



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ \_\_\_\_\_ «Информатика и системы управления»

КАФЕДРА \_\_\_\_\_ «Программное обеспечение ЭВМ и информационные технологии»

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 по курсу «Анализ алгоритмов»

«Трудоёмкость алгоритмов сортировки»

Студент \_\_\_\_\_ Маслова Марина Дмитриевна

Группа \_\_\_\_\_ ИУ7-53Б

Оценка (баллы) \_\_\_\_\_

Преподаватель \_\_\_\_\_ Волкова Лилия Леонидовна

2021 г.

# Содержание

<b>Введение</b>	<b>4</b>
<b>1 Аналитическая часть</b>	<b>5</b>
1.1 Алгоритм сортировки вставками . . . . .	5
1.2 Алгоритм сортировки перемешиванием . . . . .	5
1.3 Алгоритм сортировки выбором . . . . .	5
1.4 Вывод . . . . .	6
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Разработка алгоритмов . . . . .	7
2.2 Оценка трудоемкости алгоритмов . . . . .	11
2.2.1 Модель вычислений . . . . .	11
2.2.2 Алгоритм сортировки вставками . . . . .	11
2.2.3 Алгоритм сортировки перемешиванием . . . . .	12
2.2.4 Алгоритм сортировки выбором . . . . .	13
2.3 Структура разрабатываемого ПО . . . . .	13
2.4 Классы эквивалентности при тестировании . . . . .	14
2.5 Вывод . . . . .	14
<b>3 Технологическая часть</b>	<b>15</b>
3.1 Требования к программному обеспечению . . . . .	15
3.2 Средства реализации . . . . .	15
3.3 Листинги кода . . . . .	16
3.4 Описание тестирования . . . . .	17
3.5 Вывод . . . . .	17
<b>4 Исследовательская часть</b>	<b>18</b>
4.1 Технические характеристики . . . . .	18
4.2 Примеры работы программы . . . . .	18
4.3 Результаты тестирования . . . . .	18
4.4 Постановка эксперимента по замеру времени . . . . .	19
4.5 Результаты эксперимента . . . . .	19
4.6 Вывод . . . . .	23

<b>Заключение</b>	<b>24</b>
<b>Список литературы</b>	<b>25</b>

# Введение

Сортировка – процесс перестановки объектов данного множества в определенном порядке. Сортировка служит для последующего облегчения поиска элементов в отсортированном множестве. При этом сортировка является примером огромного разнообразия алгоритмов, выполняющих одну и ту же задачу. Несмотря на большое количество алгоритмов сортировки, любой из них можно разбить на три основные части:

- сравнение, определяющее порядок элементов в паре;
- перестановка, меняющая элементы в паре местами;
- сам сортирующий алгоритм, который выполняет два предыдущих шага до полного упорядочивания.

Ввиду большого количества алгоритмов сортировки одни из них имеют преимущества над другими, что приводит к необходимости их сравнительного анализа [1].

**Целью данной работы** является получение навыков анализа алгоритмов сортировок.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- изучить три алгоритма сортировки: вставками, перемешиванием, выбором;
- разработать каждый из алгоритмов;
- дать теоретическую оценку трудоемкости алгоритмов сортировки;
- реализовать каждый алгоритм;
- провести тестирование реализованных алгоритмов;
- провести сравнительный анализ алгоритмов по процессорному времени работы реализации;

# 1 Аналитическая часть

В данном разделе представлено теоретическое описание алгоритмов сортировки вставками, перемешиванием и выбором.

## 1.1 Алгоритм сортировки вставками

В алгоритме сортировки вставками сортируемая последовательность условно делится на входную неотсортированную часть и выходную отсортированную часть. На каждом шаге из неотсортированной части выбирается элемент и помещается на нужную позицию в уже отсортированной части. В начале алгоритма считается, что первый элемент последовательности является отсортированной частью, поэтому вставка элементов в отсортированную часть начинается со второго элемента [2].

## 1.2 Алгоритм сортировки перемешиванием

Алгоритм сортировки перемешиванием является модификацией алгоритма сортировки пузырьком. В отличие от сортировки пузырьком, где происходит обход последовательности только в одном направлении, в алгоритме сортировки перемешиванием после достижения одной из границ рабочей части последовательности (то есть той части, которая еще не отсортирована и в которой происходит смена элементов) меняет направление движения. При этом при движении в одном направлении алгоритм перемещает к границе рабочей области максимальный элемент, а в другом направлении – минимальный элемент. Границы рабочей части последовательности устанавливаются в месте последнего обмена [2].

## 1.3 Алгоритм сортировки выбором

Алгоритм сортировки выбором в текущей последовательности находит минимальный/максимальный элемент, производит обмен этого элемента со значением первой неотсортированной позиции и сортирует оставшуюся часть последовательности, исключив из рассмотрения уже отсортированные элементы [2].

## 1.4 Вывод

В данном разделе были рассмотрены алгоритмы сортировки вставками, перемешиванием и выбором. Из представленных описаний можно предъявить ряд требований к разрабатываемому программному обеспечению:

- на вход подается последовательность, которую необходимо отсортировать;
- на выходе должна выдаваться отсортированная последовательность.

## **2 Конструкторская часть**

В данном разделе разрабатываются алгоритмы сортировки, структура программы и способы её тестирования, также проводится оценка трудоемкости каждого из алгоритмов.

### **2.1 Разработка алгоритмов**

На рисунке 2.1 представлена схема алгоритма сортировки вставками, на рисунке 2.2 — схема алгоритма сортировки перемешиванием, на рисунке 2.3 — схема алгоритма сортировки выбором.

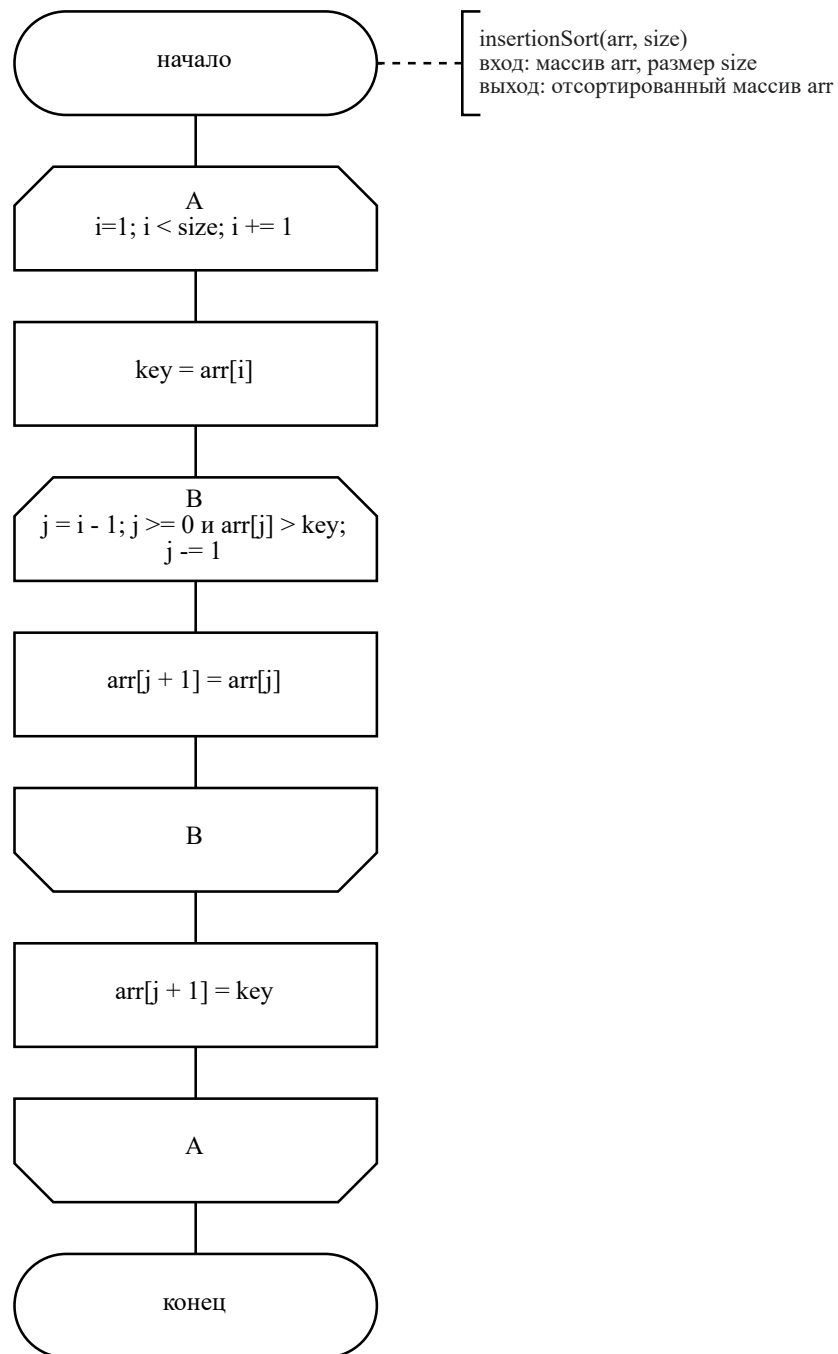


Рисунок 2.1 – Схема алгоритма сортировки вставками



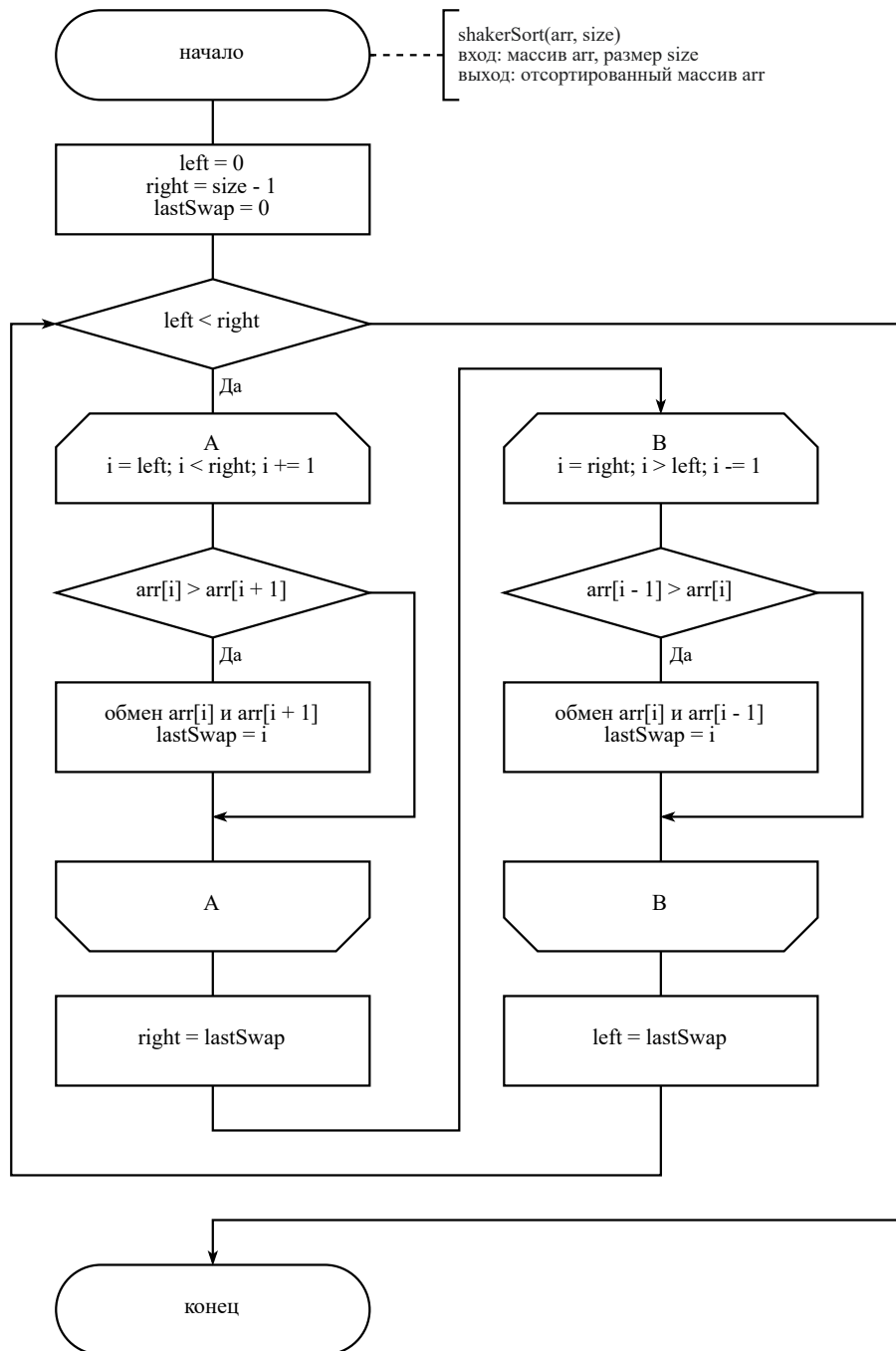


Рисунок 2.2 – Схема алгоритма сортировки перемешиванием

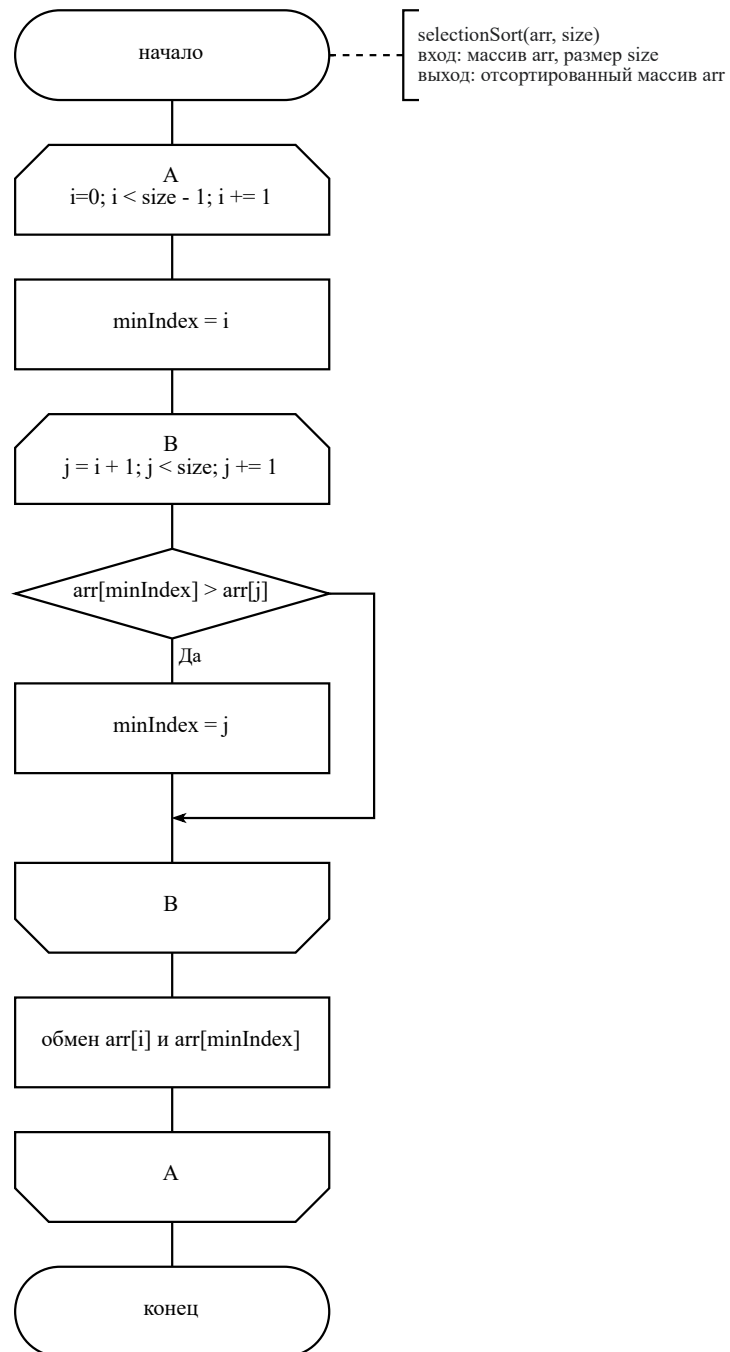


Рисунок 2.3 – Схема алгоритма сортировки выбором

## 2.2 Оценка трудоемкости алгоритмов

В данном подразделе производится оценка трудоемкости каждого из алгоритмов.

### 2.2.1 Модель вычислений

Введем модель вычислений для оценки трудоемкости алгоритмов:

- операции из списка 2.1 имеют трудоемкость 2;

$$*, /, //, \%, *=, /=, // = \quad (2.1)$$

- операции из списка 2.2 имеют трудоемкость 1;

$$\begin{aligned} &=, +, -, +=, -=, <, >, ==, !=, \\ &>=, <=, [], <<, >>, ++, --, and, or \end{aligned} \quad (2.2)$$

- трудоемкость оператора выбора `if условие then A else B` рассчитывается по формуле 2.3:

$$f_{if} = f + \begin{cases} f_A, & \text{если условие выполняется;} \\ f_B, & \text{иначе} \end{cases} \quad (2.3)$$

- трудоемкость оператора цикла рассчитывается по формуле 2.4:

$$f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инкремента} + f_{сравнения}) \quad (2.4)$$

- трудоемкость вызова функции равна 0.

### 2.2.2 Алгоритм сортировки вставками

В лучшем случае алгоритму подается на вход упорядоченная последовательность. При таких входных данных цикл  $A$  обрабатывает  $N - 1$  раз, где  $N = size$ , а внутренний цикл  $B$  на каждой итерации осуществляет только инициализацию и проверку условия, которое оказывается ложным, из-за чего тело не выполняется. Таким образом, в лучшем случае трудоемкость алгорит-

ма сортировки вставками равна (формула 2.5):

$$f^{\wedge} = 2 + (N - 1)(2 + 2 + 4 + 3) = 13N - 11 = O(N) \quad (2.5)$$

В худшем случае алгоритму подается на вход последовательность, упорядоченная в обратном порядке. При таких входных данных цикл  $B$  отрабатывает в среднем  $\frac{N+2}{2}$ . Таким образом, в худшем случае трудоемкость алгоритма сортировки вставками равна (формула 2.6):

$$f^{\vee} = 2 + (N - 1)(2 + 2 + 6 + \frac{N + 2}{2}(5 + 4) + 3) = \frac{9}{2}N^2 + \frac{35}{2}N - 20 = O(N^2) \quad (2.6)$$

### 2.2.3 Алгоритм сортировки перемешиванием

В лучшем случае алгоритму подается на вход упорядоченная последовательность. При таких входных данных первый внутренний цикл  $A$  осуществляет полный проход по последовательности в одну сторону и не находит обменов соседних элементов, поэтому на первой же итерации внешнего цикла правая граница становится равной левой границе, из-за чего второй внутренний цикл  $B$  не отрабатывает, произведя только инициализацию и одну проверку условия, а внешний цикл завершает свою работу после первой итерации. Таким образом, в лучшем случае трудоемкость алгоритма сортировки перемешиванием равна (формула 2.7):

$$f^{\wedge} = 4 + 1 + 2 + (N - 1)(2 + 4) + 1 + 2 + 1 + 1 = 6N + 6 = O(N) \quad (2.7)$$

В худшем случае алгоритму подается на вход упорядоченная в обратном порядке последовательность. Внешний цикл отрабатывает  $\frac{N}{2}$  раз, так как за одну итерацию на свое место ставятся 2 элемента. Количество итераций каждого цикла зависит от четности размера подаваемой последовательности, если  $N$  — четно, то цикл  $A$  отработает  $\frac{N-1+1}{2}$  итераций, а цикл  $B$  —  $\frac{N-2+1}{2}$ , если  $N$  — нечетно, то цикл  $A$  отработает  $\frac{N-1+2}{2}$  итераций, а цикл  $B$  —  $\frac{N-2}{2}$ , однако и в том, и другом случае в общем два цикла отработают  $\frac{2N-1}{2}$  итераций, а так как сложность тел обоих циклов одинаковая при вычислениях можно сразу пользоваться общим числом итераций. Таким образом, в худшем случае

трудоемкость алгоритма сортировки перемешиванием равна (формула 2.8):

$$\begin{aligned} f^{\vee} &= 4 + \frac{N}{2}(1 + 2 + 1 + 2 + 1 + \frac{2N-1}{2}(2 + 4 + 9 + 1)) = \\ &= 4 + \frac{N}{2}(16N - 1) = 8N^2 - \frac{N}{2} + 4 = O(N^2) \quad (2.8) \end{aligned}$$

### 2.2.4 Алгоритм сортировки выбором

В любом случае в алгоритме сортировки выбором внешний цикл  $A$  выполняет  $N - 1$  итераций, а внутренний цикл в среднем  $\frac{N}{2}$  итерации.

В лучшем случае на вход алгоритму подается упорядоченная последовательность. При таких входных данных никогда не выполняется блок по ветке `then` условия. Таким образом, в лучшем случае трудоемкость алгоритма сортировки выбором равна (формула 2.9):

$$\begin{aligned} f^{\wedge} &= 3 + (N - 1)(3 + 1 + 3 + \frac{N}{2}(2 + 3) + 7) = \\ &= 3 + (N - 1)(14 + \frac{5}{2}N) = \frac{5}{2}N^2 + \frac{23}{2}N - 11 = O(N^2) \quad (2.9) \end{aligned}$$

В худшем случае на вход алгоритму подается упорядоченная в обратном порядке последовательность. При таких входных данных всегда выполняется блок по ветке `then` условия. Таким образом, в худшем случае трудоемкость алгоритма сортировки выбором равна (формула 2.10):

$$\begin{aligned} f^{\wedge} &= 3 + (N - 1)(3 + 1 + 3 + \frac{N}{2}(2 + 3 + 1) + 7) = \\ &= 3 + (N - 1)(14 + 3N) = 3N^2 + 11N - 11 = O(N^2) \quad (2.10) \end{aligned}$$

## 2.3 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

## 2.4 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- пустой массив;
- упорядоченный массив четной длины;
- упорядоченный массив нечетной длины;
- упорядоченный в обратном порядке массив четной длины;
- упорядоченный в обратном порядке массив нечетной длины;
- случайный массив;
- массив из одного элемента.

## 2.5 Вывод

В данном разделе были разработаны алгоритмы сортировки вставками, перемешиванием и выбором, также была произведена оценка трудоемкостей алгоритмов. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

## 3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

### 3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- выбор режима работы: для единичного эксперимента и для массовых экспериментов;
- в режиме единичного эксперимента ввод последовательности для сортировки и вывод результатов работы каждого из алгоритмов сортировки (отсортированной последовательности);
- в режиме массовых экспериментов измерение времени работы каждого из алгоритмов сортировки при различных длинах сортируемой последовательности и различных её первоначальных состояниях: отсортированная, отсортированная в обратном порядке, случайная последовательности.

### 3.2 Средства реализации

Для реализации данной лабораторной работы выбран интерпретируемый язык программирования высокого уровня Python[3], так как он позволяет реализовывать сложные задачи за кратчайшие сроки за счет простоты синтаксиса и наличия большого количества подключаемых библиотек.

В качестве среды разработки выбран текстовый редактор Vim[4] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции `process_time_ns` из библиотеки `time`[5].

### 3.3 Листинги кода

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

- алгоритм сортировки вставками (листинг 3.1);
- алгоритм сортировки перемешиванием (листинг 3.2);
- алгоритм сортировки выбором (листинги 3.3).

Листинг 3.1 – Реализация алгоритма сортировки вставками

```
1 def insertionSort(arr, size):
2     for i in range(1, size):
3         key = arr[i]
4
5         j = i - 1
6         while j >= 0 and arr[j] > key:
7             arr[j + 1] = arr[j]
8             j -= 1
9
10        arr[j + 1] = key
11
12    return arr
```

Листинг 3.2 – Реализация алгоритма сортировки перемешиванием

```
1 def shakerSort(arr, size):
2     left = 0
3     right = size - 1
4     lastSwap = 0
5
6     while left < right:
7         for i in range(left, right):
8             if arr[i] > arr[i + 1]:
9                 arr[i], arr[i + 1] = arr[i + 1], arr[i]
10                lastSwap = i
11
12        right = lastSwap
13
14        for i in range(right, left, -1):
15            if arr[i - 1] > arr[i]:
16                arr[i], arr[i - 1] = arr[i - 1], arr[i]
17                lastSwap = i
18
19        left = lastSwap
20
21    return arr
```



### Листинг 3.3 – Реализация алгоритма сортировки выбором

```
1 def selectionSort(arr, size):
2     for i in range(size - 1):
3         minIndex = i
4
5         for j in range(i + 1, size):
6             if arr[minIndex] > arr[j]:
7                 minIndex = j
8
9         arr[minIndex], arr[i] = arr[i], arr[minIndex]
10
11     return arr
```

## 3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировки.

Таблица 3.1 – Функциональные тесты

Последовательность	Ожидаемый результат
[ ]	[ ]
[1, 2, 3, 4]	[1, 2, 3, 4]
[-2, -1, 0, 3, 4]	[-2, -1, 0, 3, 4]
[100, 50, -20, -99]	[-99, -20, 50, 100]
[13, 7, 1, -1, -17]	[-17, -1, 1, 7, 13]
[4, 2, -44, 76, 0, 11]	[-44, 0, 2, 4, 11, 76]
[113]	[133]

## 3.5 Вывод

В данном разделе были реализованы алгоритмы сортировки вставками, перемешиванием, выбором. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

## 4 Исследовательская часть

### 4.1 Технические характеристики

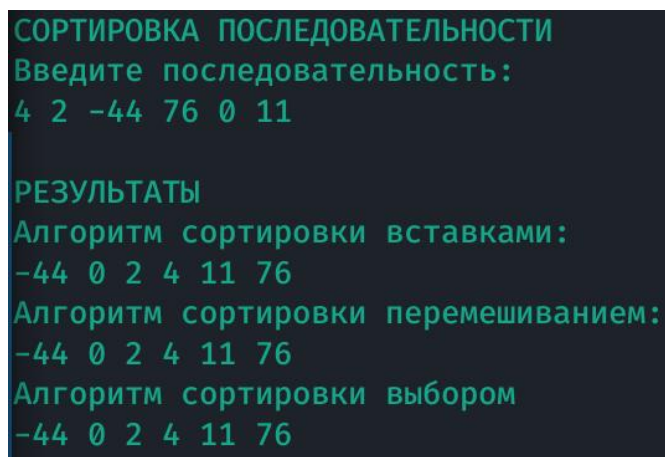
Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Manjaro [6] Linux x86\_64.
- Память: 8 GiB.
- Процессор: Intel® Core™ i5-8265U[7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

### 4.2 Примеры работы программы

На рисунке 4.1 представлены результаты работы программы на случайной последовательности.



```
СОРТИРОВКА ПОСЛЕДОВАТЕЛЬНОСТИ
Введите последовательность:
4 2 -44 76 0 11

РЕЗУЛЬТАТЫ
Алгоритм сортировки вставками:
-44 0 2 4 11 76
Алгоритм сортировки перемешиванием:
-44 0 2 4 11 76
Алгоритм сортировки выбором
-44 0 2 4 11 76
```

Рисунок 4.1 – Пример работы программы

### 4.3 Результаты тестирования

Программа была протестирована на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

## **4.4 Постановка эксперимента по замеру времени**

Для оценки времени работы реализаций алгоритмов сортировки был проведен эксперимент, в котором определялось влияние размеров последовательности на время работы каждого из алгоритмов. Тестирование проводилось на размерах последовательностей от 100 до 1000 с шагом 100. Так как от запуска к запуску процессорное время, затрачиваемое на выполнение алгоритма, менялось в определенном промежутке, необходимо было усреднить вычисляемые значения. Для этого каждый алгоритм запускался по 500 раз, и для полученных 500 значений определялось среднее арифметическое, которое заносилось в таблицу результатов.

Так как трудоемкость алгоритмов сортировки зависит от состояния входной последовательности, эксперимент проводился на трех видах последовательностей: отсортированных, отсортированных в обратном порядке и случайных.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

## **4.5 Результаты эксперимента**

В таблицах 4.1, 4.2, 4.3 представлены результаты измерения времени работы алгоритмов на отсортированных, отсортированных в обратном порядке и случайных последовательностях соответственно.

На основе табличных данных построены графики зависимости времени работы каждого алгоритма от размеров последовательностей. На рисунке 4.2 представлены графики зависимостей времени работы алгоритмов от длины отсортированных последовательностей. Для наглядности графики зависимостей для сортировки вставками и перемешиванием вынесены в отдельный рисунок 4.3. На рисунках 4.4 и 4.5 представлены графики зависимостей времени работы алгоритмов от длины отсортированных в обратном порядке и случайных последовательностей соответственно.

Таблица 4.1 – Время работы алгоритмов на отсортированных последовательностях

Размер	Вставками, нс	Перемешиванием, нс	Выбором, нс
100	15357	6521	212917
200	20364	11572	810167
300	31025	17277	1888263
400	42730	23613	3513342
500	53369	29545	5551291
600	65072	36129	8037746
700	77253	41814	10992958
800	87667	47893	14428982
900	98768	54206	18273969
1000	116285	63919	23223570

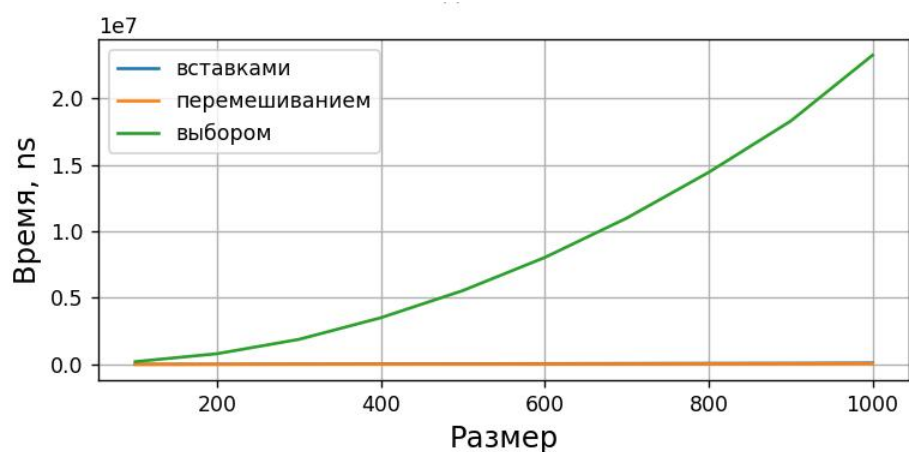


Рисунок 4.2 – Графики зависимости времени работы алгоритмов от размера отсортированных последовательностей

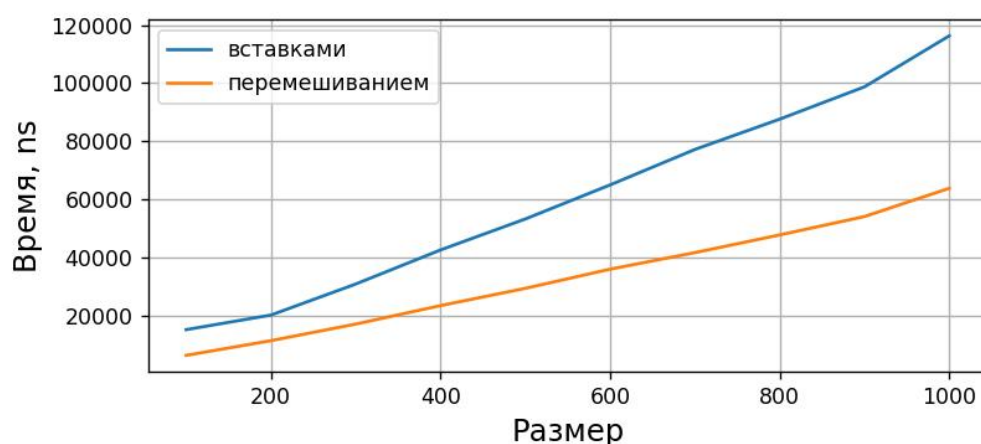


Рисунок 4.3 – Графики зависимости времени работы алгоритмов сортировки вставками и перемешиванием от размера отсортированных последовательностей

Таблица 4.2 – Время работы алгоритмов на отсортированных в обратном порядке последовательностях

Размер	Вставками, нс	Перемешиванием, нс	Выбором, нс
100	444248	650640	233515
200	1734685	2536449	895593
300	3853448	5797428	2076416
400	7027511	10861004	3862679
500	10996683	17645316	6122028
600	15990398	25911669	8839561
700	21948567	36308538	12330891
800	29169452	47516062	15895489
900	37706970	60883267	20528770
1000	46118476	75814417	26127407

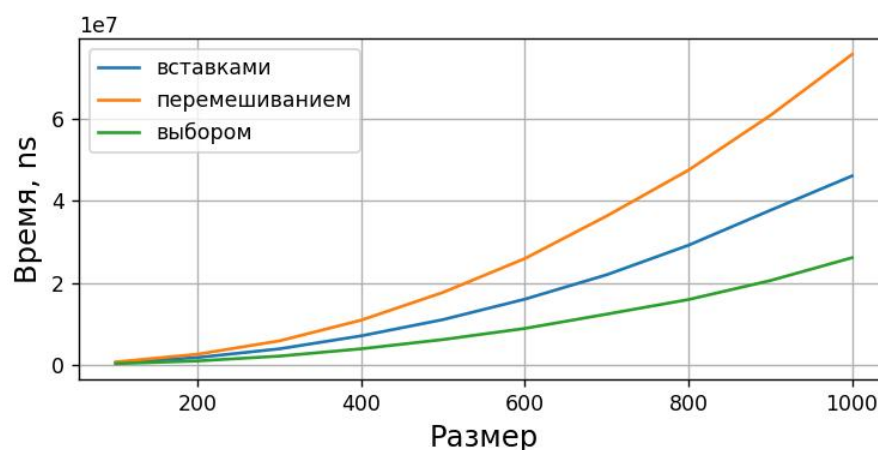


Рисунок 4.4 – Графики зависимости времени работы алгоритмов от размера отсортированных в обратном порядке последовательностей

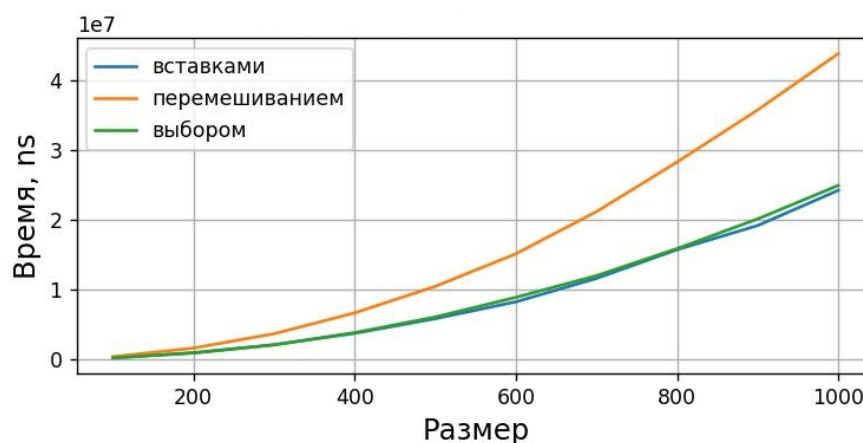


Рисунок 4.5 – Графики зависимости времени работы алгоритмов от размера случайных последовательностей

Таблица 4.3 – Время работы алгоритмов на случайных последовательностях

<b>Размер</b>	<b>Вставками, нс</b>	<b>Перемешиванием, нс</b>	<b>Выбором, нс</b>
100	252681	440573	271115
200	1013346	1669295	960011
300	2151410	3722280	2121445
400	3786263	6709814	3876523
500	5884831	10506119	6118259
600	8292222	15168688	8931808
700	11644589	21196548	12033555
800	15756234	28295827	15908374
900	19213556	35756587	20156346
1000	24247742	43814694	24947774

## 4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- на отсортированных последовательностях алгоритм сортировки выбором работает в 150-200 раз дольше двух других алгоритмов, так как имеет в лучшем случае сложность  $O(N^2)$ , в то время как два других алгоритма имеют в лучшем случае сложность  $O(N)$ ;
- на отсортированных последовательностях алгоритм сортировки перемешиванием работает в 1.8-2 раза быстрее алгоритма сортировки вставками, так как при отсутствии обмена элементов сразу же завершается, что также отражено в трудоемкости алгоритмов в лучшем случае (мультипликативная константа при  $N$  у трудоемкости алгоритма вставками в лучшем случае примерно в 2 раза больше, чем у трудоемкости алгоритма перемешиванием);
- в худшем случае, то есть при отсортированных в обратном порядке последовательностях, все алгоритмы имеют квадратичную сложность, что отражено на соответствующем графике (рисунок 4.4), однако трудоемкости имеют различные мультипликативные константы, что определяет их различное время работы, так алгоритм сортировки перемешиванием работает примерно в 1.7 раза дольше алгоритма сортировки вставками, и примерно в 2.7 раза дольше алгоритма сортировки выбором, что соответствует отношениям их мультипликативных констант при  $N^2$ ;
- в случае случайных последовательностей алгоритмы сортировки вставками и выбором работают приблизительно одинаковое время, в то время как алгоритм сортировки перемешиванием работает приблизительно в 1.8 раза дольше.

Таким образом, для сортировки последовательностей, которые в своем большинстве являются неупорядоченными, из данных трех алгоритмов, стоит предпочесть алгоритм сортировки вставками, так как на случайных последовательностях работает время соизмеримое с алгоритмом сортировки выбором и меньше по сравнению с алгоритмом сортировки перемешиванием и в то же время на упорядоченных в том или ином порядке последовательностях в среднем дает лучший результат по сравнению с алгоритмом сортировки выбором.

# Заключение

В ходе выполнения лабораторной работы:

- были описаны и разработаны алгоритмы сортировки вставками, перемешиванием и выбором;
- была произведена оценка трудоемкости каждого из алгоритмов;
- был реализован каждый из описанных алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов, которые были подтверждены теоретическими расчетами трудоемкости алгоритмов.



# Список литературы

- [1] Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989. с. 360.
- [2] Шагбазян Д. В., Штанюк А. А., Малкина Е. В. Алгоритмы сортировки. Анализ, реализация, применение. Нижний Новгород: Нижегородский госуниверситет, 2019. с. 42.
- [3] Welcome to Python [Электронный ресурс]. Режим доступа: <https://www.python.org> (дата обращения: 12.10.2021).
- [4] welcome home : vim online [Электронный ресурс]. Режим доступа: <https://www.vim.org/> (дата обращения: 12.10.2021).
- [5] time — Time access and conversions [Электронный ресурс]. Режим доступа: [https://docs.python.org/3/library/time.html#time.process\\_time\\_ns](https://docs.python.org/3/library/time.html#time.process_time_ns) (дата обращения: 04.10.2021).
- [6] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: <https://manjaro.org/> (дата обращения: 17.10.2021).
- [7] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html> (дата обращения: 17.10.2021).