



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"

Тема Алгоритмы сортировки

Студент Артюхин Н.П.

Группа ИУ7-51Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2022 г.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>5</b>
1.1 Гномья сортировка . . . . .	5
1.2 Поразрядная сортировка . . . . .	5
1.3 Сортировка выбором . . . . .	6
<b>2 Конструкторская часть</b>	<b>8</b>
2.1 Алгоритм гномьей сортировки . . . . .	8
2.2 Алгоритм поразрядной сортировки . . . . .	10
2.3 Алгоритм сортировки выбором . . . . .	13
2.4 Оценка трудоемкости алгоритмов сортировки . . . . .	15
2.4.1 Алгоритм гномьей сортировки . . . . .	15
2.4.2 Алгоритм поразрядной сортировки . . . . .	16
2.4.3 Алгоритм сортировки выбором . . . . .	18
<b>3 Технологическая часть</b>	<b>20</b>
3.1 Требования к программному обеспечению . . . . .	20
3.2 Выбор средств реализации . . . . .	20
3.3 Реализация алгоритмов . . . . .	20
3.4 Тестирование . . . . .	23
<b>4 Исследовательская часть</b>	<b>24</b>
4.1 Пример работы программного обеспечения . . . . .	24
4.2 Технические характеристики . . . . .	24
4.3 Время выполнения реализаций алгоритмов . . . . .	25
<b>Заключение</b>	<b>30</b>
<b>Список использованной литературы</b>	<b>31</b>

# Введение

**Цель лабораторной работы** - получение навыков оценки трудоемкости алгоритмов на материале трех алгоритмов сортировки: гномья, поразрядная и выбором.

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов определенным образом в заданной последовательности. Одной из главных целей сортировки является упрощение задачи поиска элемента в отсортированной последовательности.

Сортировка является одним из важнейших классов алгоритмов обработки данных и осуществляется большим количеством способов [1].

Алгоритмы сортировки имеют большое практическое применение и часто встречаются там, где нужно обрабатывать и хранить большие объемы информации. Часто обрабатывать данные проще, их заранее упорядочить. Упорядоченные данные содержатся, например, в библиотеках, словарях, различных архивах.

В настоящее время проблема эффективной сортировки остается актуальной из-за постоянно растущих объемов данных.

Для достижения поставленной цели требуется решить следующие задачи:

- 1) изучение трех алгоритмов сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- 2) разработка трех алгоритмов сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- 3) реализация трех алгоритмов сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- 4) выполнение замеров процессорного времени работы реализаций алгоритмов сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- 5) сравнительный анализ трудоемкости реализаций разработанных алгоритмов сортировки на основе теоретических расчетов;

- 6) сравнительный анализ процессорного времени работы реализаций разработанных алгоритмов сортировки на основе экспериментальных данных.

# 1 Аналитическая часть

В данном разделе будут представлены описания алгоритмов сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором.

## 1.1 Гномья сортировка

Алгоритм гномьей сортировки похож на сортировку вставками, но в отличие от последней, перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком.[1]

Общие идеи алгоритма:

- обход массива ведется слева направо (от начала массива до его конца), аналогично пузырьковой сортировке сравниваются соседние элементы и меняются местами, если левое значение больше правого;
- если два соседних элемента пришлось поменять местами, то делается шаг назад на 1 элемент.

## 1.2 Поразрядная сортировка

Алгоритм поразрядной сортировки сильно отличается от других алгоритмов сортировки [2].

Во-первых, он совсем не использует сравнений сортируемых элементов.

Во-вторых, ключ, по которому происходит сортировка, необходимо разделить на части, разряды ключа (число делится на цифры, слова делятся на буквы).

До начала сортировки необходимо знать два параметра:  $k$  и  $m$ , где

- $k$  - количество разрядов в самом длинном ключе;
- $m$  - разрядность данных, то есть количество возможных значений разряда ключа.

Например, при сортировке десятичных чисел  $m=10$ , так как десятичная цифра может принимать не более 10 значений (от 0 до 9). Если в самом длинном числе 12 цифр, то  $k=12$ .

Эти параметры нельзя изменять в процессе работы алгоритма.

Основная идея алгоритма:

Сравнение производится поразрядно, а именно сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

## 1.3 Сортировка выбором

Алгоритм сортировки выбором основан на сравнении каждого элемента с каждым, в случае необходимости производится обмен [3].

Шаги выполнения алгоритма.

1. Проходим по массиву в поисках максимального элемента, запоминаем его номер;
2. Найденный максимум меняем местами с последним элементом (обмен не нужен, если максимальный элемент уже находится на нужной позиции);
3. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум);
4. К неотсортированной части применяем те же действия, то есть находим максимум и ставим его на последнее место в неотсортированной части массива;

5. Продолжаем таким образом до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.

Для устойчивости алгоритма необходимо в пункте 2 максимальный элемент непосредственно вставлять в последнюю неотсортированную позицию, не меняя порядок остальных элементов, иначе число обменов может резко возрасти.

## Вывод

В данном разделе были рассмотрены основные идеи, лежащие в основе рассматриваемых алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором).

## 2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором) и вычисления трудоемкости данных алгоритмов.

### 2.1 Алгоритм гномьей сортировки

На рисунке 2.1 приведена схема алгоритма гномьей сортировки.



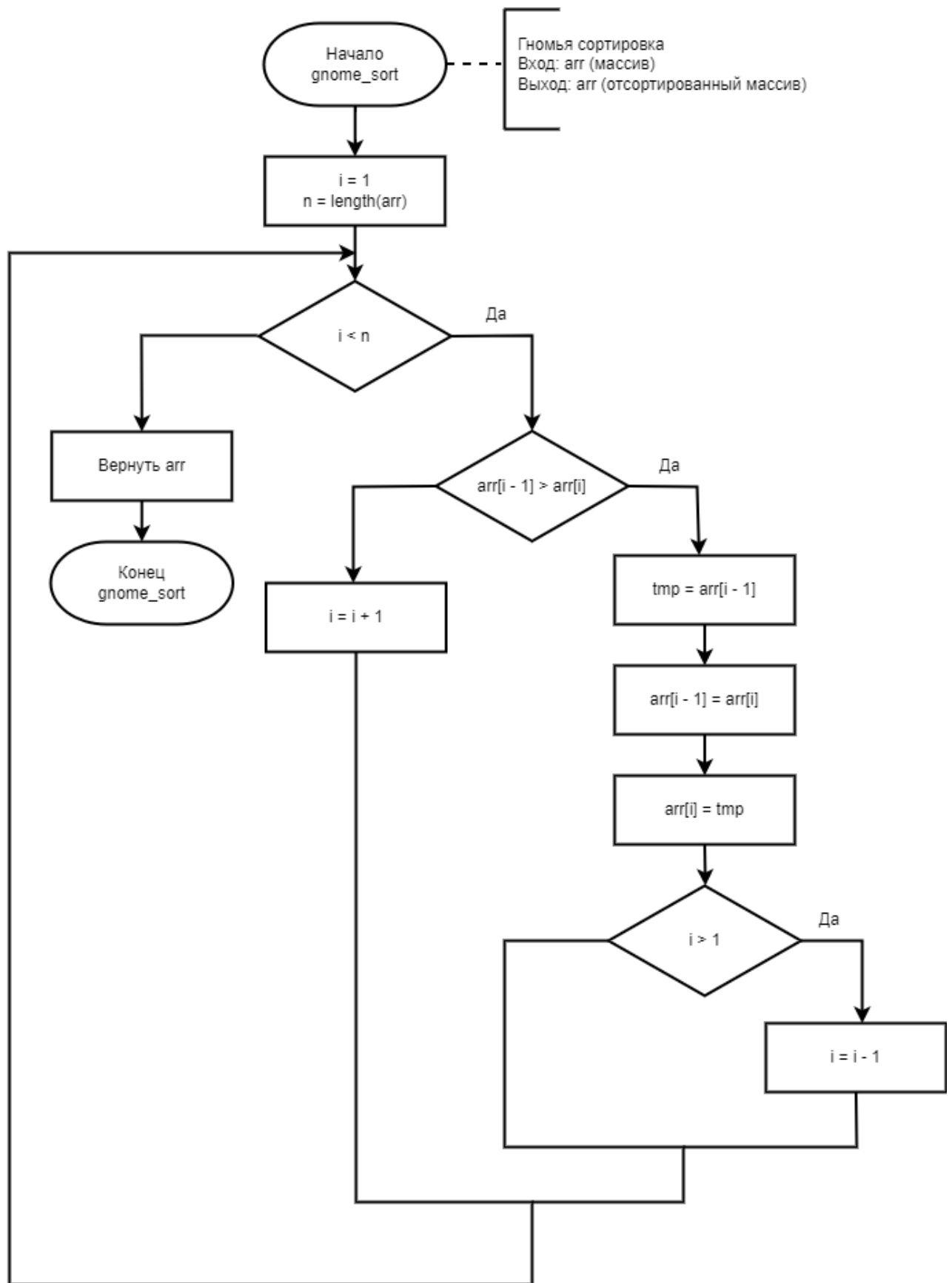


Рисунок 2.1 – Схема алгоритма гномьей сортировки

## 2.2 Алгоритм поразрядной сортировки

На рисунках 2.2 и 2.3 приведена схема алгоритма поразрядной сортировки.

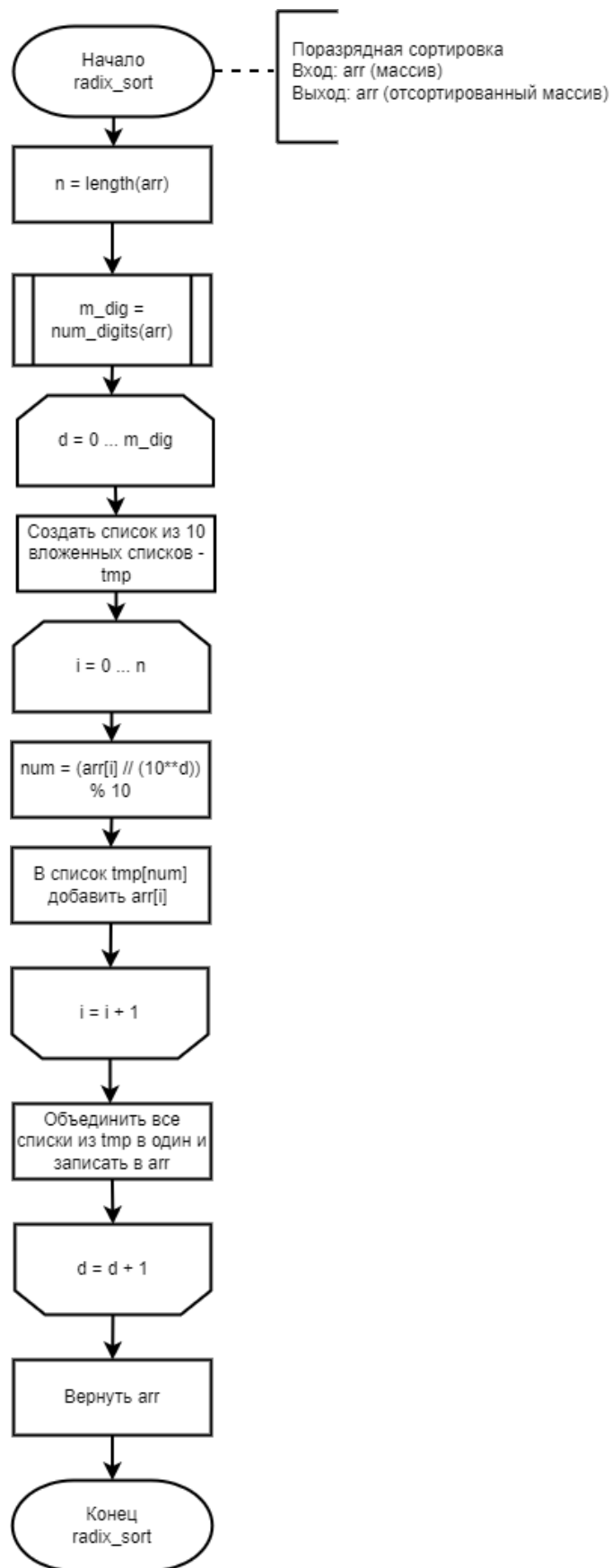


Рисунок 2.2 – Схема алгоритма поразрядной сортировки

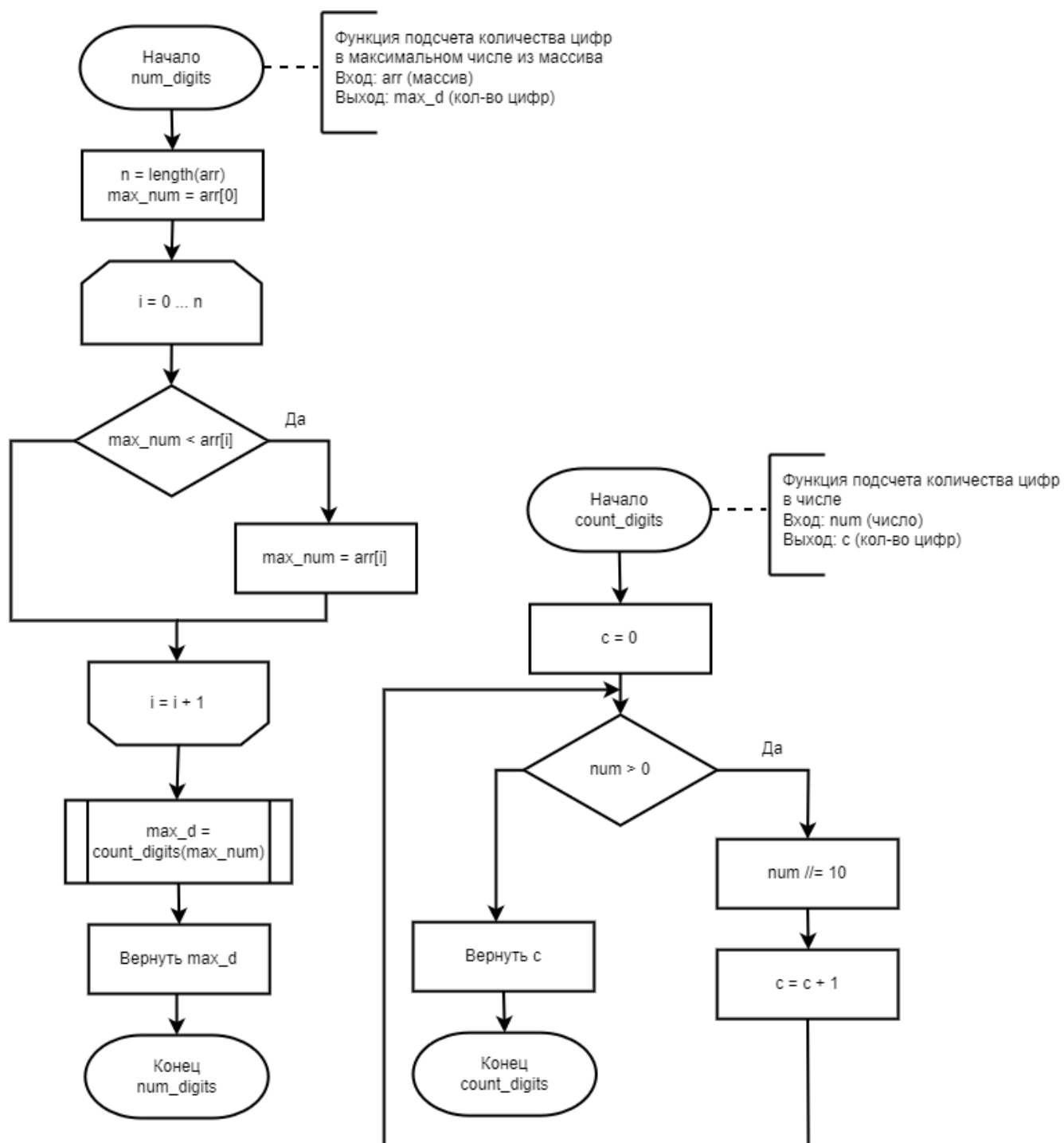


Рисунок 2.3 – Схема алгоритма поразрядной сортировки

## 2.3 Алгоритм сортировки выбором

На рисунке 2.4 приведена схема алгоритма сортировки выбором.

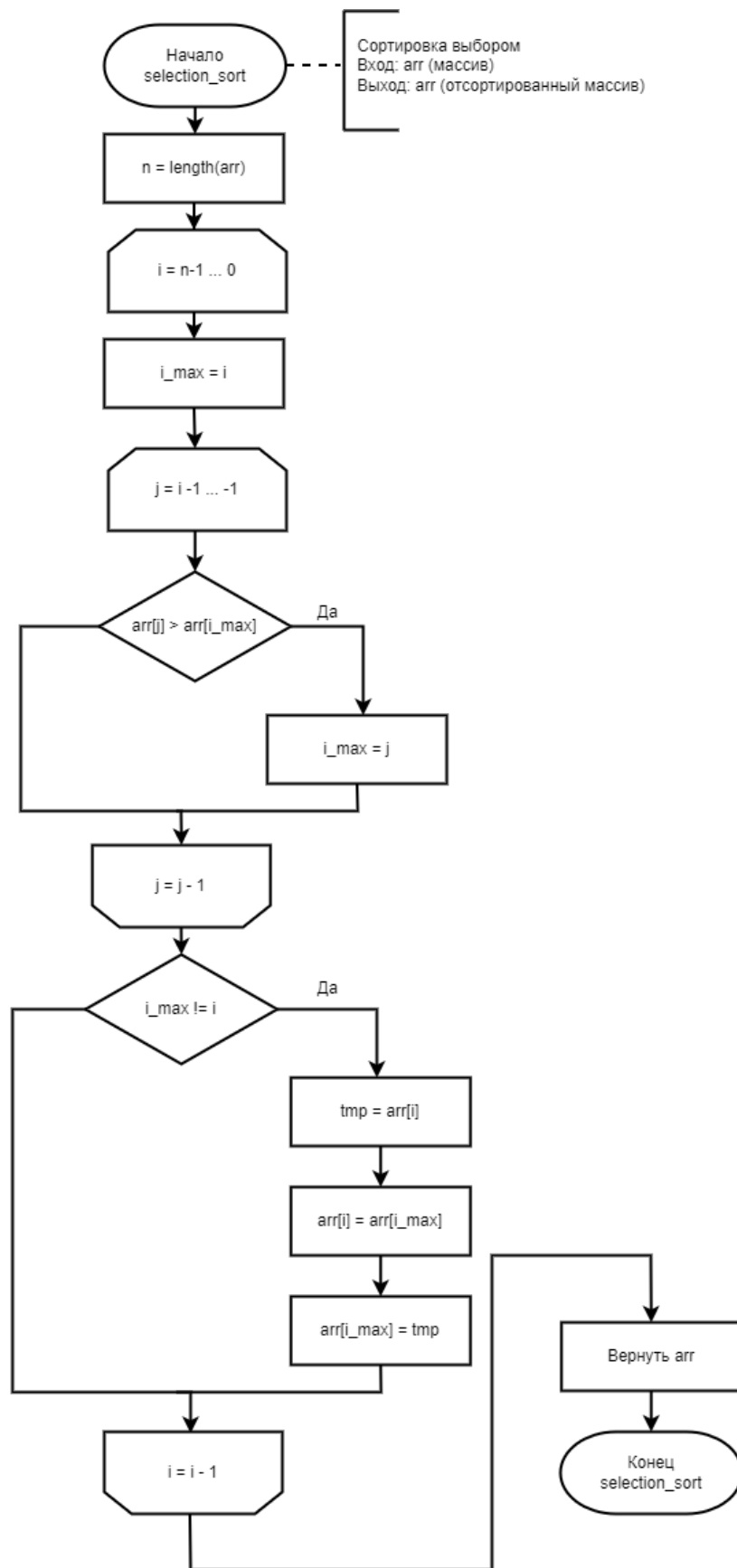


Рисунок 2.4 – Схема алгоритма сортировки выбором

## 2.4 Оценка трудоемкости алгоритмов сортировки

Для последующего вычисления трудоемкости необходимо ввести следующую модель вычислений:

- базовые операции с трудоемкостью 2:  $/$ ,  $/=$ ,  $*$ ,  $*=$ ,  $\%$ ,  $\%=$ ;
- базовые операции с трудоемкостью 1:  $+$ ,  $++$ ,  $+=$ ,  $-$ ,  $--$ ,  $-=$ ,  $==$ ,  $!=$ ,  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $\ll$ ,  $\gg$ ,  $[]$ ;
- трудоемкость цикла:  $F_{\text{цикла}} = F_{\text{иниц.}} + F_{\text{сравн.}} + M * (F_{\text{тела}} + F_{\text{инк.}} + F_{\text{сравн.}})$ , где  $F_{\text{иниц.}}$ ,  $F_{\text{сравн.}}$ ,  $F_{\text{тела}}$ ,  $F_{\text{инк.}}$  - трудоемкости инициализации, проверки условия цикла, тела цикла и инкрементирования соответственно, а  $M$  - количество итераций;
- трудоемкость условного оператора:  $F_{if} = F_{\text{сравн.}} +$

$$+ \begin{cases} \min(f1, f2) - \text{лучший случай} \\ \max(f1, f2) - \text{худший случай} \end{cases}, \quad (2.1)$$

где  $F_{\text{сравн.}}$ ,  $f1$ ,  $f2$  - трудоемкости проверки условия, первого блока и второго блока, соответственно.

Обозначим во всех последующих вычислениях размер массивов как  $N$ ,  $K$  - количество дополнительных итераций из-за обменов элементов.

### 2.4.1 Алгоритм гномьей сортировки

Трудоемкость алгоритма сортировки выбором состоит из:

- Трудоемкость цикла  $while(i < N)$ , которая равна (2.2):

$$f_{while} = 2 + 2 \cdot (N - 1) \cdot f_{if} \cdot (K + 1) \quad (2.2)$$

- Трудоемкость условия в цикле, которая равна (2.3):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \quad (2.3)$$

- Количество дополнительных итераций из-за обменов элементов (2.4):

$$K = \begin{cases} 0, & \text{л.с.} \\ N - 1, & \text{х.с.} \end{cases} \quad (2.4)$$

Трудоемкость в лучшем случае (2.5):

$$f_{best} = 8N - 6 \approx 8N = O(N) \quad (2.5)$$

Трудоемкость в худшем случае (2.6):

$$f_{worst} = 26N^2 - 13N + 2 \approx 26N^2 = O(N^2) \quad (2.6)$$

## 2.4.2 Алгоритм поразрядной сортировки

Трудоемкость алгоритма сортировки выбором состоит из:

- Трудоемкость подсчета  $m$  для цикла функции `radix_sort`, где  $m$  - количество цифр в максимальном числе из массива (2.7):

$$f_{count\_m} = 1 + f_{num\_digits} \quad (2.7)$$

- Трудоемкость функции `num_digits` (2.8):

$$f_{num\_digits} = 4 + f_{for} + f_{count\_digits} \quad (2.8)$$

- Трудоемкость сравнения, инкремента цикла функции `num_digits`, а также зависимых только от него операций, по  $d \in [0...N)$ , которая равна (2.9):

$$f_{for} = N * (2 + f_{if}) \quad (2.9)$$



- Трудоемкость условия в цикле, которая равна (2.10):

$$f_{if} = 2 + \begin{cases} 0, & \text{л.с.} \\ 2, & \text{х.с.} \end{cases} \quad (2.10)$$

- Трудоемкость функции `count_digits` (2.11):

$$f_{count\_digits} = 1 + f_{while} \quad (2.11)$$

- Трудоемкость цикла `while(num > 0)`, где  $m$  - количество цифр в максимальном числе из массива, равна (2.12):

$$f_{while} = m * (2 + 2) = 4m \quad (2.12)$$

- Трудоемкость сравнения, инкремента внешнего цикла функции `radix_sort`, а также зависимых только от него операций, по  $d \in [0...m)$ , где  $m$  - количество цифр в максимальном числе из массива, которая равна (2.13):

$$f_{outer} = 1 + m \cdot (42 + 11) + f_{inner} = 1 + 53m + f_{inner} \quad (2.13)$$

- Суммарная трудоемкость внутренних циклов функции `radix_sort` по  $i \in [0...N)$ , которая равна (2.14):

$$f_{inner} = m + N \cdot (2 + 3 + 6 + 2 \cdot \frac{(m - 1) \cdot m}{2}) = m + N \cdot (11 + m^2 - m) \quad (2.14)$$

Трудоемкость в лучшем случае, где  $m$  - количество цифр в максимальном числе из массива ( $m \ll N$ ), равна (2.15):

$$f_{best} = N \cdot (m^2 - m) + 58m + 15N + 7 \approx N * (m^2 - m + 15) = O(N) \quad (2.15)$$

Трудоемкость в худшем случае, где  $m$  - количество цифр в максимальном числе из массива ( $m \ll N$ ), равна (2.16):

$$f_{worst} = N \cdot (m^2 - m) + 58m + 17N + 7 \approx N * (m^2 - m + 17) = O(N) \quad (2.16)$$

### 2.4.3 Алгоритм сортировки выбором

Трудоёмкость алгоритма сортировки выбором состоит из:

- Трудоёмкость сравнения, декремента внешнего цикла, а также зависимых только от него операций, по  $i \in [N - 1..0]$ , которая равна (2.17):

$$f_{outer} = 3 + (5 + f_{ifout}) \cdot (N - 1) \quad (2.17)$$

- Трудоёмкость условия во внешнем цикле, которая равна (2.18):

$$f_{ifout} = 1 + \begin{cases} 0, & \text{л.с.} \\ 7, & \text{х.с.} \end{cases} \quad (2.18)$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке  $[N - 2..0]$ , которая равна (2.19):

$$f_{inner} = 2 \cdot (N - 1) + f_{ifin} \cdot \frac{(N - 2) \cdot (N - 1)}{2} \quad (2.19)$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.20):

$$f_{ifin} = 3 + \begin{cases} 0, & \text{л.с.} \\ 1, & \text{х.с.} \end{cases} \quad (2.20)$$

Трудоёмкость в лучшем случае (2.21):

$$f_{best} = \frac{3}{2}N^2 + \frac{9}{2}N - 2 \approx \frac{3}{2}N^2 = O(N^2) \quad (2.21)$$

Трудоёмкость в худшем случае (2.22):

$$f_{worst} = 2N^2 + 9N - 8 \approx 3N^2 = O(N^2) \quad (2.22)$$

## Вывод

В данном разделе были разработаны схемы алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором). Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

## 3 Технологическая часть

В данном разделе будут представлены требования к программному обеспечению, средства реализации, листинги кода и тесты.

### 3.1 Требования к программному обеспечению

Вход: массив целых чисел, количество чисел в массиве;

Выход: тот же массив, но уже отсортированный.

Результат сортировки массива должен быть выведен для каждого из реализованных алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором).

### 3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python [4]. Данный язык ускоряет процесс разработки и удобен в использовании.

Процессорное время реализованных алгоритмов было замерено с помощью функции `process_time()` из библиотеки `time` [5].

В качестве среды разработки был выбран PyCharm Professional [6]. Данная среда разработки является кросс-платформенной, предоставляет функциональный отладчик, средства для рефакторинга кода и возможность быстрой установки необходимых библиотек при необходимости.

### 3.3 Реализация алгоритмов

В листингах 3.1 - 3.4 представлены реализации различных алгоритмов сортировки: гномьей сортировки, поразрядной сортировки, сортировки выбором.

### Листинг 3.1 – Функция алгоритма гномьей сортировки

```
1 def gnome_sort(arr):
2     i = 1
3     n = len(arr)
4     while i < n:
5         if arr[i - 1] > arr[i]:
6             arr[i - 1], arr[i] = arr[i], arr[i - 1]
7             if i > 1:
8                 i -= 1
9         else:
10            i += 1
11    return arr
```

### Листинг 3.2 – Функция алгоритма поразрядной сортировки (классическая реализация для сортировки целых неотрицательных чисел)

```
1 # counting quantity of digits in number
2 def count_digits(num):
3     c = 0
4     while num > 0:
5         num /= 10
6         c += 1
7     return c
8
9
10 # counting quantity of digits in maximum number in array
11 def num_digits(arr):
12     max_num = arr[0]
13     n = len(arr)
14     for i in range(n):
15         if max_num < arr[i]:
16             max_num = arr[i]
17     return count_digits(max_num)
18
19
20 # main function
21 def radix_sort(arr):
22     m_dig = num_digits(arr)
23     for d in range(0, m_dig):
24         # 10, because of 10 possible digits (0 - 9)
25         tmp = [[] for i in range(10)]
26         for i in range(len(arr)):
```

```

27         num = (arr[i] // (10 ** d)) % 10
28         tmp[num].append(arr[i])
29     arr = reduce(lambda x, y: x + y, tmp)
30     return arr

```

Листинг 3.3 – Функция алгоритма поразрядной сортировки  
(модифицированная реализация для сортировки целых чисел)

```

1  # counting quantity of digits in number
2  def count_digits(num):
3      c = 0
4      num = abs(num)
5      while num > 0:
6          num //= 10
7          c += 1
8      return c
9
10
11 # counting quantity of digits in maximum number in array
12 def num_digits(arr):
13     max_num = abs(arr[0])
14     n = len(arr)
15     for i in range(n):
16         if max_num < abs(arr[i]):
17             max_num = abs(arr[i])
18     return count_digits(max_num)
19
20
21 # main function
22 def radix_sort(arr):
23     m_dig = num_digits(arr)
24     for d in range(0, m_dig):
25         # 10, because of 10 possible digits (0...9),
26         # but for negative numbers add digits (-9...-1)
27         tmp = [[] for i in range(19)]
28         for i in range(len(arr)):
29             if arr[i] < 0:
30                 num = -((abs(arr[i]) // (10 ** d)) % 10)
31             else:
32                 num = (arr[i] // (10 ** d)) % 10
33             tmp[9 + num].append(arr[i])
34     arr = reduce(lambda x, y: x + y, tmp)

```

### Листинг 3.4 – Функция алгоритма сортировки выбором

```

1 def selection_sort(arr):
2     n = len(arr)
3     for i in range(n - 1, 0, -1):
4         i_max = i
5         for j in range(i - 1, -1, -1):
6             if arr[j] > arr[i_max]:
7                 i_max = j
8         if i_max != i:
9             arr[i], arr[i_max] = arr[i_max], arr[i]
10    return arr

```

## 3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором). Все тесты были пройдены успешно.

Таблица 3.1 – Тесты

№	Исходный массив	Ожидаемый результат
1	0	0
2	5, 5, 5, 5	5, 5, 5, 5
3	1, 45, 78, 109, 764	1, 45, 78, 109, 764
4	78, 44, 32, 1	1, 32, 44, 78
5	-90, -356, -4, -1058	-1058, -356, -90, -4
6	234, 7, -11, 68, -3, 1005	-11, -3, 7, 68, 234, 1005

## Вывод

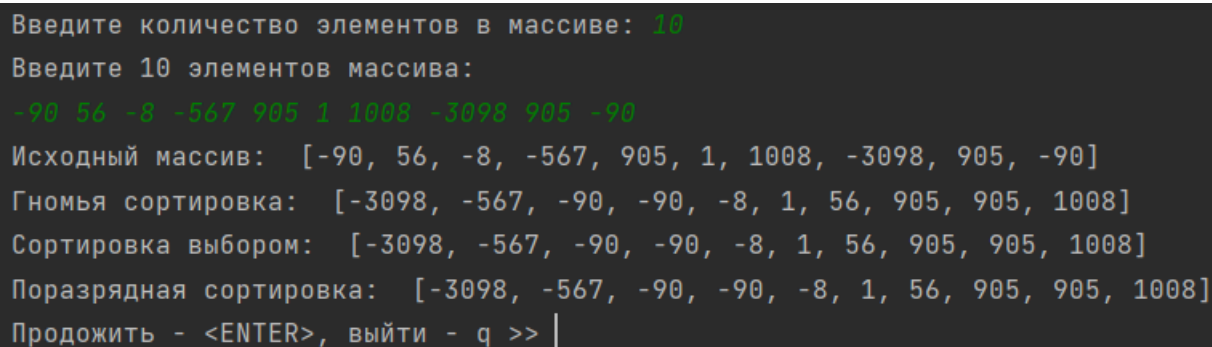
В данном разделе были представлены требования к программному обеспечению и средства реализации, реализованы и протестированы алгоритмы сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором.

## 4 Исследовательская часть

В текущем разделе будут представлены примеры работы разработанного программного обеспечения, постановка эксперимента и сравнительный анализ реализованных алгоритмов.

### 4.1 Пример работы программного обеспечения

На рисунке 4.1 представлен результат работы программы.



```
Введите количество элементов в массиве: 10
Введите 10 элементов массива:
-90 56 -8 -567 905 1 1008 -3098 905 -90
Исходный массив: [-90, 56, -8, -567, 905, 1, 1008, -3098, 905, -90]
Гномья сортировка: [-3098, -567, -90, -90, -8, 1, 56, 905, 905, 1008]
Сортировка выбором: [-3098, -567, -90, -90, -8, 1, 56, 905, 905, 1008]
Поразрядная сортировка: [-3098, -567, -90, -90, -8, 1, 56, 905, 905, 1008]
Продолжить - <ENTER>, выйти - q >> |
```

Рисунок 4.1 – Пример работы программы

### 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10 [7];
- оперативная память: 16 Гб;
- процессор: Intel® Core™ i5 10300H 2.5 ГГц.

Во время тестирования ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.



## 4.3 Время выполнения реализаций алгоритмов

Замеры процессорного времени реализованных алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором) проводились с помощью функции `process_time()` из библиотеки `time` языка Python.

Функция `process_time()` возвращает время в секундах (сумму системного и пользовательского процессорного времени).

Замеры времени для каждой длины массива (от 100 до 1000 элементов с шагом 200) проводились 100 раз для всех трех реализованных алгоритмов сортировки. В качестве результата бралось среднее время работы алгоритма на каждой длине массива.

Все реализации алгоритмов сортировки сравнивались на трех видах массивов: упорядоченный (лучший случай - все реализации алгоритмов сортировки выдают наименьшее время работы), упорядоченный в обратном порядке (худший случай - все реализации алгоритмов сортировки выдают наибольшее время работы), случайно сгенерированный массив (произвольный случай).

На рисунке 4.2 представлено сравнение процессорного времени работы реализаций алгоритмов сортировки на упорядоченных массивах (лучший случай). На графике видно, что алгоритм сортировки выбором на данном виде массивов менее эффективен по времени, чем алгоритмы поразрядной сортировки и гномьей сортировки, которые практически одинаково эффективны по времени. Алгоритм сортировки выбором не зависимо от массива выдает сложность  $O(n^2)$ .

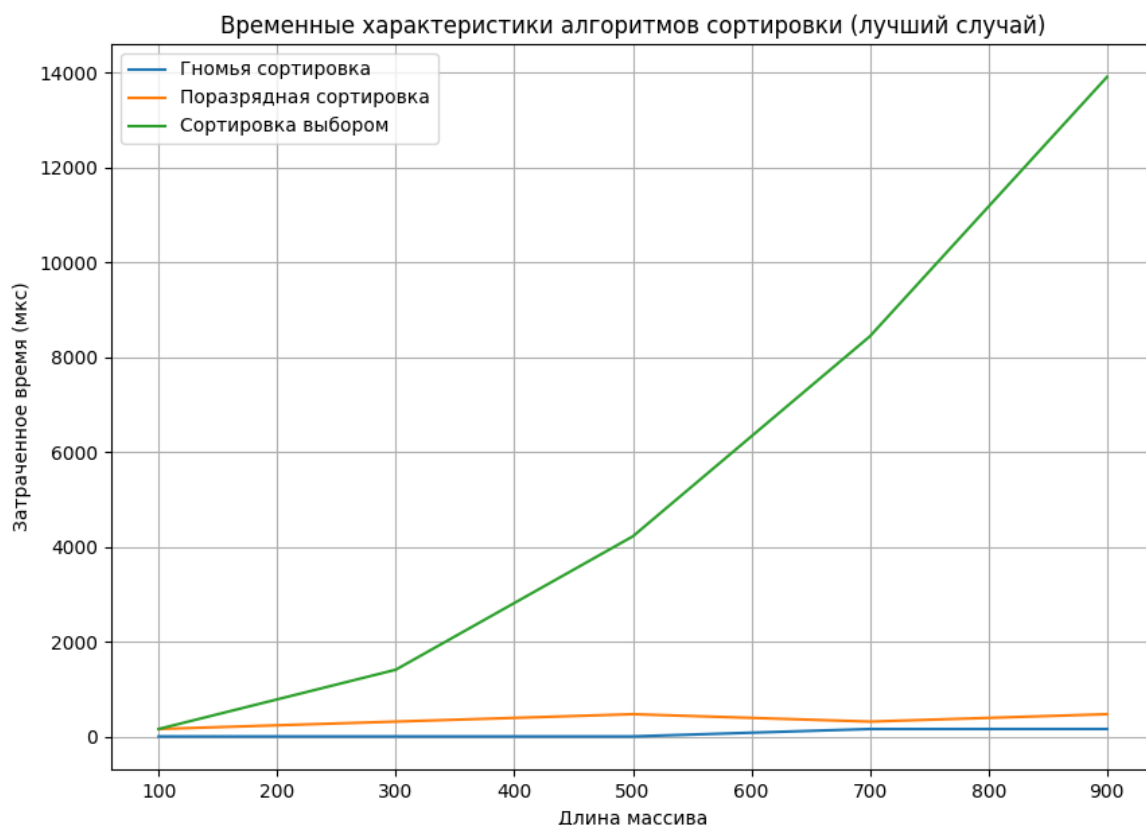


Рисунок 4.2 – Сравнение процессорного времени работы реализаций алгоритмов сортировки на упорядоченных массивах (лучший случай)

На рисунке 4.3 представлено сравнение процессорного времени работы реализаций алгоритмов сортировки на упорядоченных в обратном порядке массивах (худший случай). На графике видно, что алгоритм гномьей сортировки на данном виде массивов является самым неэффективным по времени среди реализованных алгоритмов, на втором месте по эффективности – алгоритм сортировки выбором, на первом месте (самый эффективный по времени на данном виде массивов) – алгоритм поразрядной сортировки, причем его преимущество растет с увеличением длины массива.

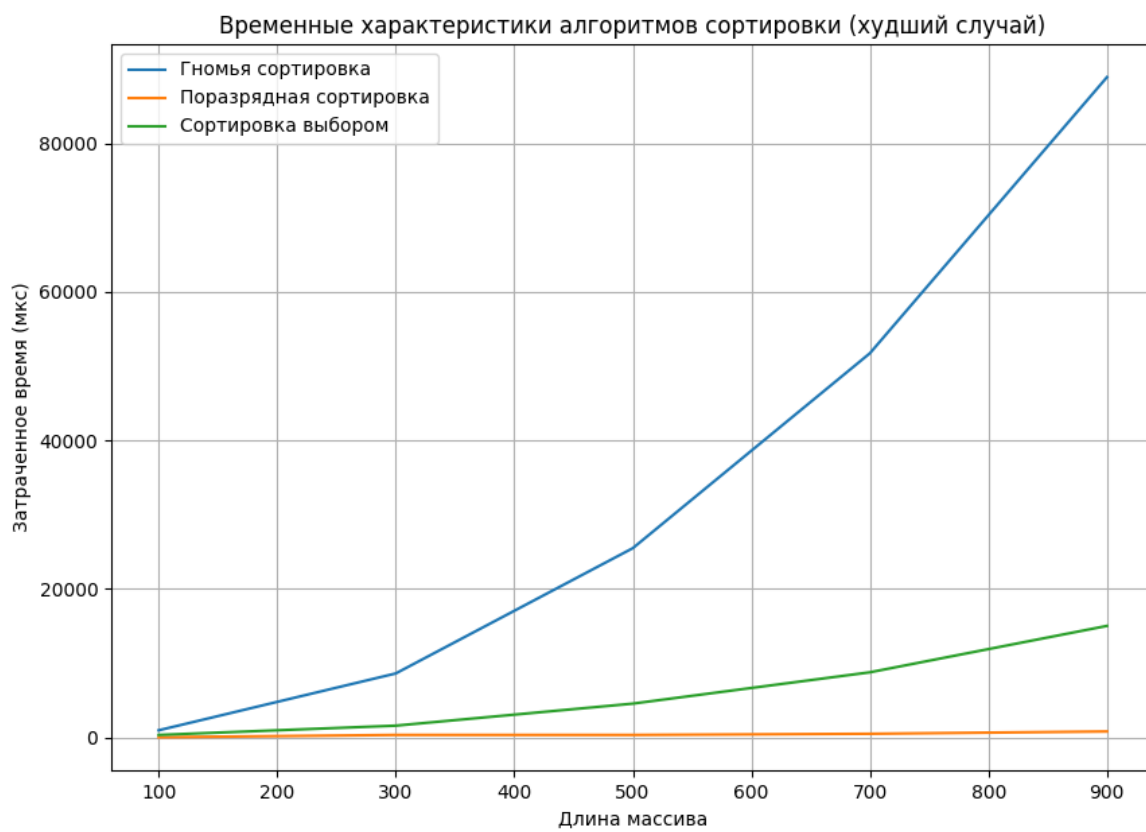


Рисунок 4.3 – Сравнение процессорного времени работы реализаций алгоритмов сортировки на упорядоченных в обратном порядке массивах (худший случай)

На рисунке 4.4 представлено сравнение процессорного времени работы реализаций алгоритмов сортировки на случайно сгенерированных массивах (произвольный случай). На графике видно, что алгоритм гномьей сортировки на данном виде массивов является самым неэффективным по времени среди реализованных алгоритмов, на втором месте по эффективности - алгоритм поразрядной сортировки, на первом месте (самый эффективный по времени на данном виде массивов) - алгоритм сортировки выбором, причем его преимущество растет с увеличением длины массива.

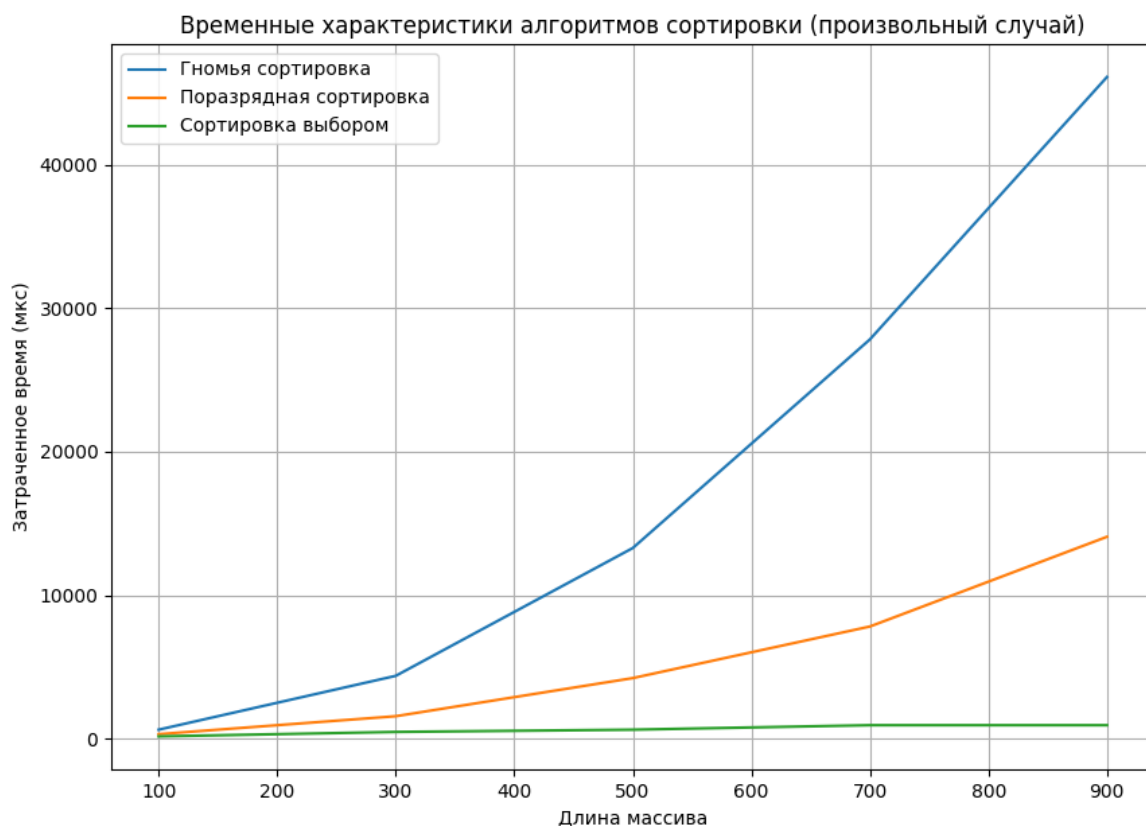


Рисунок 4.4 – Сравнение процессорного времени работы реализаций алгоритмов сортировки на случайно сгенерированных массивах (произвольный случай)

## Вывод

Реализация алгоритма гномьей сортировки показывает худший результат среди реализованных алгоритмов сортировки по эффективности по времени в худшем и произвольном случаях и квадратично зависит от количества элементов в массиве, однако если массив уже был упорядочен, что бывает достаточно редко, реализация алгоритма гномьей сортировки показывает лучший результат наравне с поразрядной сортировкой.

Реализация алгоритма сортировки выбором является наиболее эффективной по времени в произвольном случае, однако в худшем случае проигрывает по эффективности по времени поразрядной сортировке, причем

отставание растет с увеличением числа элементов в массиве. В лучшем случае сортировка выбором, наоборот, показывает худший результат среди реализованных алгоритмов.

Реализация алгоритма поразрядной сортировки является наиболее эффективной по времени в лучшем случае наравне с гномьей сортировкой и в худшем случае. В произвольном случае поразрядная сортировка уступает лишь сортировке выбором.

Таким образом, самой эффективной по времени по экспериментальным данным с учетом работы на всех видах массивов является поразрядная сортировка, но классическая реализация алгоритма поразрядной сортировки позволяет корректно сортировать только неотрицательные целые числа, для сортировки любых целых чисел или вещественных чисел потребуется модифицировать алгоритм.

# Заключение

В результате выполнения лабораторной работы цель достигнута, а именно на примере трех сортировок (гномья сортировка, поразрядная сортировка, сортировка выбором) были получены навыки оценки трудоемкости алгоритмов.

В ходе выполнения данной работы были решены следующие задачи:

- изучены три алгоритма сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- разработаны три алгоритма сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- реализованы алгоритмы сортировки: гномья сортировка, поразрядная сортировка, сортировка выбором;
- выполнены замеры процессорного времени работы реализаций алгоритмов сортировки (гномья сортировка, поразрядная сортировка, сортировка выбором);
- проведен сравнительный анализ трудоемкости реализаций разработанных алгоритмов сортировки на основе теоретических расчетов;
- проведен сравнительный анализ процессорного времени работы реализаций разработанных алгоритмов сортировки на основе полученных экспериментальных данных.

В результате лабораторной работы можно сделать вывод, что самой эффективной по времени (на основе полученных экспериментальных данных) в лучшем случае наравне с реализацией алгоритма гномьей сортировки и худшем случае является реализация алгоритма поразрядной сортировки, самой эффективной по времени в произвольном случае является реализация алгоритма сортировки выбором.

# Литература

- [1] Шагбазян Д.В. Штанюк А.А. Малкина Е.В. Алгоритмы сортировки. Анализ, реализация, применение. Нижний Новгород: Нижегородский госуниверситет, 2019. Т. 42.
- [2] Поразрядная сортировка [Электронный ресурс]. Режим доступа: [http://algotlist.ru/sort/radix\\_sort.php](http://algotlist.ru/sort/radix_sort.php) (дата обращения: 04.10.2022).
- [3] Сортировка выбором [Электронный ресурс]. Режим доступа: <https://kvodo.ru/sortirovka-vyiborom-2.html> (дата обращения: 04.10.2022).
- [4] Лутц Марк. Изучаем Python, том 1, 5-е изд. Пер. с англ. — СПб.: ООО “Диалектика”, 2019. Т. 832.
- [5] time — Time access and conversions [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html> (дата обращения: 20.09.2021).
- [6] Узнайте все о PyCharm [Электронный ресурс]. Режим доступа: <https://www.jetbrains.com/ru-ru/pycharm/learn/> (дата обращения: 20.09.2022).
- [7] Windows 10 [Электронный ресурс]. Режим доступа: <https://learn.microsoft.com/ru-ru/windows/> (дата обращения: 20.09.2022).