|  |
| --- |
|  |
| 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: Жаворонкова А.А. |
|  |  |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ №1 4](#_30j0zll)

[2.1 Формулировка задачи 4](#_1fob9te)

[2.2 Математическая модель решения алгоритма 5](#_3znysh7)

[2.2.1 Описание выполнения и блок-схема алгоритма простой сортировки вставками 5](#_2et92p0)

[2.2.2 Доказательство корректности циклов алгоритма простой сортировки вставками 6](#_tyjcwt)

[2.2.3 Определение ёмкостной сложности, ситуации лучшего, среднего и худшего случая и функции роста времени работы алгоритма простой сортировки вставками 7](#_3dy6vkm)

[2.3 Реализация алгоритма на языке C++, проведение тестирования и построение графика 8](#_4d34og8)

[2.3.1 Реализация алгоритма простой сортировки вставками на языке C++ 8](#_2s8eyo1)

[2.3.2 Тестирование и построение графика 9](#_17dp8vu)

[2.3.3 Построение графика 10](#_3rdcrjn)

[2.4 Вывод по заданию №1 11](#_26in1rg)

[3 ЗАДАНИЕ №2 12](#_lnxbz9)

[3.1 Формулировка задачи 12](#_35nkun2)

[3.2 Тестирование программы 12](#_1ksv4uv)

[3.2.1 Массив упорядоченный по убыванию 12](#_44sinio)

3.2.2 Массив упорядоченный по возрастанию 16

[3.3 Вывод по заданию №2 18](#_3j2qqm3)

[4 ЗАДАНИЕ №3 19](#_1y810tw)

[3.1 Формулировка задания 19](#_4i7ojhp)

[3.2 Описание математической модели 19](#_2xcytpi)

[3.2.1 Описание выполнения и блок-схема первого алгоритма 19](#_1ci93xb)

[3.3 Доказательство корректности циклов 20](#_2bn6wsx)

[3.4 Определение вычислительной сложности алгоритма 20](#_qsh70q)

[3.5 Реализация алгоритма на языке С++ 21](#_3as4poj)

[3.6 Тестирование 22](#_1pxezwc)

[3.7 Выводы по заданию №2 24](#_49x2ik5)

[4 КОНТРОЛЬНЫЕ ВОПРОСЫ 25](#_2p2csry)

[5 ВЫВОДЫ 29](#_147n2zr)

[6 ЛИТЕРАТУРА 30](#_3o7alnk)

# 1 ЦЕЛЬ

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

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи**

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 Математическая модель решения алгоритма**

### **2.2.1 Описание выполнения и блок-схема алгоритма сортировки простыми вставками**

Последовательное добавление элементов из неотсортированной к уже отсортированной части массива с сохранением в ней упорядоченности.

Эта сортировка обладает естественным поведением, т.е. алгоритм работает быстрее для частично упорядоченного массива. Алгоритм устойчив – элементы с одинаковыми ключами не переставляются.

Отсортированной частью массива по умолчанию считаем начальный элемент. На первом шаге следующий второй элемент (он же первый элемент неотсортированной части) сравнивается с первым (крайним справа элементом отсортированной части).

Если он больше первого, то остаётся на своём месте и просто включается в отсортированную часть, а остальная (неотсортированная) часть массива остаётся без изменений.

Если же он меньше первого, то последовательно сравнивается с элементами в упорядоченной части (начиная с наибольшего – крайнего справа), пока не встретится элемент меньший. После чего происходит вставка на место первого, большего его в отсортированной части, со сдвигом всех элементов правее в отсортированной части на одну позицию вправо.

Этот процесс повторяется аналогично для всех последующих элементов неотсортированной части исходного массива.

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

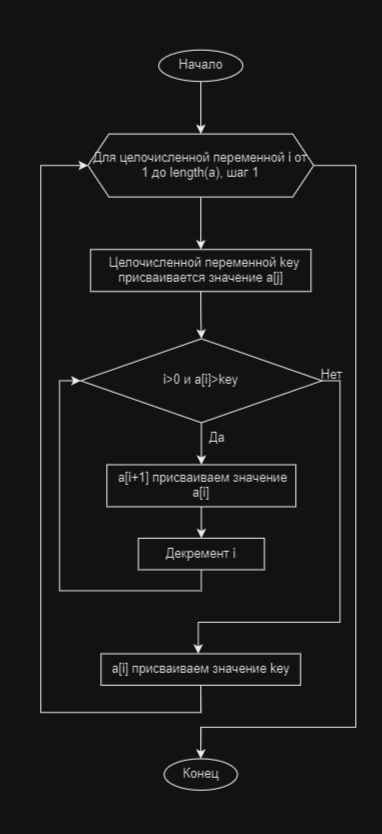


Рисунок 1 – Блок-схема алгоритма сортировки простыми вставками

### **2.2.2 Доказательство корректности циклов алгоритма сортировки простыми вставками**

Инвариант для внешнего цикла: значение переменной i всегда меньше length(a).

Инвариант для внутреннего цикла: значение переменной i всегда больше 0 и a[i] больше key.

Докажем конечность циклов. Внешний цикл for проходит через все элементы массива начиная со второго. На каждом повторении внешнего цикла текущий элемент key перемещается влево в отсортированную часть массива до тех пор, пока не встретит элемент, меньший или равный ему, или не дойдет до начала массива. Внутренний цикл while сдвигает все элементы отсортированной части массива, которые больше key, вправо, пока не найдет место для x или не дойдет до начала массива. Затем key помещается на свое место. Таким образом, циклы не могут быть бесконечны.

Из доказательства можно сделать вывод, что все циклы данного алгоритма корректны.

### **2.2.3 Определение ситуаций лучшего, среднего и худшего случая и функции роста времени работы алгоритма сортировки простыми вставками**

Таблица 1-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 | } |  |

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

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(n).

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 линейно с ростом размера входного массива.

Ёмкостная сложность алгоритма будет равна O(1).

## 

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

### **2.3.1 Реализация алгоритма сортировки простыми вставками на языке C++**

Реализуем данный алгоритм на языке C++(рис.2,3). Для реализации понадобятся такие библиотеки, как iostream, vector и random, chrono. Iostream — это заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++.Vector — это шаблон класса для контейнеров последовательности. Вектор хранит элементы заданного типа в линейном расположении и обеспечивает быстрый случайный доступ к любому элементу. Random - позволяет генерировать случайные числа в диапазоне. В данной программе задан диапазон от 1 до 10. Chrono позволяет реализовать такие концепции, как: интервалы времени, моменты времени, таймеры. Для подсчёта количество операций присваивания или сравнения введём переменную operations, которая представляет собой целое число в диапазоне от -2 147 483 648 до 2 147 483 648 и занимает 4 байта в памяти.

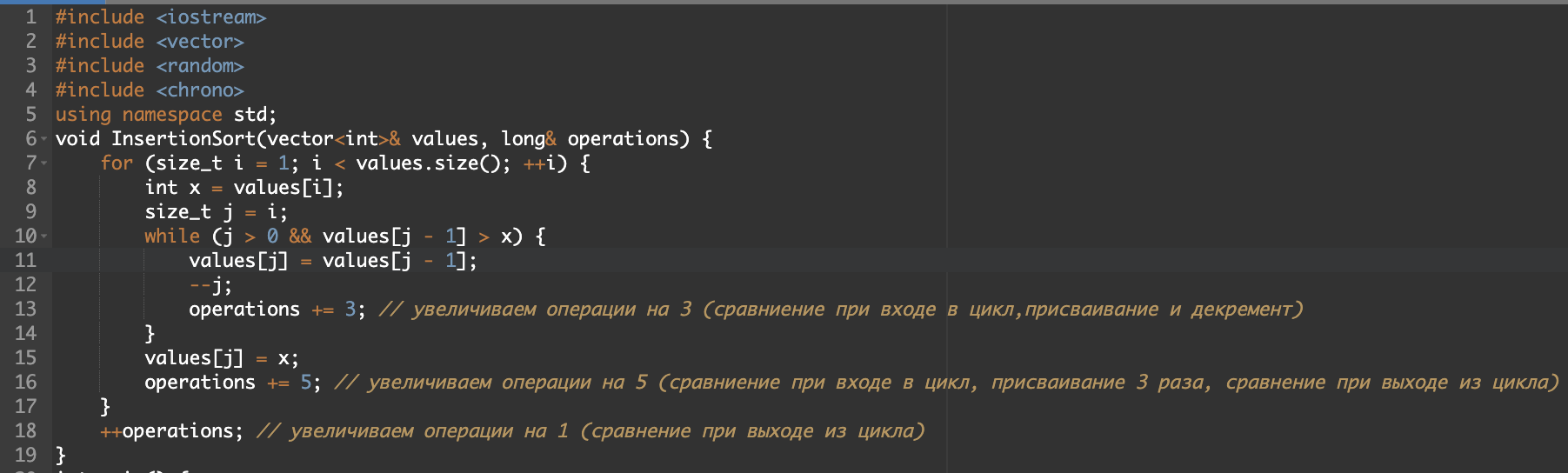


Рисунок 2 – Программа алгоритма сортировки простыми вставками



Рисунок 3 – Функция main для алгоритма сортировки простыми вставками

### **2.3.2 Тестирование**

Стоит задача протестировать программу с заданным размером массива n=10 (рис.4), 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 2. Воспользуемся структурой high\_resolution\_clock для подсчёта затраченного времени на сортировку. Для более точных результатов в программе будем рассматривать микросекунды, которые в дальнейшем, для заполнения таблицы, переведем в миллисекунды.

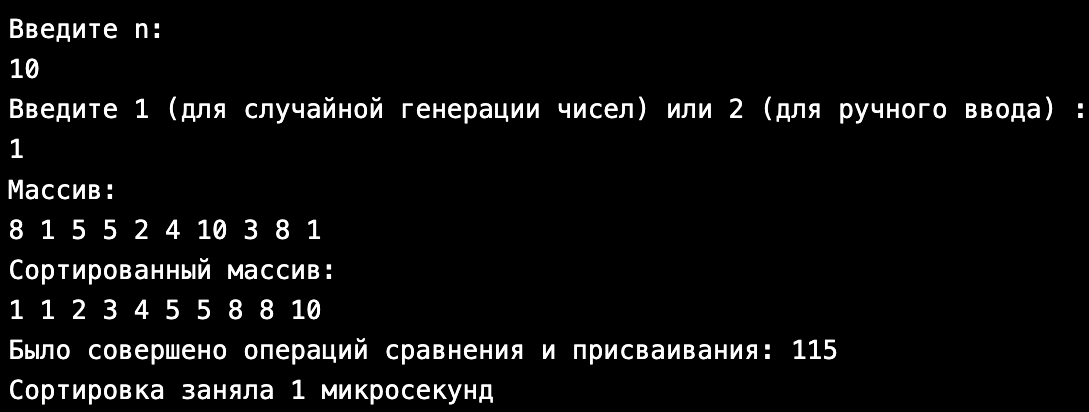


Рисунок 4 - Тестирование программы при n=10

Таблица 2. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.02 | - | 7402 |
| 1000 | 2.49 | - | 688876 |
| 10000 | 145.14 | - | 66876511 |
| 100000 | 14232.87 | - | 6749003338 |
| 1000000 | 1569885.74 | - | 665451729126 |

### **2.3.3 Построение графика**

На основе полученных данных, продемонстрированных в таблице 2 построим график функции роста Тп этого алгоритма от размера массива n(рис.5).

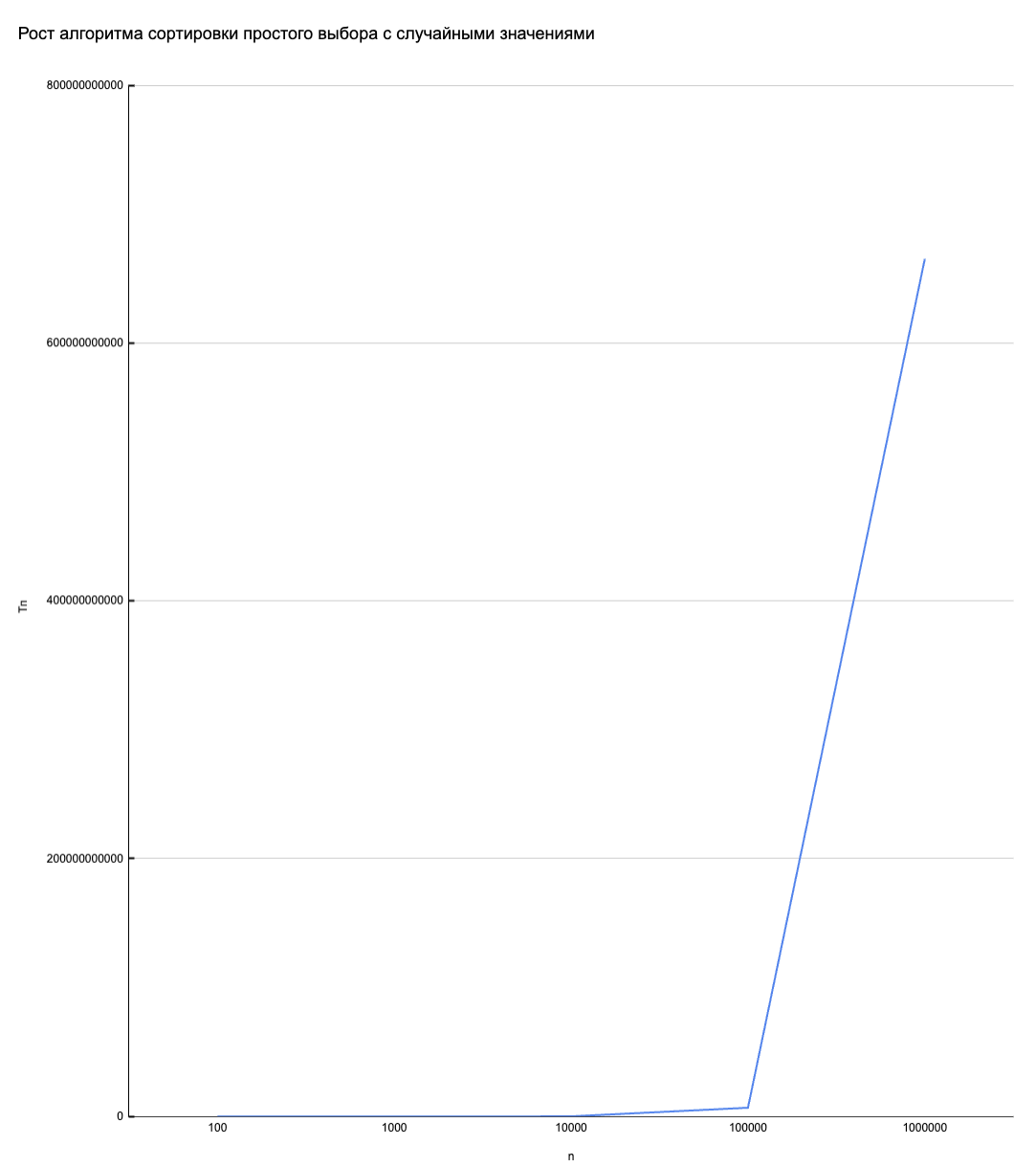


Рисунок 5 - График функции роста Тп этого алгоритма от размера массива n

## **2.4 Вывод по заданию №1**

На основе полученных данных тестирования и вычислительной сложности алгоритма, можно сделать вывод, что алгоритм сортировки простыми вставками имеет квадратичную вычислительную сложность, что означает, что время выполнения будет расти с увеличением размера массива квадратично.

# 3 ЗАДАНИЕ №2

## **3.1 Формулировка задачи**

Оценить вычислительную сложность алгоритма простой сортировки вставками 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**

Дополнительное тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов.

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

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго убывающем порядке. Изменим программу в функции main, чтобы значения элементов массива сортировались в убывающем порядке (рис.7) и продемонстрируем работу программы при n=10 (рис.8). Для сортировки значений добавим библиотеку algorithm (рис.6). Algorithm - стандартная библиотека C++, которая обрабатывает диапазоны итератора, которые обычно определяются их начальными или конечными позициями. Алгоритм сортировки простыми вставками не изменяется и соответствует продемонстрированному на рисунке 2.

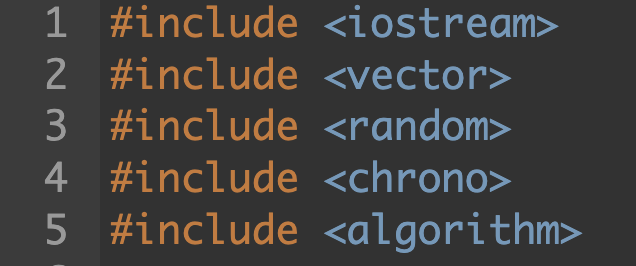


Рисунок 6 – Использованные библиотеки



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

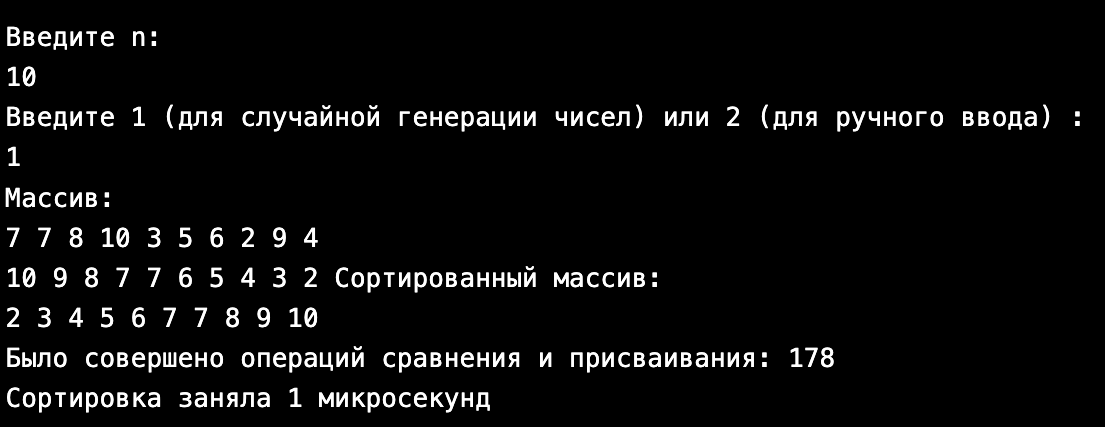


Рисунок 8 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

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 3.

Таблица 3. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.036 | - | 15346 |
| 1000 | 3.989 | - | 1503496 |
| 10000 | 911.055 | - | 150034996 |
| 100000 | 31680.913 | - | 15000349996 |
| 1000000 | 4364139.541 | - | 1500003499996 |

На основе полученных данных, продемонстрированных в таблице 3, построим график функции роста Тп алгоритма сортировки простыми вставками с отсортированными значениями по убыванию от размера массива n (рис.9).

### 

Рисунок 9 - График функции роста Тп алгоритма сортировки простыми вставками с отсортированными значениями по убыванию от размера массива n

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

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго возрастающем порядке. Изменим программу в функции main, чтобы значения элементов массива сортировались в возрастающем порядке (рис. 10) и продемонстрируем работу программы при n=10 (рис.11). Для сортировки значений добавим библиотеку algorithm (рис.6). Algorithm - стандартная библиотека C++, которая обрабатывает диапазоны итератора, которые обычно определяются их начальными или конечными позициями. Алгоритм сортировки простыми вставками не изменяется и соответствует продемонстрированному на рисунке 2.



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

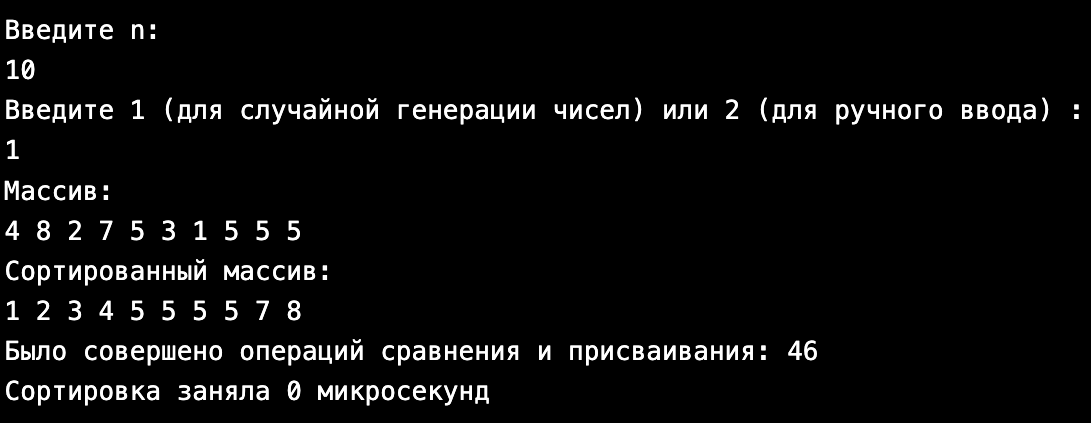


Рисунок 11 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Так как значения элементов массива идут в строго возрастающем порядке, то можно сделать вывод, что данная ситуация будет являться лучшим случаем, так как нет необходимости сдвигать элементы массива, а следовательно сложность алгоритма равна O(n). Следовательно, в лучшем случае алгоритм является линейным. Результаты тестирования будут приведены в таблице 4.

Таблица 4. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.001 | - | 496 |
| 1000 | 0.019 | - | 4996 |
| 10000 | 0.161 | - | 49996 |
| 100000 | 1.425 | - | 499996 |
| 1000000 | 15.358 | - | 4999996 |

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.12).

### 

Рисунок 12 - График функции роста Тп алгоритма сортировки простыми вставками с отсортированными значениями по возрастанию от размера массива n

## **3.3 Вывод по заданию №2**

Алгоритм сортировки вставкой зависим от исходной упорядоченности массива. Время выполнения алгоритма сортировки простой вставкой зависит от первоначальной упорядоченности массива. Если значения элементов массива строго возрастают, алгоритму потребуется меньше действий, так как нет необходимости сдвигать элементы.

Если значения элементов массива строго убывают, алгоритму сортировки простой вставкой потребуется большее количество шагов, так как для каждого элемента потребуется больше сдвигов, чтобы найти подходящее место для вставки.

Таким образом, время выполнения алгоритма сортировки простой вставкой непосредственно зависит от начальной степени упорядоченности массива.

# 4 ЗАДАНИЕ №3

## **4.1 Формулировка задания**

Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию алгоритма простого выбора.

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 Математическая модель решения алгоритма**

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

На первом шаге неупорядоченная часть – весь массив длиной n. В неупорядоченной части выбирается элемент с минимальным значением, который обменивается местами с первым элементом этой части массива и исключается из дальнейшей сортировки (включается в уже упорядоченную часть массива).

Затем выбирается минимальный элемент среди оставшихся n-1 элементов исходного массива, обменивается местами начальным элементом неупорядоченной части и также исключается из сортировки.

Процесс продолжается, пока в неотсортированной части массива не окажется единственный элемент, который считается упорядоченным.

Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.13).

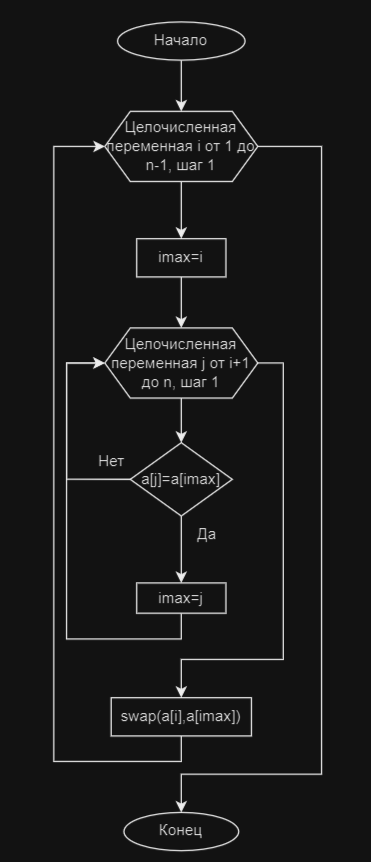


Рисунок 13 – Блок-схема алгоритма сортировки простым выбором

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

Инвариант для внешнего цикла: значение переменной i всегда меньше n-1.

Инвариант для внутреннего цикла: значение переменной j всегда больше i и меньше n.

Докажем конечность циклов. Внешний цикл for проходит через все элементы массива начиная со второго, так как первые 0 элементов в массиве уже отсортированы по определению. Внутренний цикл находит минимум среди неотсортированных элементов. Этот минимум меняется местами с текущим элементом, тем самым осуществляя сортировку. Этот процесс повторяется, пока не будут рассмотрены все элементы массива. Таким образом циклы алгоритма завершаются за конечное число повторений.

Из доказательства можно сделать вывод, что все циклы данного алгоритма корректны.

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

Таблица 5. 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 | } |  |

a. Лучший случай - массив уже отсортирован. В этом случае алгоритм будет иметь сложность 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).

Для данного метода сортировки, время исполнения постоянно увеличивается квадратично с ростом размера входного массива. Следовательно, можно использовать квадратичную функцию для описания функции роста данного сортировочного метода.

Ёмкостная сложность алгоритма будет равна O(1).

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

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

Реализуем данный алгоритм на языке C++(рис.14,15). Для реализации понадобятся 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 позволяет реализовать такие концепции, как: интервалы времени, моменты времени, таймеры.Для подсчёта количество операций присваивания или сравнения введём переменную operations, которая представляет собой целое число в диапазоне от -2 147 483 648 до 2 147 483 648 и занимает 4 байта в памяти.

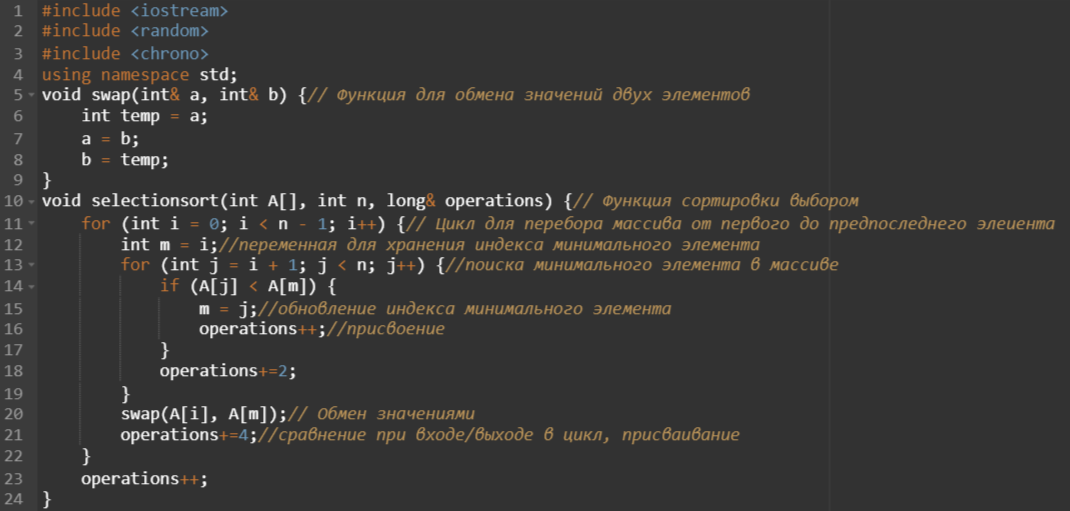


Рисунок 14 – Программа алгоритма сортировки простым выбором



Рисунок 15 – Функция main для алгоритма сортировки простым выбором

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

Стоит задача протестировать программу с заданным размером массива n=10 (рис.16), 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 6. Воспользуемся структурой high\_resolution\_clock для подсчёта затраченного времени на сортировку. Для более точных результатов в программе будем рассматривать микросекунды, которые в дальнейшем, для заполнения таблицы, переведем в миллисекунды.

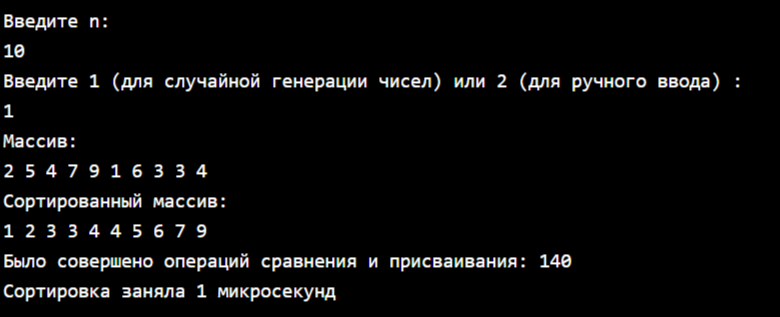


Рисунок 16 - Тестирование программы при n=10

Таблица 6. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.013 | - | 10471 |
| 1000 | 1.704 | - | 1004998 |
| 10000 | 394.468 | - | 100049455 |
| 100000 | 11673.021 | - | 10000493340 |
| 1000000 | 2065895.43 | - | 1000004955430 |

### **4.3.3 Построение графика**

На основе полученных данных, продемонстрированных в таблице 6 построим график функции роста Тп этого алгоритма от размера массива n(рис.17).

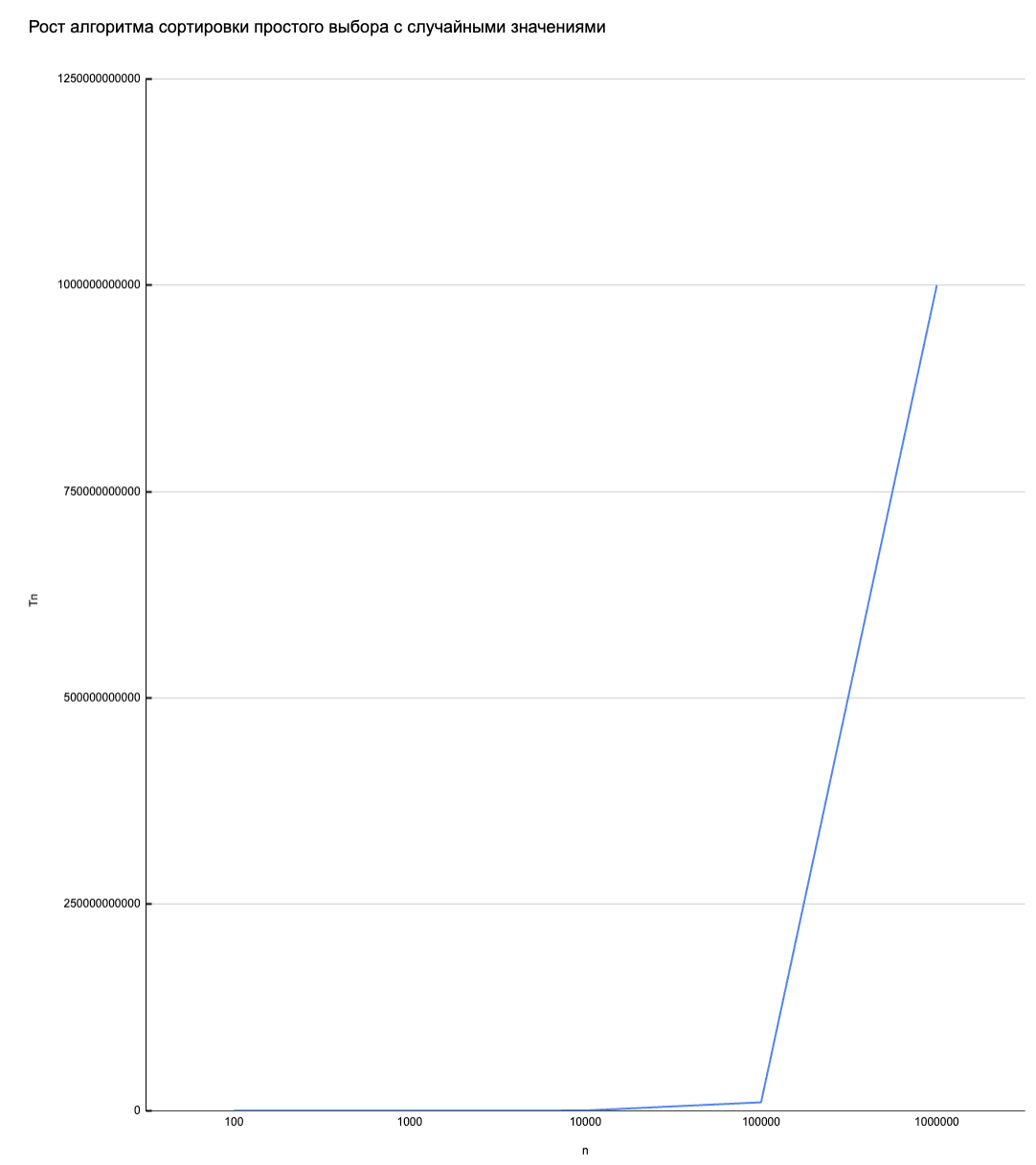


Рисунок 17 - График функции роста Тп этого алгоритма от размера массива n

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

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго убывающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по убыванию (рис.18). В функцию main добавим вызов функции сортировки по убыванию (рис.19) и продемонстрируем работу программы при n=10 (рис.20). Алгоритм сортировки простым выбором не изменяется и соответствует продемонстрированному на рисунке 14.

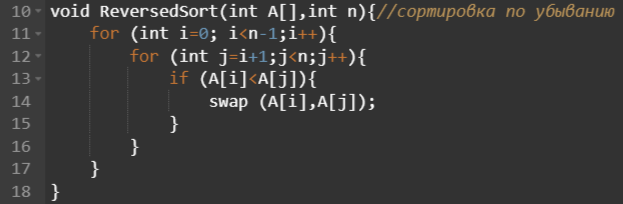


Рисунок 18 – Функция сортировки по убыванию



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

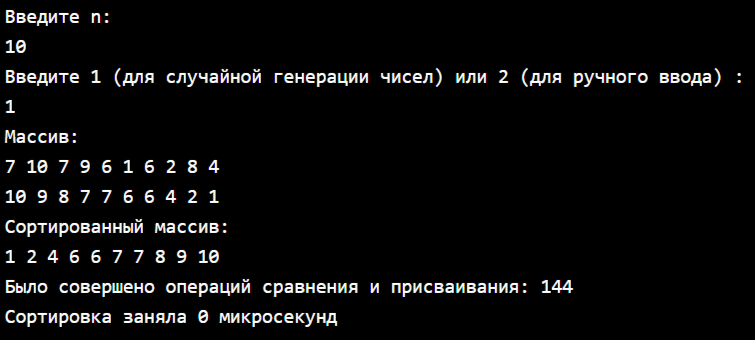


Рисунок 20 – Результаты тестирования программы при n=10 и с отсортированными значениями по убыванию

Так как значения идут в строго убывающем порядке, то можно сделать вывод, что данная ситуация будет являться худшим случаем, а следовательно сложность алгоритма равна O(n2). Следовательно, в худшем случае алгоритм имеет квадратичную сложность алгоритма. Результаты тестирования будут приведены в таблице 7.

Таблица 7. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.026 | - | 10546 |
| 1000 | 1.410 | - | 1005541 |
| 10000 | 122.213 | - | 100056081 |
| 100000 | 11557.2 | - | 10000551513 |
| 1000000 | 1778306.364 | - | 1000005509835 |

На основе полученных данных, продемонстрированных в таблице 7, построим график функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по убыванию от размера массива n (рис.21).

### 

Рисунок 21 - График функции роста Тп алгоритма сортировки простым выбором с отсортированными значениями по убыванию от размера массива n

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

Будет проведено тестирование программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, которые отсортированы в строго возрастающем порядке. Добавим в программу функцию, в которой проведем сортировку массива по возрастанию(рис.22). В функцию main добавим вызов функции сортировки по возрастанию(рис.23) и продемонстрируем работу программы при n=10 (рис.24). Алгоритм сортировки простым выбором не изменяется и соответствует продемонстрированному на рисунке 14.

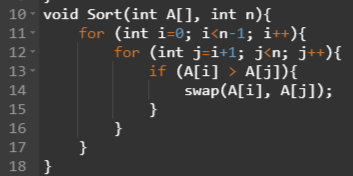


Рисунок 22 - Функция сортировки массива по возрастанию

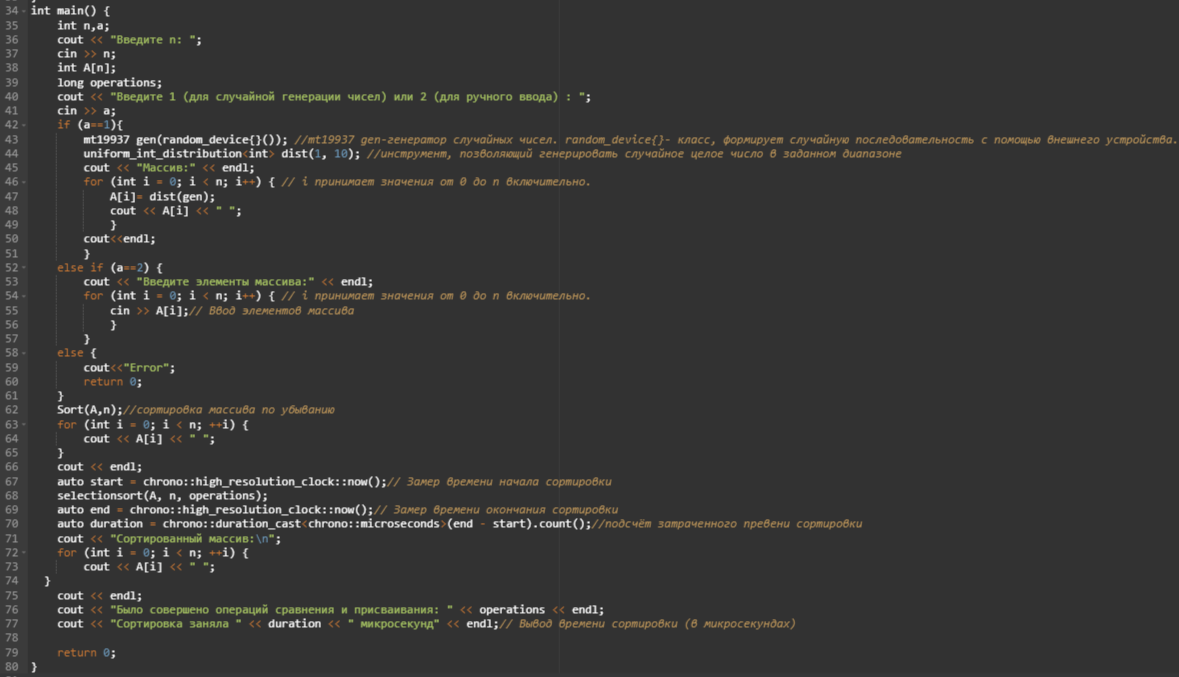


Рисунок 23 – Тестирование программы при n=10 и с отсортированными значениями по возрастанию

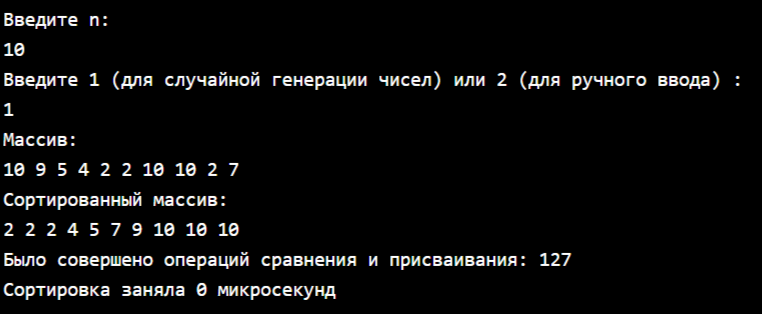


Рисунок 24 – Результаты тестирования программы при n=10 и с отсортированными значениями по возрастанию

Так как значения элементов массива идут в строго возрастающем порядке, то можно сделать вывод, что данная ситуация будет являться лучшим случаем, так как нет необходимости совершать обмен значениями, а следовательно сложность алгоритма равна O(n2). Следовательно, в лучшем случае алгоритм имеет квадратичную сложность алгоритма. Результаты тестирования будут приведены в таблице 8.

Таблица 8. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n), мс** | **Тт=C+M** | **Тп=Cп+Mп** |
| 100 | 0.025 | - | 10297 |
| 1000 | 1.195 | - | 1002997 |
| 10000 | 115.807 | - | 100029997 |
| 100000 | 11509.095 | - | 10000299997 |
| 1000000 | 1730953.6 | - | 1000002999997 |

На основе полученных данных, продемонстрированных в таблице 8, построим график функции роста Тп этого алгоритма от размера массива n с отсортированными значениями по возрастанию (рис.25).

### 

Рисунок 25 - 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 основании данных из таблицы 3 и графика алгоритма сортировки простыми вставками в худшем случае(рис.9), и таблицы 6 и графика алгоритма сортировки простым выбором в худшем случае(рис.21), мы создадим новый график для сравнения роста графиков(рис.26).

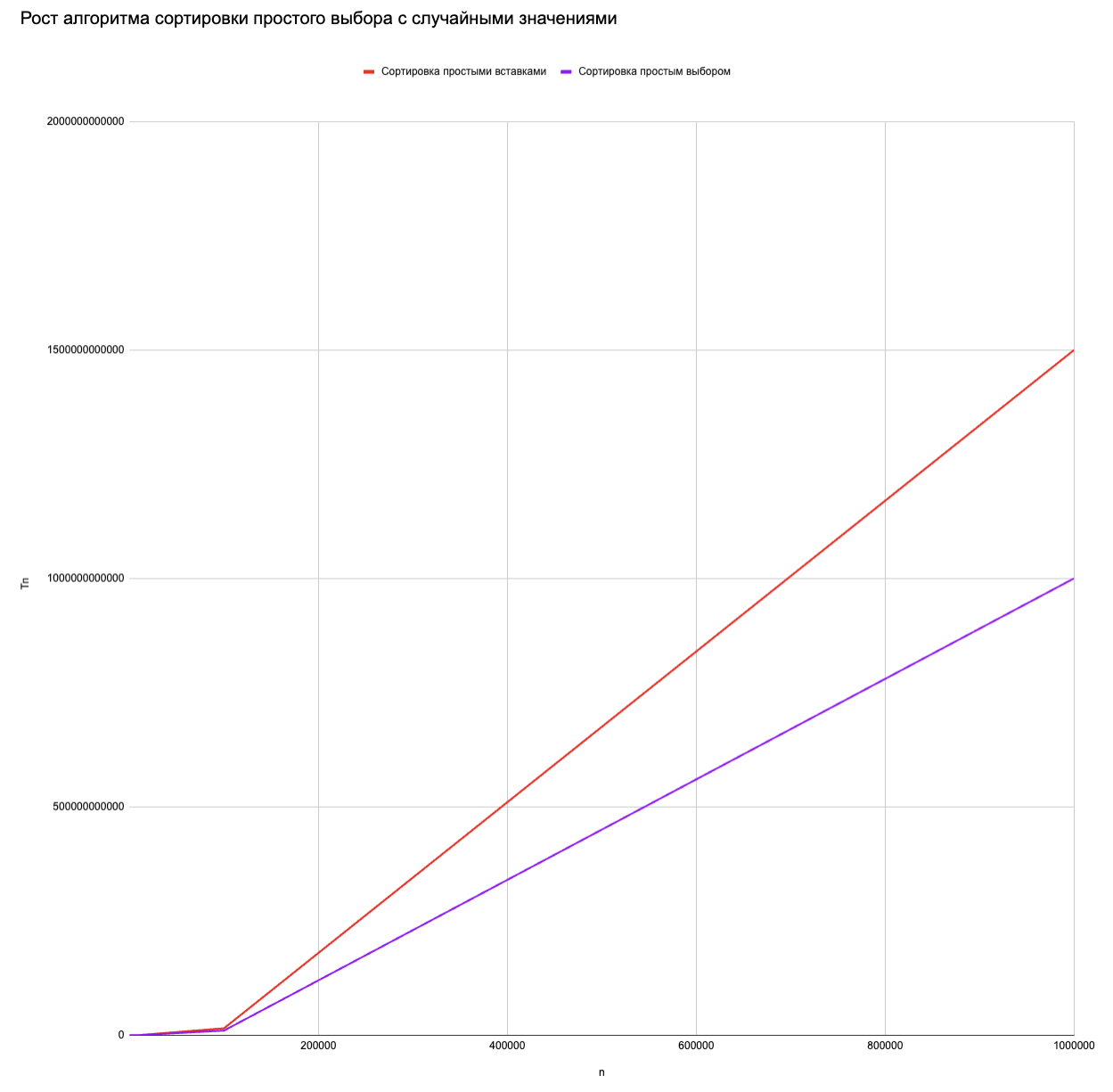


Рисунок 26 – График двух алгоритмов в худшем случае

На основе таблиц 3 и 7 и графика(рис.26), можно сделать вывод, что в худшем случае алгоритм сортировки простой вставкой менее эффективный, чем алгоритм сортировки простого выбора.

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

На основании данных из таблицы 4 и графика алгоритма сортировки простыми вставками в лучшем случае(рис.12), и таблицы 8 и графика алгоритма сортировки простым выбором в лучшем случае(рис.25), мы создадим новый график для сравнения роста графиков(рис.27).

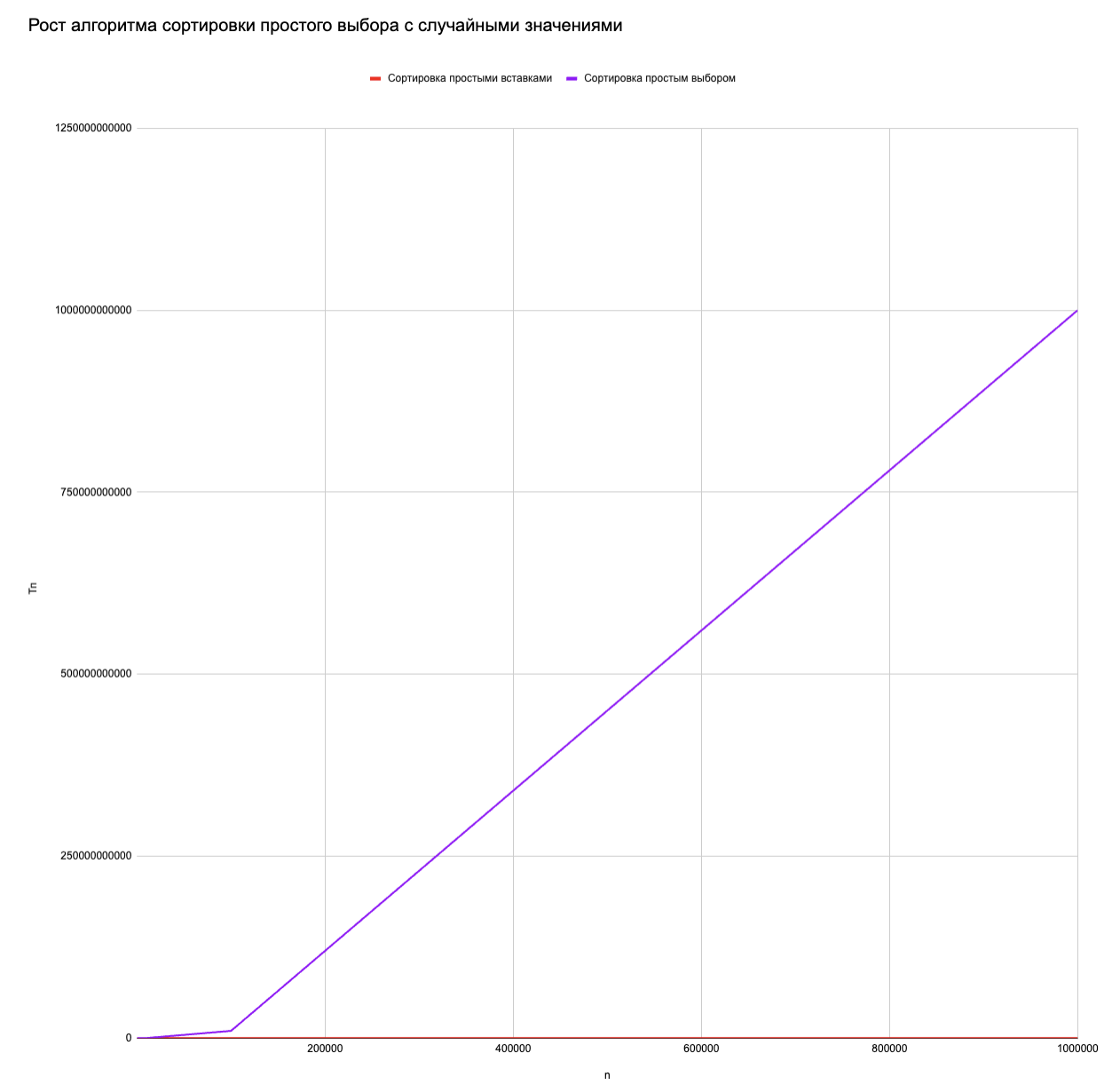
****

Рисунок 27 – График двух алгоритмов в лучшем случае

На основе таблиц 4 и 8 и графика(рис.27), можно сделать вывод, что в лучшем случае алгоритм сортировки простой вставкой более эффективный, чем алгоритм сортировки простого выбора.

## **4.8 Выводы по заданию №3**

Алгоритм сортировки простыми вставками имеет линейную вычислительную сложность в лучшем случае O(n), в среднем и в худшем случае O(n2). Этот алгоритм эффективен для небольших наборов данных, но может стать очень медленным на больших массивах.

Алгоритм сортировки простым выбором также имеет квадратичную вычислительную сложность в худшем случае O(n2). В лучшем случае алгоритм простого выбора также демонстрирует квадратичную вычислительную сложность, так как у данной сортировки нет возможности преобразоваться для лучшего случая. Этот алгоритм может быть очень медленным на больших массивах данных.

Таким образом, выбор алгоритма сортировки зависит от конкретной задачи и данных, с которыми он будет работать. В некоторых случаях сортировка простым выбором может быть более эффективной, а в других случаях - сортировка простой вставкой. Поэтому при выборе алгоритма необходимо учитывать размер и последовательность входных данных.

# 4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие сортировки называют простыми?

Простыми сортировками являются сортировка пузырьком, сортировка вставками и сортировка выбором.

2. Что означает понятие «внутренняя сортировка»?

Методы сортировки можно разделить на внутренние и внешние. Внутренняя сортировка – это упорядочение последовательности элементов, когда она целиком находится в оперативной памяти. Такие алгоритмы применяются к относительно небольшим по своему объёму последовательностям. Алгоритмам внутренней сортировки достаточно для решения задачи ресурса внутренней (оперативной) памяти.

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

Критическими (основными) называют операции, которые выполняются наиболее часто и время выполнения которых составляет основную часть общего времени выполнения алгоритма.

В ходе анализа эффективности алгоритма следует предварительно выявить эти критические операции (группы операций).

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

Критериями эффективности алгоритма являются скорость (время выполнения или, то же, время работы исполнителя – процессора ЭВМ) и расход памяти (внутренней в первую очередь) и/или других ресурсов.

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

1. Простой обмен:

* Вычислительная сложность: O(n2)
* Емкостная сложность: O(1)

2. Простая вставка:

* Вычислительная сложность: O(n2) в худшем случае, O(n) в лучшем случае
* Емкостная сложность: O(1)

3. Простой выбор:

* Вычислительная сложность: O(n2)
* Емкостная сложность: O(1)

6. Какую роль в сортировке обменом играет условие Айверсона?

Условие Айверсона: если в очередном проходе сортировки при сравнении элементов не было сделано ни одной перестановки, то множество считается упорядоченным.

Это условие позволяет оптимизировать работу алгоритма для случаев, когда массив уже отсортирован или требует минимального количества обменов. При выполнении условия Айверсона - алгоритм прекращает дальнейшую сортировку, так как массив уже отсортирован.

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

Исходный массив сортировался алгоритмом сортировки простым выбором.

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

Лучший случай - массив уже отсортирован. В этом случае алгоритм будет иметь сложность 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).

# 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 вставками и выбором;

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

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

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

- Произведено сравнение алгоритмов простой сортировки вставками и выбором 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).