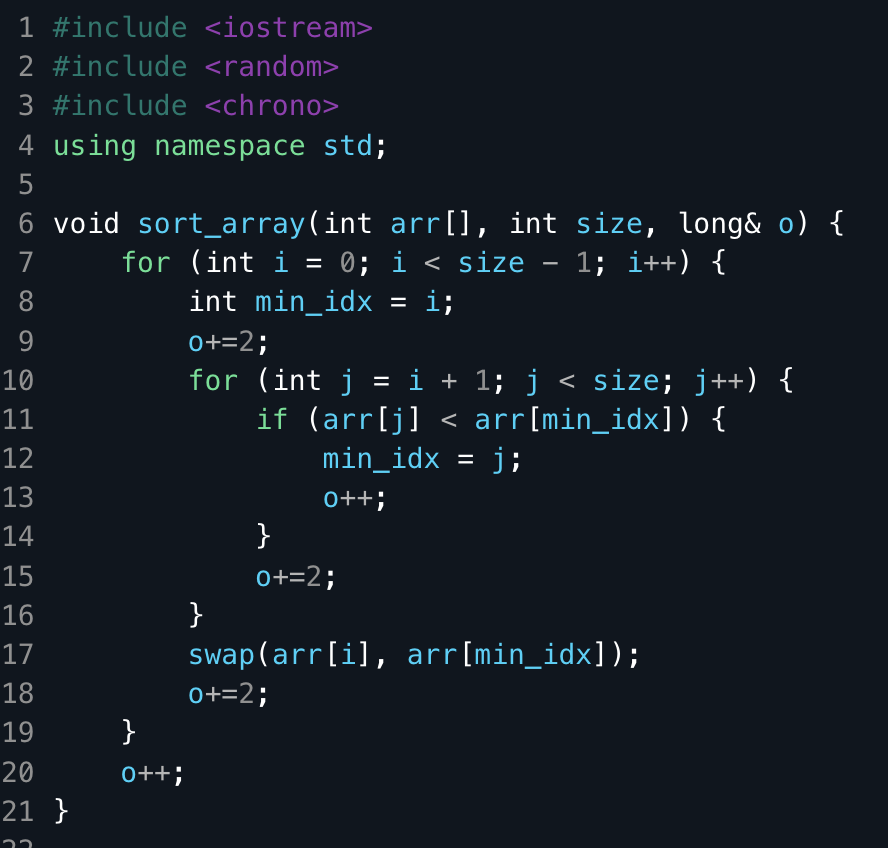


Рисунок 2 – Программа алгоритма сортировки простым выбором



Рисунок 3 – Функция main для алгоритма сортировки простым выбором

### **2.3.2 Тестирование**

Необходимо провести тестирование программы с массивами различного размера: n=10(рис.4), n=100, n=1000, n=10000, n=100000, n=1000000. Для этого будем использовать случайную генерацию чисел. Результаты тестирования для массивов размером от 100 до 1000000 будут представлены в таблице 2. Будем измерять время в наносекундах для получение максимально точных результатов, а затем для заполнения таблицы переведем данные в миллисекунды.

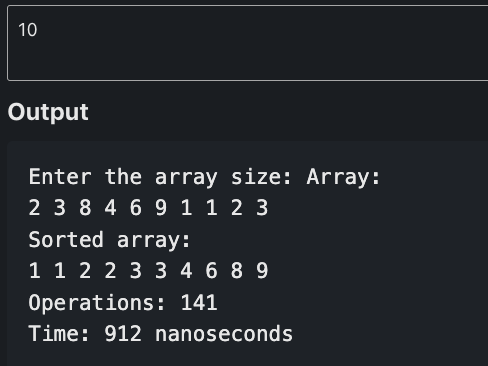


Рисунок 4 - Тестирование программы при n=10

Таблица 2. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,029 | - | 10491 |
| 1000 | 1,5 | - | 1004848 |
| 10000 | 194,41 | - | 100049362 |
| 100000 | 119248,85 | - | 10000493142 |
| 1000000 | 30614531,41 | - | 1000004931038 |

### 

### **2.3.3 Построение графика**

На основе данных, представленных в таблице 2, будет построен график зависимости времени выполнения алгоритма от размера массива n (рис.5).

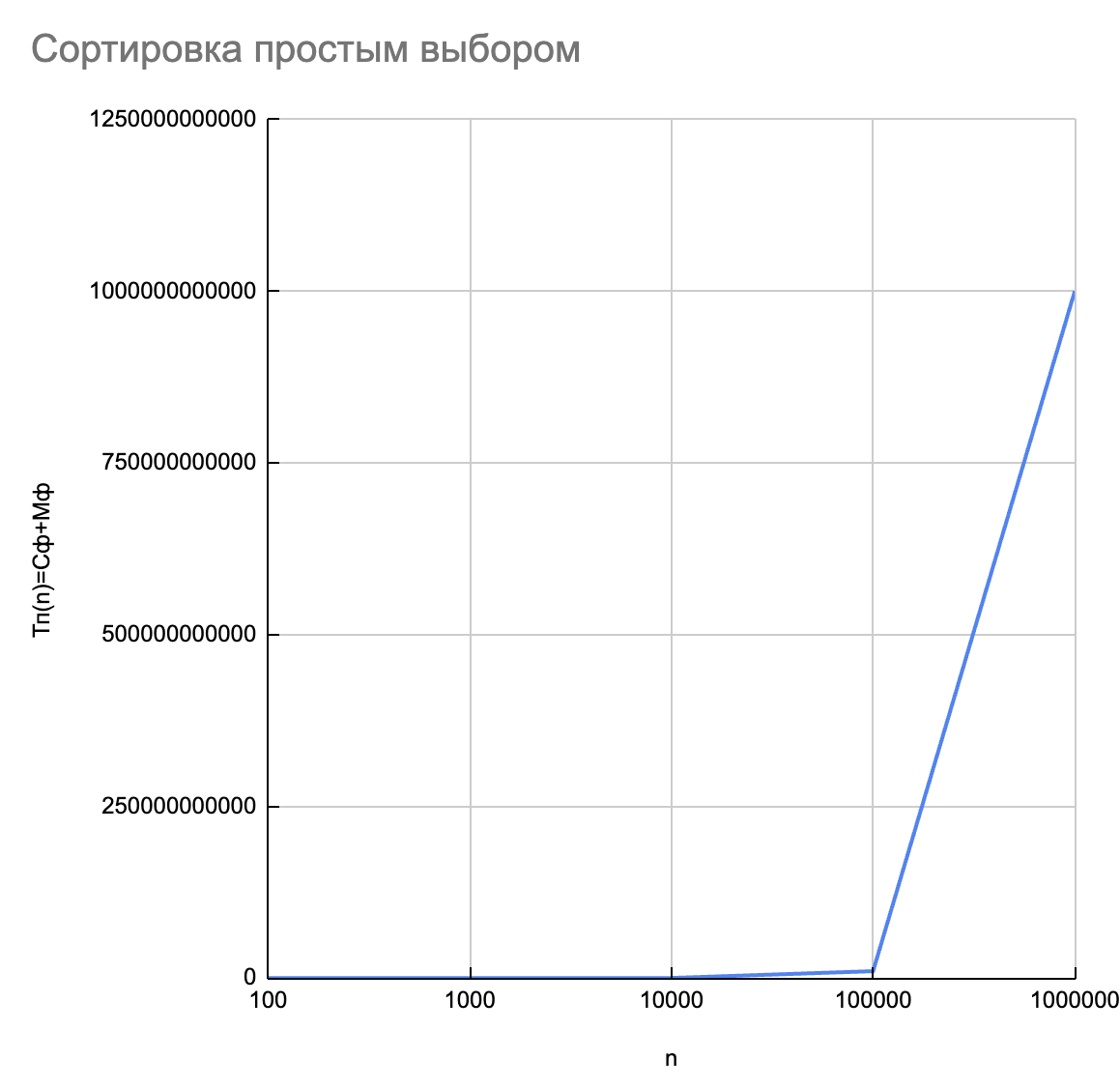


Рисунок 5 - График функции роста Тп этого алгоритма от размера массива n

## **2.4 Вывод по заданию №1**

Эмпирическая вычислительная сложность алгоритма сортировки выбором основана на скорости роста функции, которая определяет, как быстро увеличивается количество операций (сравнений, перестановок и т. д.) в зависимости от размера входных данных (например, массива).

Для алгоритма сортировки выбором, его эмпирическая вычислительная сложность обычно составляет O(n2), где n - количество элементов в сортируемом массиве. Это означает, что время работы алгоритма будет пропорционально квадрату размера входных данных.

Исходя из этого, можно сделать вывод о том, что с увеличением размера входного массива время работы алгоритма сортировки выбором будет увеличиваться квадратично. То есть при удвоении размера массива время работы алгоритма увеличится примерно в четыре раза.

Важно помнить, что хотя алгоритм сортировки выбором прост в реализации, его квадратичная сложность делает его неэффективным для больших объемов данных. В таких случаях более предпочтительными могут быть более эффективные алгоритмы сортировки, такие как сортировка слиянием или быстрая сортировка.

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи**

Оценить вычислительную сложность алгоритма простой сортировки выбором 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;

b. строго в возрастающем порядке значений, результаты представить в сводной таблице;

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

## **3.2 Тестирование программы**

### **3.2.1 Массив упорядоченный по убыванию**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort и объект greater<int>() в main(рис.6). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем исполнение программы при n=10 (рис.7). Результаты тестирования запишем в таблицу 3.



Рисунок 6 – Функция main программы простой сортировки выбором с отсортированными значениями по убыванию

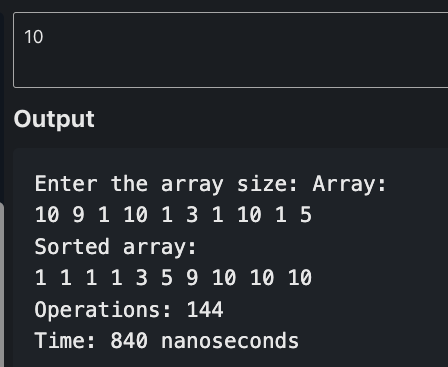


Рисунок 7 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Таблица 3. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,027 |  | 10558 |
| 1000 | 1,75 |  | 1005786 |
| 10000 | 144,88 |  | 100058872 |
| 100000 | 1155,46 |  | 10000551798 |
| 1000000 | 526547,143 |  | 10000055585297 |

На основе данных, представленных в таблице 3, будет построен график зависимости времени выполнения алгоритма от размера массива n (рис.8).

### 

Рисунок 8 - График функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по убыванию от размера массива n

### **3.2.2 Массив упорядоченный по возрастанию**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort в main(рис.9). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем исполнение программы при n=10 (рис.10). Результаты тестирования продемонстрируем в таблице 4.



Рисунок 9 – Функция main программы простой сортировки выбором с отсортированными значениями по возрастанию

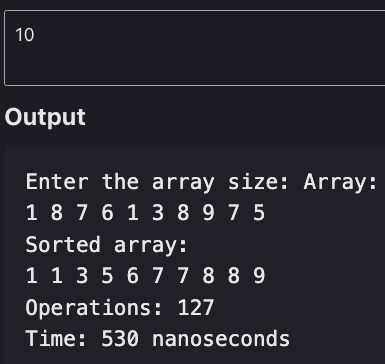


Рисунок 10 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Таблица 4. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,031 |  | 10297 |
| 1000 | 2,25 |  | 1002997 |
| 10000 | 173,13 |  | 100029997 |
| 100000 | 20624,73 |  | 10000299997 |
| 1000000 | 1815463,515 |  | 1000002999997 |

На основе полученных данных, продемонстрированных в таблице 4, построим график функции роста Тп этого алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.11).

### 

Рисунок 11 - График функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по возрастанию от размера массива n

## **3.3 Вывод по заданию №2**

Алгоритм сортировки выбором не зависит от исходной упорядоченности массива. Это означает, что даже если массив уже отсортирован или частично отсортирован, алгоритм все равно будет выполнять одинаковое количество операций сравнения и обмена элементов. Алгоритм сортировки выбором всегда выполняет одинаковое количество операций, вне зависимости от того, как упорядочен исходный массив.

# 4 ЗАДАНИЕ №3

## **4.1 Формулировка задания**

Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию алгоритма простого обмена.

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях (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 Математическая модель решения алгоритма**

### **4.2.1 Описание выполнения и блок-схема алгоритма сортировки простым выбором**

Пузырьковая сортировка, также известная как сортировка простым обменом, - один из наиболее популярных алгоритмов, который обычно изучают уже в школьном курсе информатики. В этом алгоритме значения массива сравниваются попарно и при необходимости меняются местами, чтобы упорядочить элементы по возрастанию. После каждого прохода по массиву самый большой элемент занимает свою окончательную позицию в конце массива. Алгоритм завершается, когда либо вся область массива отсортирована, либо при очередном проходе не было совершено ни одной перестановки (по критерию Айверсона).

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

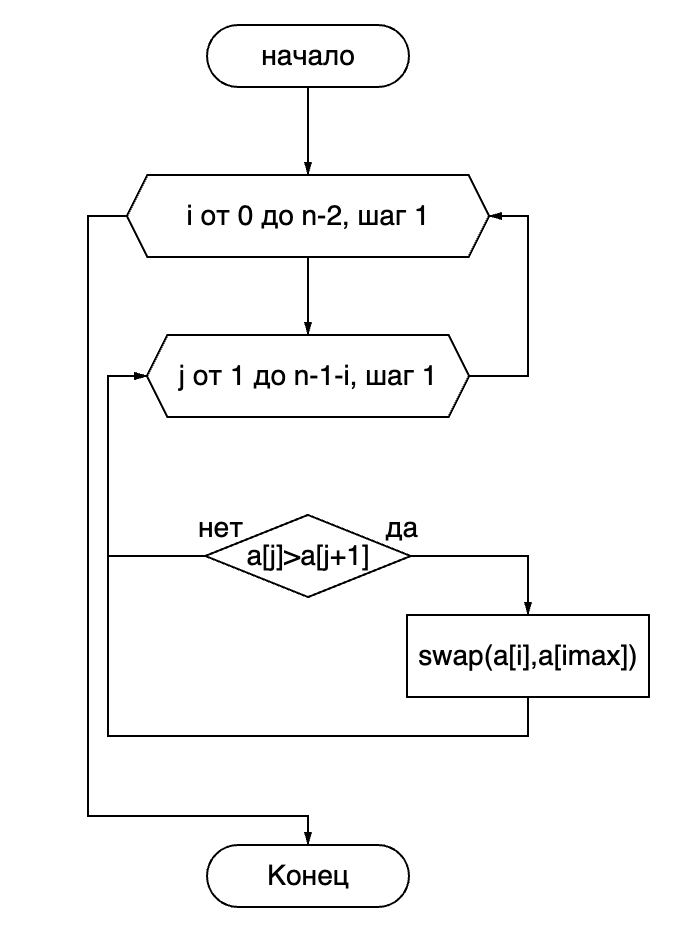


Рисунок 12 – Блок-схема алгоритма сортировки простым выбором

### **4.2.2 Доказательство корректности циклов алгоритма сортировки простым обменом**

### **4.2.3 Определение ёмкостной сложности, ситуации лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки простым обменом**

Таблица 5. Псевдокод и анализ алгоритма сортировки выбором

|  |  |  |
| --- | --- | --- |
| № | 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 лучшем случае массив уже отсортирован, что минимизирует количество операций сравнения и перемещения до O(n). В среднем случае, когда массив заполнен случайными числами, алгоритм имеет сложность O(n2). В худшем случае, когда массив отсортирован в обратном порядке, количество операций также будет O(n2).

Функции роста времени: в лучшем случае: O(n), в худшем случае: O(n2)

Для данного метода сортировки, время исполнения увеличивается квадратично с ростом размера входного массива. Поэтому 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 равна O(1).

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

### **4.3.1 Реализация алгоритма сортировки простым обменом на языке C++**

Для реализации данного алгоритма на языке C++ (рис.13) понадобятся библиотеки iostream, random и chrono. Для подсчета количества операций присваивания или сравнения введена переменная o типа long. Проведём тестирование программы с заданным размером массива n=10 (рис.14), n=100, n=1000, n=10000, n=100000, n=1000000. Чтобы провести данной тестирование генерация случайных чисел в массиве. Результаты тестирования от n=100 до n=1000000 будут продемонстрированы в таблице 6. Воспользуемся библиотекой chrono для засекания времени. Для более точных результатов в программе будем рассматривать наносекунды.



Рисунок 13 – Программа алгоритма сортировки простым обменом



Рисунок 14 – Функция main для алгоритма сортировки простым обменом

### **4.3.2 Тестирование при случайном заполнении массива**

Необходимо провести тестирование программы с массивами различного размера: n=10(рис.15), n=100, n=1000, n=10000, n=100000, n=1000000. Для этого будем использовать случайную генерацию чисел. Результаты тестирования для массивов размером от 100 до 1000000 будут представлены в таблице 6. Будем измерять время в наносекундах для получение максимально точных результатов, а затем для заполнения таблицы переведем данные в миллисекунды.

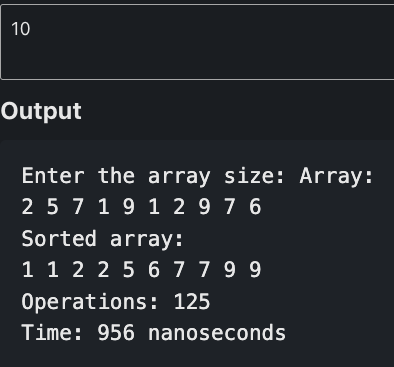


Рисунок 15 - Тестирование программы при n=10

Таблица 6. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,051 |  | 12542 |
| 1000 | 2,28 |  | 1229153 |
| 10000 | 253,24 |  | 122242692 |
| 100000 | 36145,93 |  | 12251473857 |
| 1000000 | 521553,56 |  | 1220495426936 |

### **4.3.3 Построение графика**

На основе данных, представленных в таблице 6, будет построен график зависимости времени выполнения алгоритма от размера массива n (рис.16).

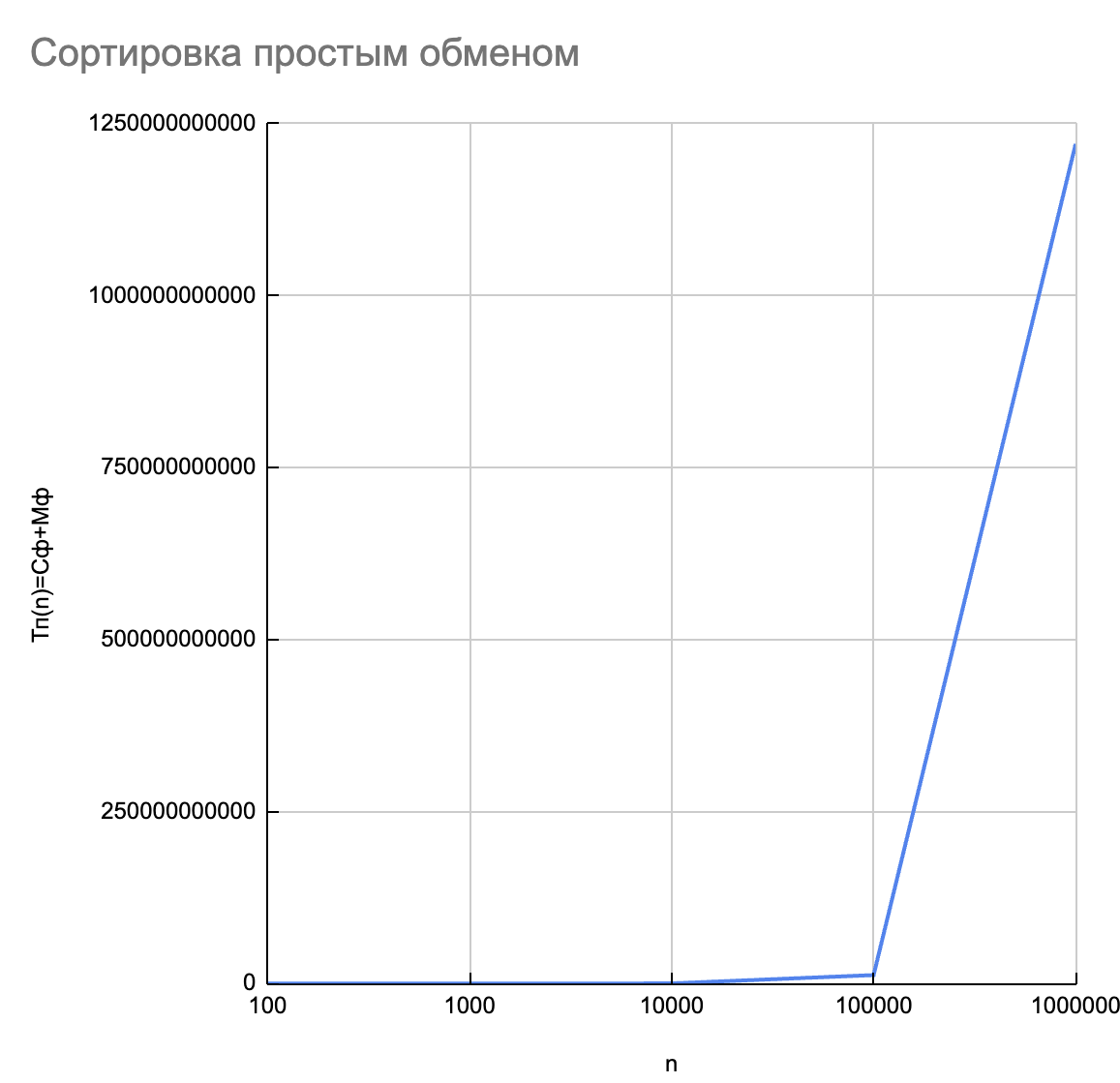


Рисунок 16 - График функции роста Тп этого алгоритма от размера массива n

### **4.3.4 Тестирование при упорядоченном по убыванию элементов массива и построение графика**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort и объект greater<int>() в main(рис.17). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем исполнение программы при n=10 (рис.18).



Рисунок 17 – Функция main программы простой сортировки обменом с отсортированными значениями по убыванию

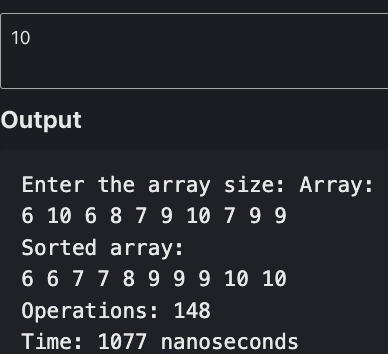


Рисунок 18 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Таблица 7. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,043 |  | 14553 |
| 1000 | 5,626 |  | 1450642 |
| 10000 | 326,24 |  | 145002646 |
| 100000 | 36184,53 |  | 14500014626 |
| 1000000 | 423256,56 |  | 1450000035642 |

На основе данных, представленных в таблице 7, будет построен график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по убыванию (рис.19).

### 

Рисунок 19 - График функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по убыванию от размера массива n

### **4.3.5 Массив упорядоченный по возрастанию**

Проведется тестирование программы на массивах различного размера: 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных в порядке убывания. Применим функцию sort в main(рис.20). Чтобы применить эту функцию понадобится библиотека algorithm. Продемонстрируем исполнение программы при n=10 (рис.21).

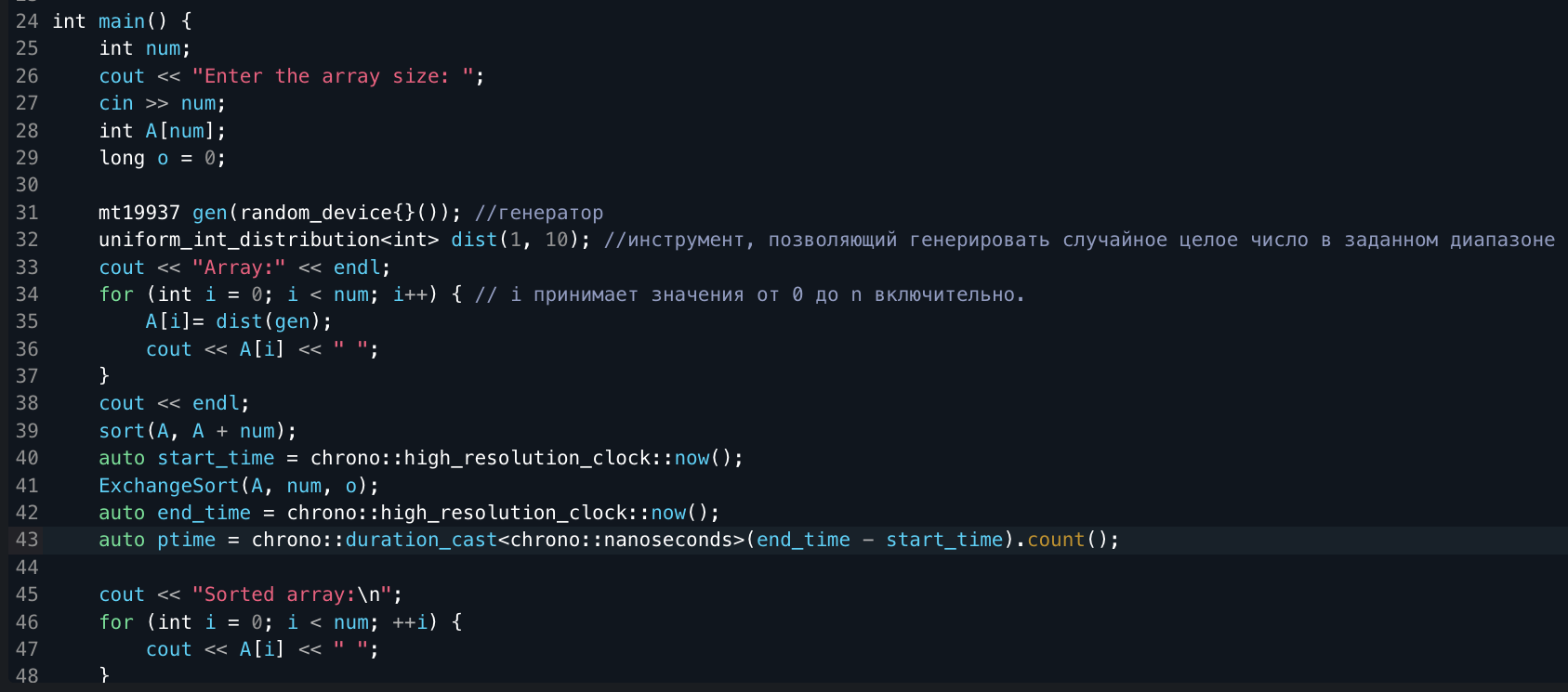


Рисунок 20 – Функция main программы простой сортировки обменом с отсортированными значениями по возрастанию

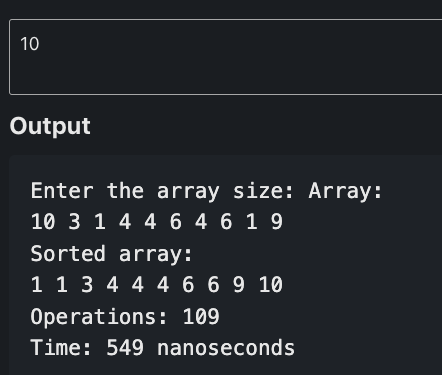


Рисунок 21 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Таблица 8. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0,05 |  | 10099 |
| 1000 | 2,14 |  | 1000999 |
| 10000 | 235,53 |  | 100009999 |
| 100000 | 15385,53 |  | 10000099999 |
| 1000000 | 5251875,531 |  | 1000000999999 |

На основе данных, представленных в таблице 8, будет построен график зависимости времени выполнения алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.22).

### 

Рисунок 22 - График функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по возрастанию от размера массива n

## **4.4 Сравнение графиков двух алгоритмов сортировки из задания 1 и 3**

**4.4.1 Отображение функции Тп(n) двух алгоритмов сортировки в худшем случае**

На основе информации из таблицы 3 и таблицы 7 будет создан график, показывающий, как время выполнения алгоритма изменяется в зависимости от размера массива n(рис.23).Этот график будет использоваться для сравнения производительности различных алгоритмов сортировки.

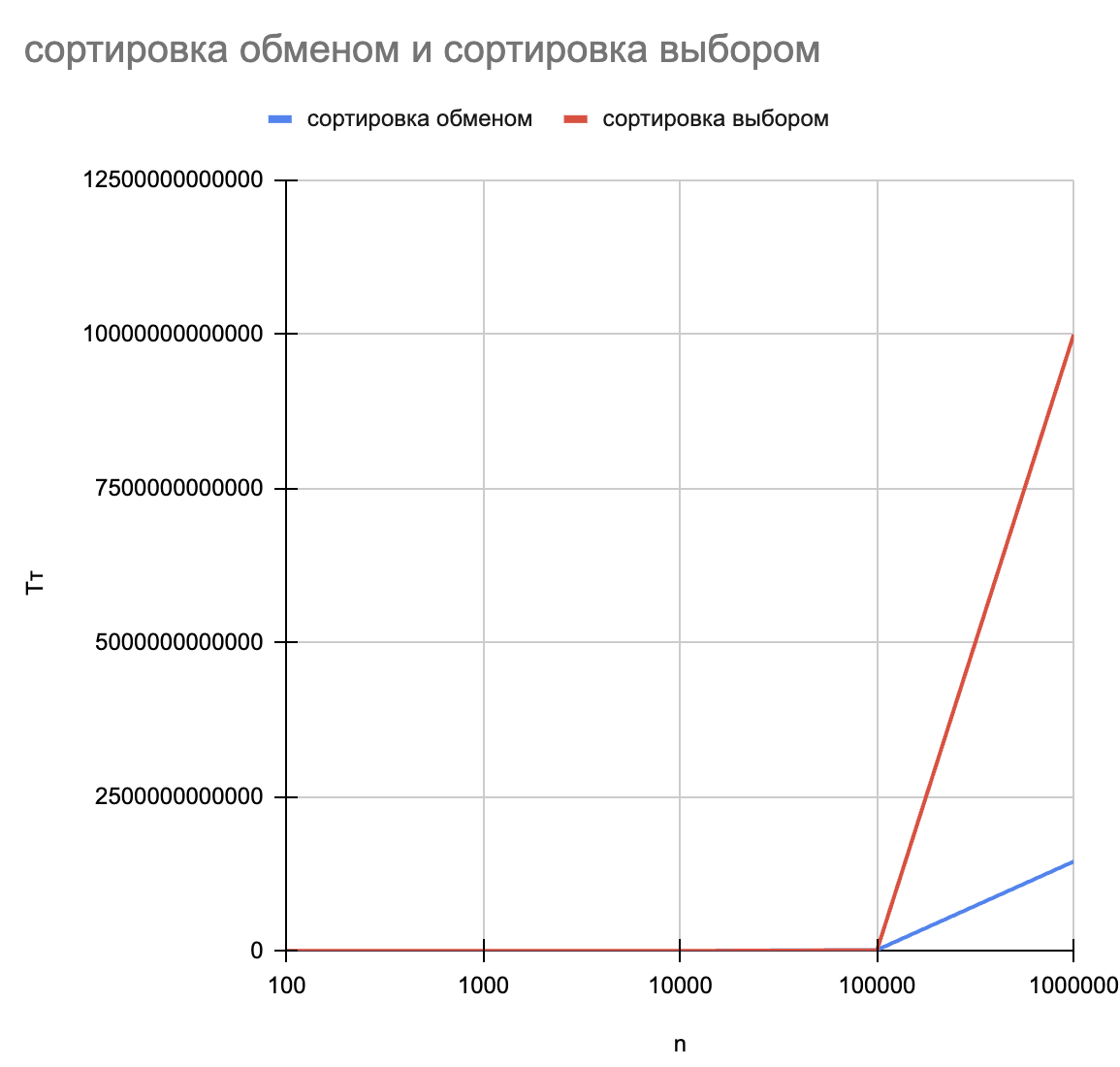


Рисунок 23 – График двух алгоритмов в худшем случае

В итоге можно заключить, что в наихудшем случае алгоритм сортировки простым обменом менее эффективен, чем алгоритм сортировки простого выбора.

**4.4.2 Отображение функции Тп(n) двух алгоритмов сортировки в лучшем случае**

Используя данные из таблиц 4 и 8, мы создадим график времени выполнения алгоритма в зависимости от размера массива n для анализа различий в росте алгоритмов сортировки(рис.24).

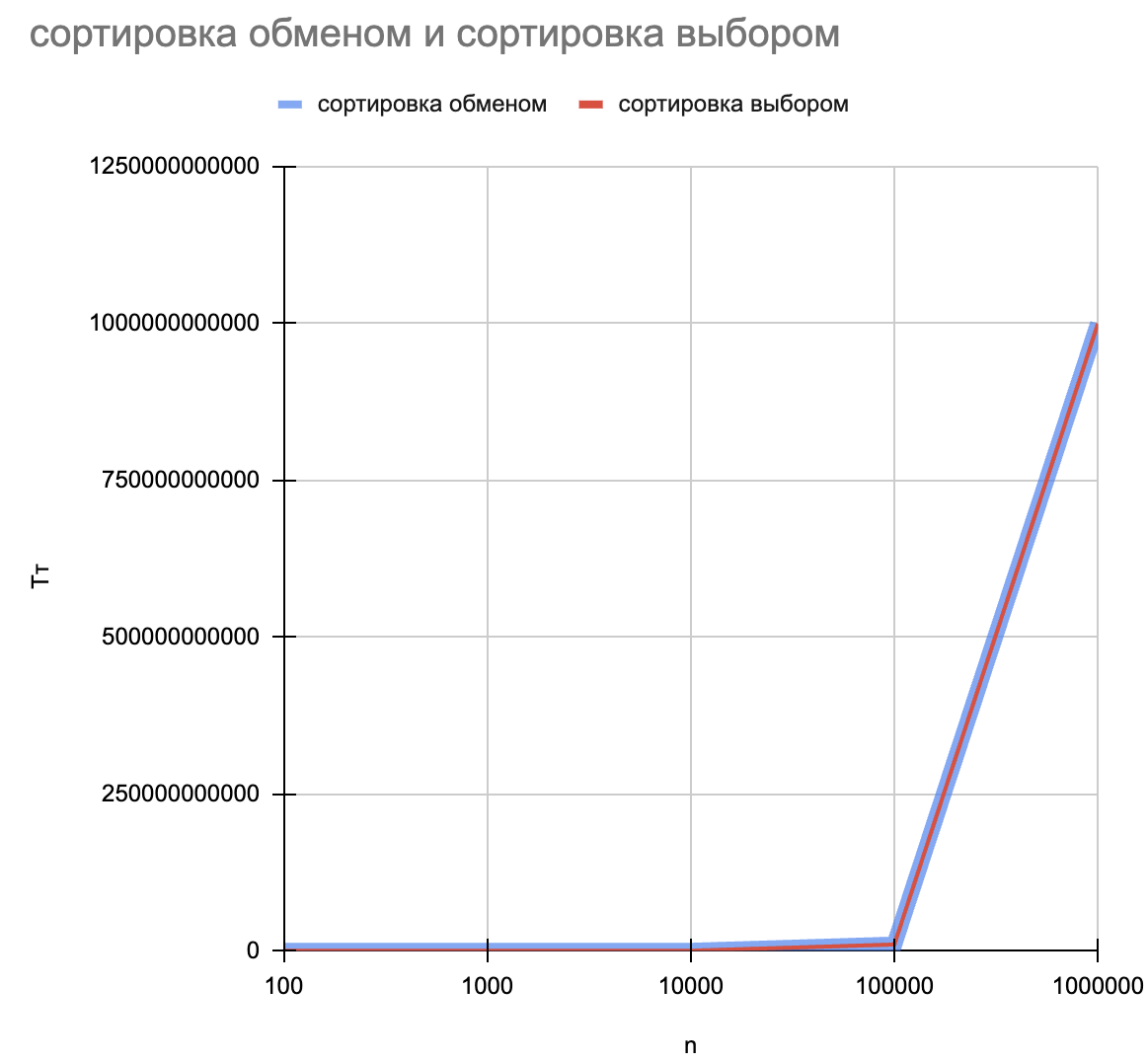


Рисунок 24 – График двух алгоритмов в лучшем случае

Более эффективным оказывается алгоритм сортировки простым обменом по сравнению с сортировкой простым выбором, хотя они почти одинаковы в лучшем случае.

## **4.5 Выводы по заданию №3**

Сравнение двух методов сортировки позволяет выявить различия в их эффективности. Один из методов, известный как пузырьковая сортировка или сортировка простым обменом, сравнивает и меняет соседние элементы в массиве. Его временная сложность составляет O(n2) из-за большого числа обменов, что делает его неэффективным для больших наборов данных. Другой метод, сортировка простым выбором, находит минимальный элемент в массиве и перемещает его на первую позицию. Временная сложность этого метода также O(n2) в худшем случае, но он требует меньше обменов по сравнению с пузырьковой сортировкой. Следовательно, в некоторых случаях сортировка простым выбором может быть более эффективной, особенно при работе с небольшими наборами данных. Однако оба метода имеют низкую производительность для больших массивов данных и не являются оптимальным выбором для сортировки крупных объемов данных.

# 5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие сортировки называют простыми?

Простыми сортировками обычно называют сортировки с простыми и понятными шагами исполнения, например, сортировка пузырьком, сортировка вставками, сортировка выбором и др.

2. Что означает понятие «внутренняя сортировка»?

Понятие "внутренняя сортировка" означает, что все операции сортировки происходят непосредственно в памяти компьютера, где хранятся сортируемые данные, без необходимости использования внешних ресурсов, таких как дополнительные диски.

3. Какие операции считаются основными при оценке сложности алгоритма сортировки?

Основные операции, которые учитываются при оценке сложности алгоритма сортировки, включают в себя сравнения элементов массива и перемещения элементов для их правильной упорядоченности.

4. Какие характеристики сложности алгоритма используются при оценке эффективности алгоритма?

При оценке эффективности алгоритма сортировки используются следующие характеристики сложности:

* Временная сложность (сколько операций требуется для выполнения алгоритма в зависимости от размера входных данных);
* Пространственная сложность (сколько памяти потребуется для выполнения алгоритма);
* Стабильность (сохраняется ли порядок элементов при равных значениях);
* Стабильность наихудшего случая (как поведет себя алгоритм в самом неэффективном случае).

5. Какая вычислительная и емкостная сложность алгоритма: простого обмена, простой вставки, простого выбора?

Простой обмен (сортировка пузырьком):

• Вычислительная сложность: O(n2)

• Емкостная сложность: O(1)

Простая вставка:

• Вычислительная сложность: О(n2) в худшем случае, O(n) в лучшем случае

• Емкостная сложность: O(1)

Простой выбор:

• Вычислительная сложность: O(n2)

• Емкостная сложность: O(1)

6. Какую роль в сортировке обменом играет условие Айверсона?

Условие Айверсона определяет порядок сортировки элементов в алгоритме сортировки простым обменом (сортировка пузырьком). Оно позволяет уменьшить количество сравнений элементов, так как при каждом проходе алгоритма самый большой элемент "всплывает" на правильное место в конце массива.

7. Определите, каким алгоритмом, рассмотренным в этом задании, сортировался исходный массив 5 6 1 2 3. Шаги выполнения сортировки:

1. 1 5 6 2 3
2. 1 2 5 6 3
3. 1 2 3 5 6

По шагам выполнения сортировки исходного массива 5 6 1 2 3 можно сказать, что использовалась сортировка пузырьком (простой обмен). Последний шаг показывает отсортированный массив, что является результатом сортировки обменом.

8. Какова вычислительная теоретическая сложность алгоритма сортировки, рассмотренного в вопросе 7.

Вычислительная теоретическая сложность алгоритма сортировки простого обмена в худшем случае составляет O(n2), что означает квадратичную зависимость от размера входных данных.

# 6 ВЫВОДЫ

В ходе практической работы были выполнены следующие задачи:

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

- Проведён анализ алгоритмов простой сортировки выбором и обменом;

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

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

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

- Произведено сравнение алгоритмов простой сортировки выбором и обменом 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.

# 7 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).