



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 Трудоёмкость сортировок

Студент Фам Минь Хиеу

Группа ИУ7-52Б

Оценка (баллы) \_\_\_\_\_

Преподаватель Волкова Л.Л., Строганов Д.В.

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Бисерная сортировка . . . . .	4
1.2 Поразрядная сортировка . . . . .	4
1.3 Сортировка бинарным деревом . . . . .	5
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Разработка алгоритмов . . . . .	7
2.2 Описание типов данных . . . . .	15
2.3 Модель вычислений для проведения оценки трудоемкости .	15
2.4 Трудоемкость алгоритмов . . . . .	16
2.4.1 Бисерная сортировка . . . . .	16
2.4.2 Поразрядная сортировка . . . . .	17
2.4.3 Сортировка бинарным деревом . . . . .	17
<b>3 Технологическая часть</b>	<b>19</b>
3.1 Требования к программному обеспечению . . . . .	19
3.2 Средства реализации . . . . .	19
3.3 Сведения о модулях программы . . . . .	19
3.4 Реализация алгоритмов . . . . .	20
3.5 Функциональные тесты . . . . .	23
<b>4 Исследовательская часть</b>	<b>24</b>
4.1 Технические характеристики . . . . .	24
4.2 Демонстрация работы программы . . . . .	24
4.3 Результаты замера процессорного времени . . . . .	26
4.4 Характеристики по памяти . . . . .	31
4.5 Вывод . . . . .	34
<b>Заключение</b>	<b>36</b>
<b>Список использованных источников</b>	<b>37</b>

# Введение

При решении различных задач встает необходимость работы с упорядоченным набором данных. Например, при поиске элемента в заданном множестве. Для упорядочивания последовательности значений используется сортировка.

**Сортировка** — процесс перегруппировки заданного множества объектов в некотором определенном порядке. Для реализации этого процесса разрабатываются алгоритмы сортировки [1].

Существует большое количество алгоритмов сортировки. Все они решают одну и ту же задачу, причем некоторые алгоритмы имеют преимущества перед другими. Поэтому существует необходимость сравнительного анализа алгоритмов сортировки.

Целью данной лабораторной работы является описание трудоемкости алгоритмов сортировки. Для решения поставленной цели требуется решить следующие задачи:

- описать алгоритмы бисерной сортировки, поразрядной сортировки, сортировки бинарным деревом;
- построить схемы рассматриваемых алгоритмов;
- создать программное обеспечение, реализующее перечисленные алгоритмы;
- провести сравнительный анализ реализованных алгоритмов по времени и по памяти;
- подготовить отчет о выполненной лабораторной работе.

# 1 Аналитическая часть

В данном разделе будут описаны алгоритмы сортировок: бисерная сортировка, поразрядная сортировка и сортировка бинарным деревом.

## 1.1 Бисерная сортировка

**Бисерная сортировка** — это уникальный алгоритм сортировки, который использует множество маленьких бусин (шариков), представляющих собой числа или элементы в сортируемом массиве. Он основан на идее использования гравитации для сортировки [2].

Принцип работы бисерной сортировки следующий:

- создается вертикальный стержень для каждой цифры или элемента в массиве;
- каждая бусина помещается на соответствующий стержень, количество бусин на стержне равно значению этой цифры или элемента;
- бусины начинают скользить вниз под действием гравитации;
- после процесса скольжения бусины распределены по стержням в отсортированном порядке.

## 1.2 Поразрядная сортировка

**Поразрядная сортировка** — это алгоритм сортировки, который основывается на разрядах (цифрах) чисел или символов в сортируемом наборе данных [3].

Её принцип работы заключается в следующем:

- первый этап алгоритма начинается с наименьшего разряда (обычно справа) числа и переходит к наибольшему разряду (слева). Для целых чисел это означает, что сначала сортируются единицы, затем десятки, сотни и так далее;

- на каждом этапе сортировки элементы разбиваются на группы по значению текущего разряда. Элементы с одинаковым значением разряда помещаются в одну группу;
- затем элементы в каждой группе упорядочиваются, обычно с использованием стабильной сортировки, такой как сортировка подсчетом или сортировка вставками;
- процесс разбиения на группы и сортировки повторяется для каждого разряда, начиная с наименьшего и заканчивая наибольшим;
- после завершения сортировки по всем разрядам, элементы находятся в отсортированном порядке.

Поразрядная сортировка обычно применяется к числам или строкам, где разряды представляют собой цифры или символы. Этот алгоритм отличается от сортировок, которые сравнивают элементы напрямую, и может быть эффективным для сортировки больших объемов данных, особенно если разрядность элементов ограничена.

## 1.3 Сортировка бинарным деревом

**Сортировка бинарным деревом** — алгоритм сортировки, который основан на структуре данных двоичного дерева поиска (Binary Search Tree, BST) [4].

Принцип сортировки бинарным деревом:

- создается пустое двоичное дерево поиска;
- все элементы, которые нужно отсортировать, последовательно вставляются в это дерево. При вставке элемента, он сравнивается с элементами уже находящимися в дереве. Если он меньше, то он помещается в левое поддерево, а если больше то в правое поддерево;
- после вставки всех элементов, обход в глубину (in-order traversal) дерева производит элементы в отсортированном порядке. Этот обход

следует левому поддереву, затем текущему узлу и, наконец, правому поддереву;

- элементы получаются в порядке, обратном порядку их вставки, поэтому для получения отсортированного списка элементов, их нужно сохранить в массив или другую структуру данных.

## Вывод

В данном разделе были теоретически разобраны алгоритмы бисерной сортировки, поразрядной сортировки и сортировки бинарным деревом.

## 2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов сортировок: бисерной сортировки, поразрядной сортировки и сортировки бинарным деревом.

### 2.1 Разработка алгоритмов

Схемы алгоритма бисерной сортировки представлены на рисунках 2.1–2.2. Схемы алгоритма поразрядной сортировки представлены на рисунках 2.3–2.4. Схемы алгоритма сортировки бинарным деревом представлены на рисунках 2.5–2.7.

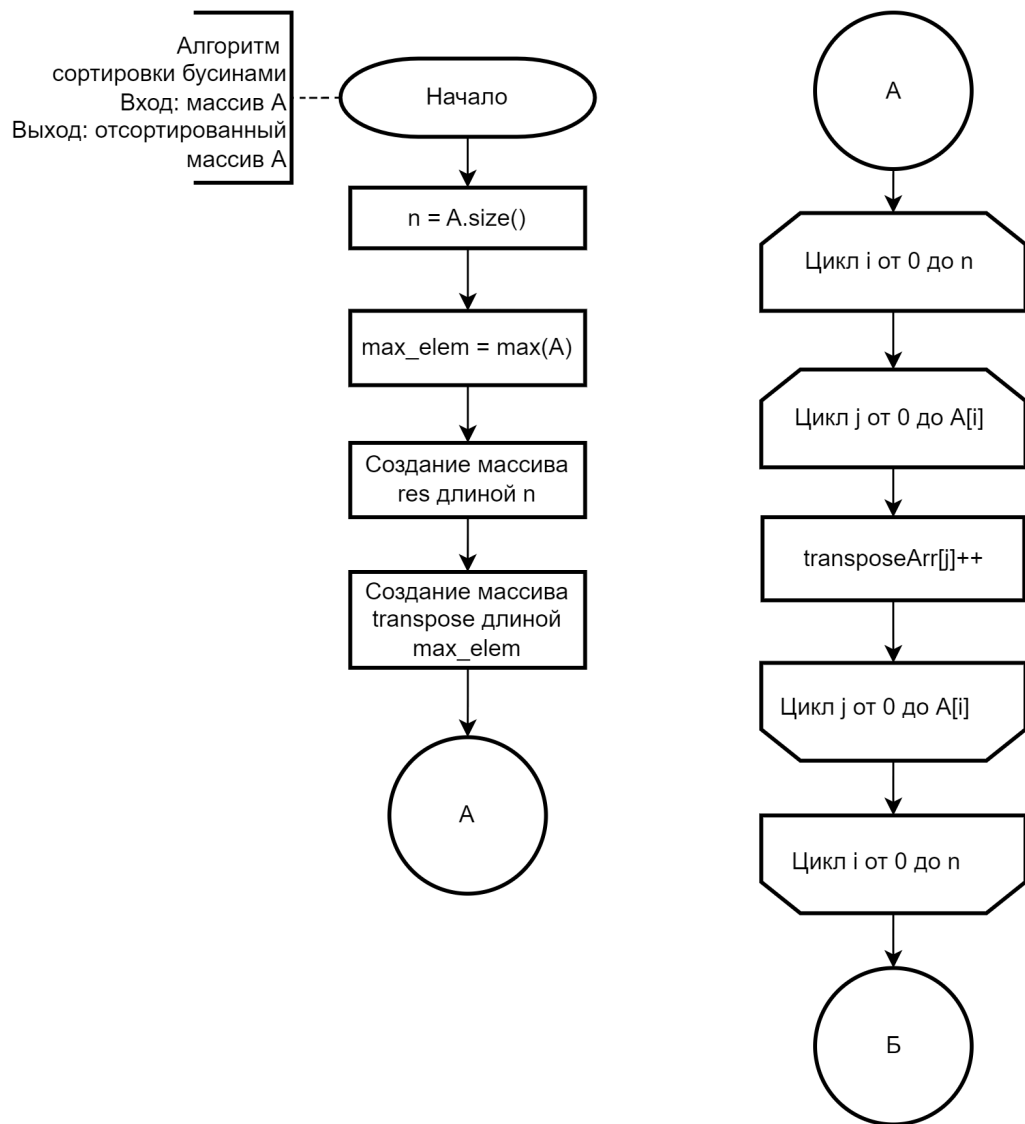


Рисунок 2.1 – Схема алгоритма бисерной сортировки (часть 1)



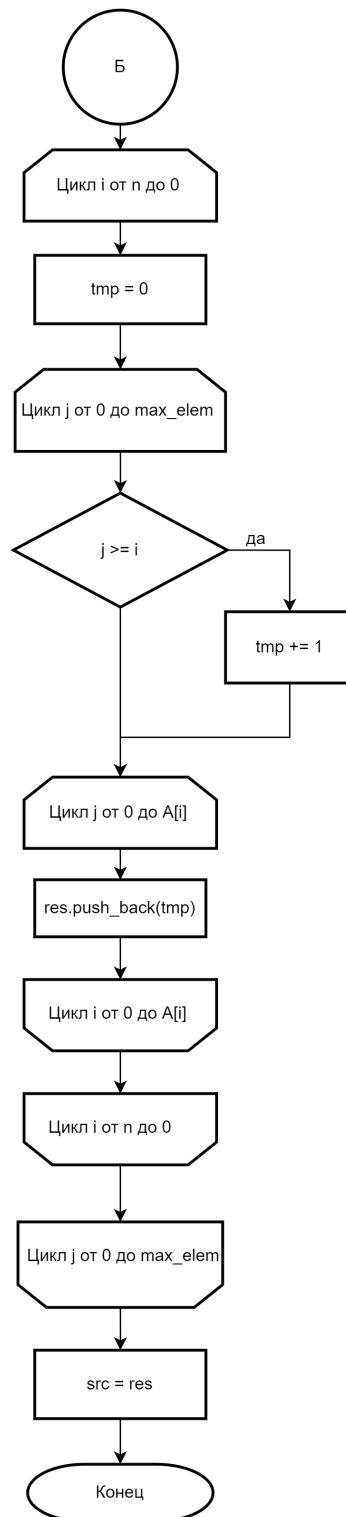


Рисунок 2.2 – Схема алгоритма бисерной сортировки (часть 2)

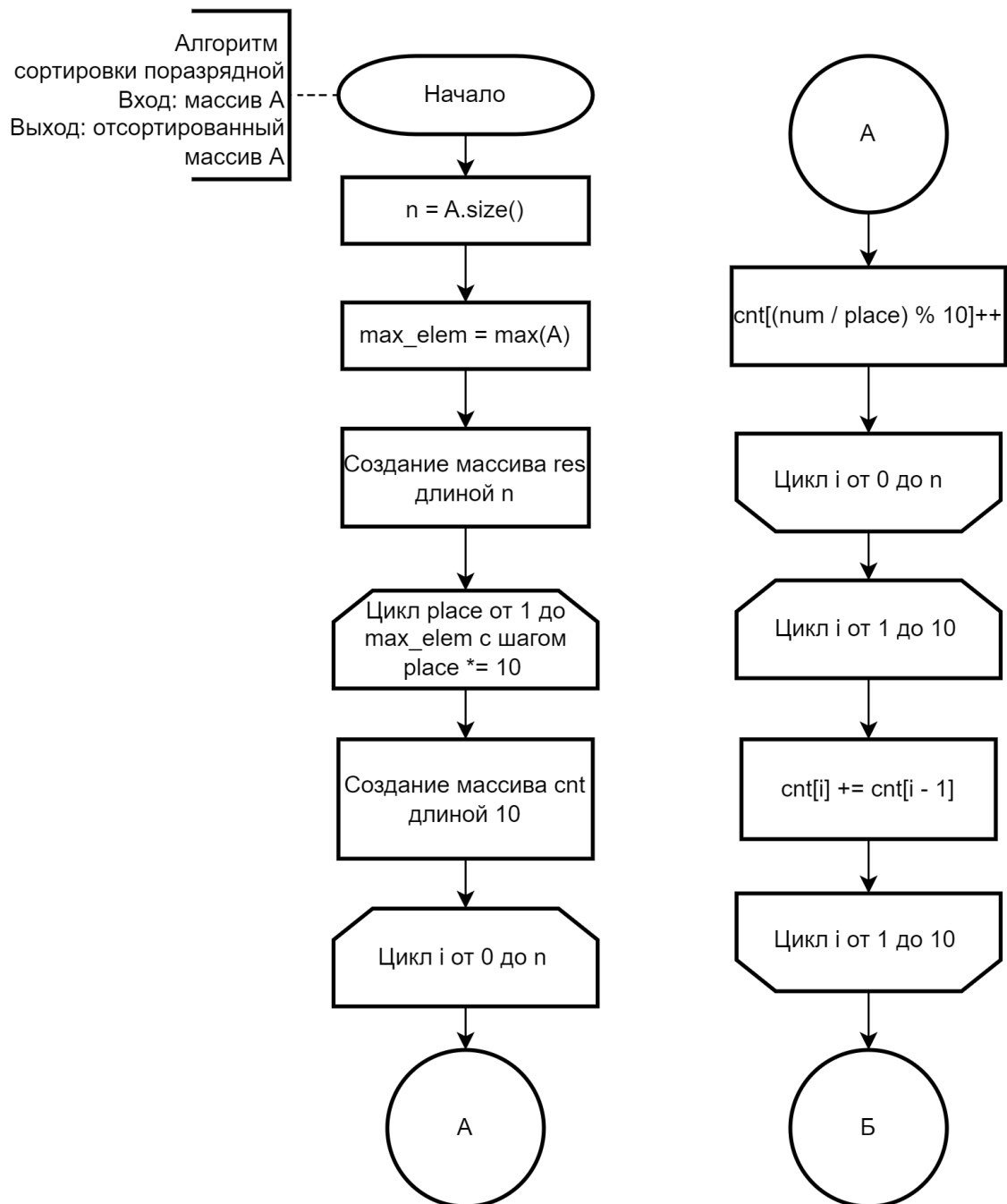


Рисунок 2.3 – Схема алгоритма поразрядной сортировки (часть 1)

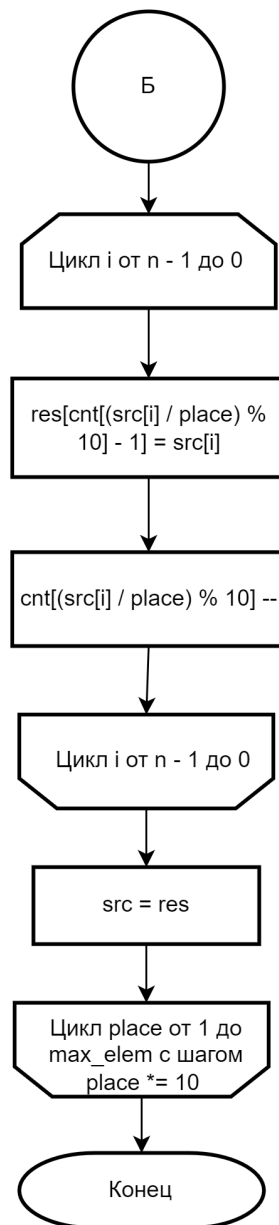


Рисунок 2.4 – Схема алгоритма поразрядной сортировки (часть 2)

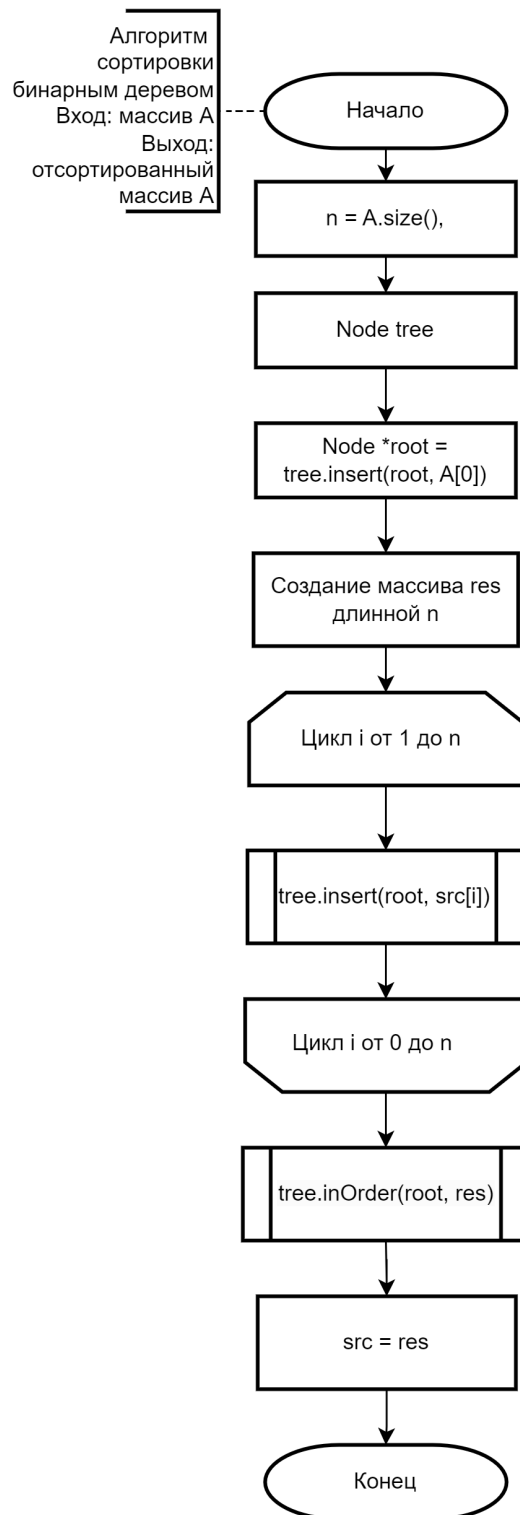


Рисунок 2.5 – Схема алгоритма сортировки бинарным деревом (часть 1)

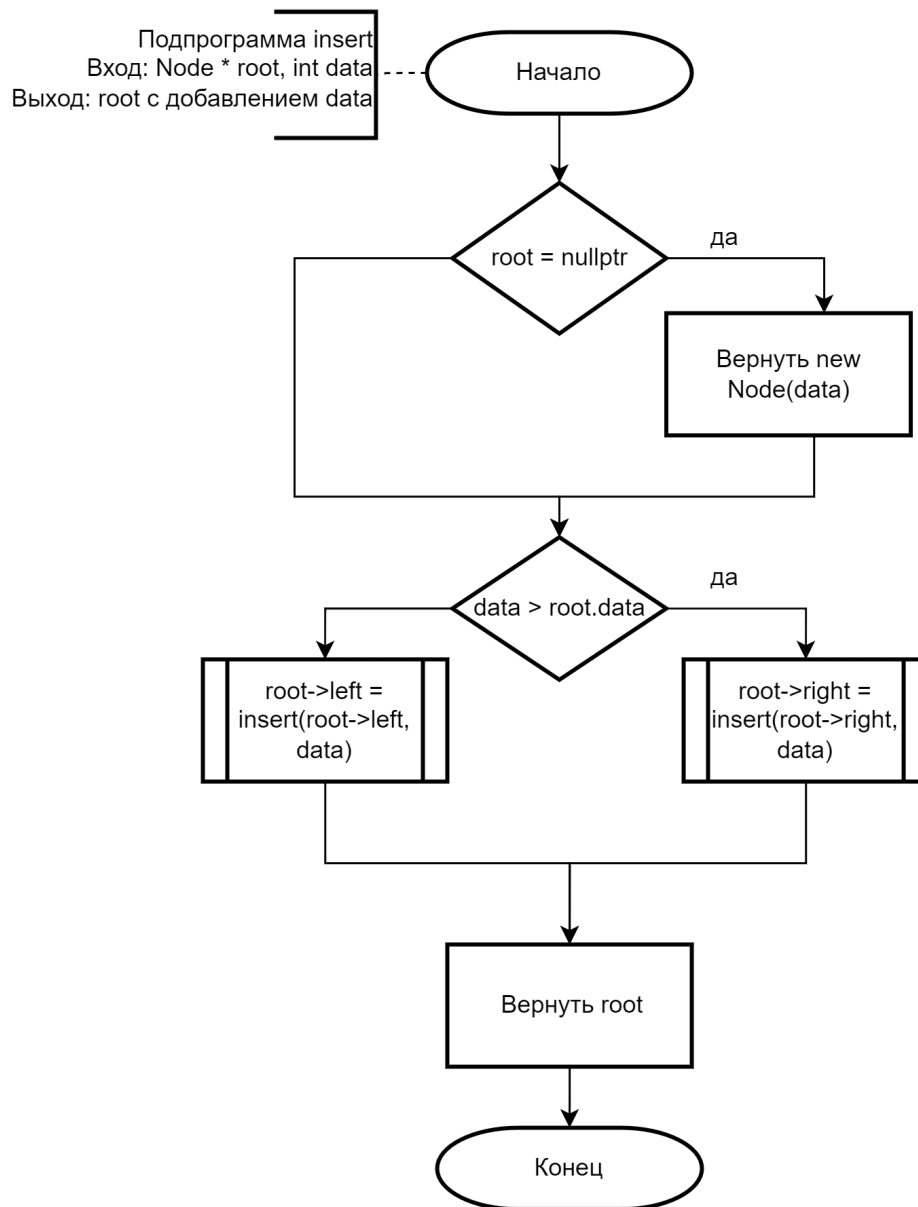


Рисунок 2.6 – Схема алгоритма сортировки бинарным деревом (часть 2)

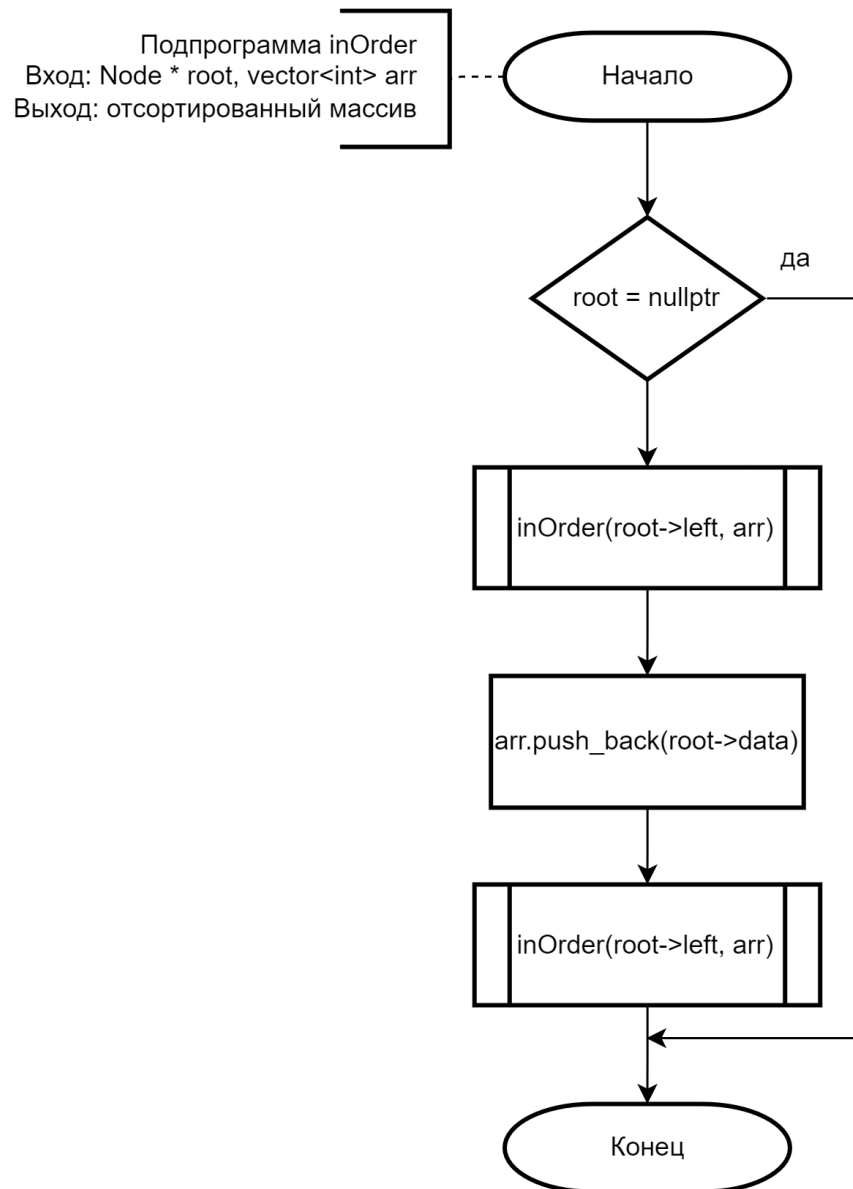


Рисунок 2.7 – Схема алгоритма сортировки бинарным деревом (часть 3)

## 2.2 Описание типов данных

При реализации алгоритмов будут использованы следующие типы и структуры данных:

- 1) массив — одномерный массив целых чисел;
- 2) целое число — длина массива.

## 2.3 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма сортировки.

- 1) Трудоемкость базовых операций.
  - Трудоемкость равна 1 для следующих базовых операций: +, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, --, &&, », «, ||, &, |.
  - Трудоемкость равна 2 для следующих базовых операций: \*, /, %, \*=, /=, %=
- 2) Трудоемкость условного оператора:

$$f_{if} = f_{условия} + \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай} \end{cases} \quad (2.1)$$

где  $f_1$  — это трудоёмкость вычисления блок *if*,  $f_2$  — это трудоёмкость вычисления блок *else*.

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (2.2)$$

4) Трудоемкость передачи параметра в функции и возврат из функции равны 0.

## 2.4 Трудоемкость алгоритмов

### 2.4.1 Бисерная сортировка

Для алгоритма бинарной сортировки трудоемкость будет складываться из следующих составляющих:

- определения длины  $N$ , максимального значения  $\max\_elem(M)$  массива, их трудоёмкость равна

$$f_{def} = N + N = 2N; \quad (2.3)$$

- создания и заполнения массива  $transposedArr$ , трудоёмкость которого равна

$$\begin{aligned} f_{init} &= M + 2 + N \cdot (2 + 2 + M \cdot (2 + 1 + 1)) = \\ &= 4MN + 4N + M + 2; \end{aligned} \quad (2.4)$$

- получения отсортированного массива с помощью массива  $transposedArr$ , трудоёмкость которого равна

$$\begin{aligned} f_{res} &= 2 + N \cdot (2 + 2 + 2 + M \cdot (1 + 1)) = \\ &= 2MN + 6N + 2; \end{aligned} \quad (2.5)$$

- присваивание временного массива исходному, трудоёмкость которого равна

$$f_{assign} = N. \quad (2.6)$$



Поэтому трудоёмкость алгоритма бисерной сортировки вычисляется следующим образом:

$$f_{bead\_sort} = f_{def} + f_{init} + f_{res} + f_{assign} \approx 6NM \quad (2.7)$$

### 2.4.2 Поразрядная сортировка

Для алгоритма поразрядной сортировки трудоёмкость будет складываться из следующих составляющих:

- определения длины  $N$ , максимального значения  $max\_elem(M)$  массива, их трудоёмкость равна

$$f_{def} = N + N = 2N; \quad (2.8)$$

- главного цикла, сортирующего массив по каждой цифре, трудоёмкость которого равна

$$\begin{aligned} f_{res} &= 4 + \log_{10} M \cdot (17 + 8 \cdot N) + 2 + 10 \cdot (2 + 4) + 2 + 19 \cdot N + N = \\ &= 28N \log_{10} M + 81 \log_{10} M + 4; \end{aligned} \quad (2.9)$$

- присваивание временного массива исходному, трудоёмкость которого равна

$$f_{assign} = N. \quad (2.10)$$

Поэтому трудоёмкость алгоритма бисерной сортировки вычисляется следующим образом:

$$f_{radix\_sort} = f_{def} + f_{res} + f_{assign} \approx 28N \log_{10} M = O(N \log(M)) \quad (2.11)$$

### 2.4.3 Сортировка бинарным деревом

Трудоёмкость алгоритма сортировки бинарным деревом состоит из следующих составляющих:

— трудоемкость построения бинарного дерева, которая равна

$$f_{make\_tree} = \begin{cases} N \cdot \log(N), & \text{неотсортированный массив} \\ N \cdot N, & \text{отсортированный массив} \end{cases} \quad (2.12)$$

— трудоемкость восстановления порядка элементов массива, которая равна

$$f_{make\_arr} = \begin{cases} N \cdot \log(N), & \text{дерево сбалансировано} \\ N \cdot N, & \text{дерево несбалансировано} \end{cases} \quad (2.13)$$

— трудоемкость присваивания временного массива исходному, которая равна

$$f_{res} = N. \quad (2.14)$$

Тогда для худшего случая (массив отсортирован) имеем:

$$f_{worst} = f_{make\_tree} + f_{make\_arr} + f_{assign\_arr} \approx N \cdot N = O(N^2) \quad (2.15)$$

Для лучшего случая (массив неотсортирован) имеем:

$$f_{best} = f_{make\_tree} + f_{make\_arr} + f_{assign\_arr} \approx N \log(N) = O(N \log(N)) \quad (2.16)$$

## Вывод

Были представлены схемы алгоритмов сортировок. Были указаны типы данных, используемые для реализации. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что в трудоёмкость выполнения алгоритма бисерной сортировки и поразрядной сортировки зависит от количество элементов и максимального значения массива. Также трудоёмкость выполнения алгоритма сортировки бинарным деревом в лучшем случае  $N \log(N)$ , а в худшем случае  $N^2$ .

## 3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены реализации алгоритмов, а также функциональные тесты.

### 3.1 Требования к программному обеспечению

Программе, реализующей данные алгоритмы, на вход будет подаваться массив. Выходным данным такой программы должна быть такой отсортированный массив. Программа должна работать в рамках следующих ограничений:

- 1) количество элементов массива — положительное целое число;
- 2) элементы массива — натуральные числа;
- 3) должно быть выдано сообщение об ошибке при вводе пустого массива.

### 3.2 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования C++. Данный выбор обусловлен наличием у языка функции *clock()* [5] измерения процессорного времени.

Визуализация графиков с помощью библиотеки *Matplotlib* [6].

### 3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

- `main.cpp` — точка входа программы;
- `algo.h` — заголовочный файл, содержащий объявления функций, реализующих рассматриваемых алгоритмов;

- algo.cpp — файл, содержащий реализации этих функций;
- measure.h — заголовочный файл, содержащий объявления функций замеров времени работы и затравоченной памяти рассматриваемых алгоритмов;
- measure.cpp — algo файл, содержащий реализации функций замеров времени работы и затравоченной памяти рассматриваемых алгоритмов.

## 3.4 Реализация алгоритмов

Алгоритм бисерной сортировки, алгоритм поразрядной сортировки и алгоритм сортировки бинарным деревом приведены в листингах 3.1–3.3.

Листинг 3.1 – Алгоритм бисерной сортировки

```

1 void beadSort(vector<int>& src)
2 {
3     size_t n = src.size();
4     vector<int> res;
5     int max_elem = *max_element(src.begin(), src.end());
6     vector<int> transposedArr(max_elem, 0);
7
8     for (size_t i = 0; i < n; i++)
9         for (int j = 0; j < src[i]; j++)
10             transposedArr[j]++;
11
12     for (size_t i = n; i > 0; i--)
13     {
14         int tmp = 0;
15         for (int j : transposedArr)
16             tmp += (j >= static_cast<int>(i));
17         res.push_back(tmp);
18     }
19     src = res;
20 }
```

Листинг 3.2 – Алгоритм поразрядной сортировки

```

1 void radixSort(vector<int>& src)
2 {
3     size_t n = src.size();
4     int max_elem = *max_element(src.begin(), src.end());
```

```

5     vector<int> res(n, 0);
6
7     for (int place = 1; max_elem / place > 0; place *= 10)
8     {
9         vector<int> cnt(10, 0);
10        for (int num : src)
11            cnt[(num / place) % 10]++;
12
13        for (size_t i = 1; i < 10; i++)
14            cnt[i] += cnt[i - 1];
15
16        for (int i = n - 1; i >= 0; i--)
17        {
18            res[cnt[(src[i] / place) % 10] - 1] = src[i];
19            cnt[(src[i] / place) % 10]--;
20        }
21        src = res;
22    }
23 }

```

Листинг 3.3 – Алгоритм сортировки бинарным деревом

```

1 Node* Node::insert(Node* root, const int& data)
2 {
3     if (!root)
4         return new Node(data);
5     if (data > root->data)
6         root->right = insert(root->right, data);
7     else if (data < root->data)
8         root->left = insert(root->left, data);
9     return root;
10 }
11
12 void Node::inOrder(Node* root, vector<int>& arr)
13 {
14     if (!root)
15         return;
16     inOrder(root->left, arr);
17     arr.push_back(root->data);
18     inOrder(root->right, arr);
19 }
20
21 void binarySort(vector<int>& src)
22 {
23     size_t n = src.size();
24     vector<int> res;
25     Node tree;
26     Node* root = nullptr;
27     root = tree.insert(root, src[0]);

```

```
28     for (size_t i = 1; i < n; i++)
29         tree.insert(root, src[i]);
30     tree.inOrder(root, res);
31     src = res;
32 }
```

## 3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы сортировок. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

	Входные данные	Ожидаемый результат		
№	Массив	Бисерная	Поразрядная	Бинарным деревом
1	[ ]	[ ]	[ ]	[ ]
2	[1]	[1]	[1]	[1]
3	[2, 2, 2, 2, 2]	[2, 2, 2, 2, 2]	[2, 2, 2, 2, 2]	[2, 2, 2, 2, 2]
4	[1, 2, 3, 4]	[1, 2, 3, 4]	[1, 2, 3, 4]	[1, 2, 3, 4]
5	[8, 6, 3, 1]	[8, 6, 3, 1]	[8, 6, 3, 1]	[8, 6, 3, 1]
6	[8, 1, 12, 4]	[1, 4, 8, 12]	[1, 4, 8, 12]	[1, 4, 8, 12]

## Вывод

Были представлены листинги реализаций всех алгоритмов сортировок — бисерной, поразрядной и бинарного дерева. Также в данном разделе была приведены функциональные тесты.

## 4 Исследовательская часть

В данном разделе будут приведены демонстрации работы программы, и будет проведен сравнительный анализ реализованных алгоритмов сортировок по времени и по памяти.

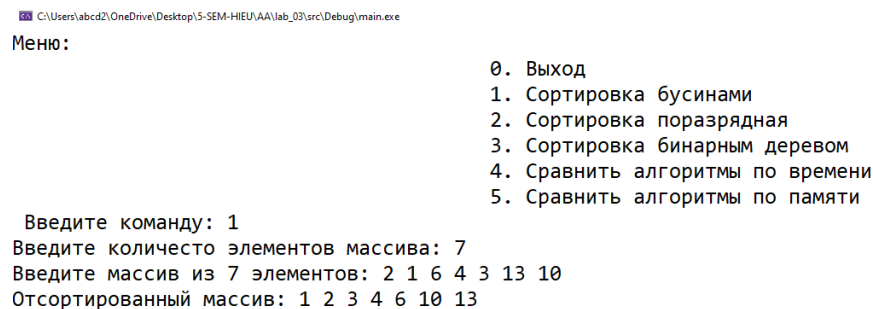
### 4.1 Технические характеристики

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система Window 10 Home Single Language;
- память 8 Гб;
- процессор 11th Gen Intel(R) Core(TM) i7-1165G7 2.80 ГГц, 4 ядра.

### 4.2 Демонстрация работы программы

На рисунках 4.1–4.3 представлен результат работы программы. В каждом примере пользователем введен массив целых чисел и получены результаты его сортировки.



```
C:\Users\abcd2\OneDrive\Desktop\5-SEM-HIEU\AA\lab_03\src\Debug\main.exe
Меню:
0. Выход
1. Сортировка бусинами
2. Сортировка поразрядная
3. Сортировка бинарным деревом
4. Сравнить алгоритмы по времени
5. Сравнить алгоритмы по памяти

Введите команду: 1
Введите количество элементов массива: 7
Введите массив из 7 элементов: 2 1 6 4 3 13 10
Отсортированный массив: 1 2 3 4 6 10 13
```

Рисунок 4.1 – Демонстрация работы программы при бисерной сортировке



Меню:

- 0. Выход
- 1. Сортировка бусинами
- 2. Сортировка поразрядная
- 3. Сортировка бинарным деревом
- 4. Сравнить алгоритмы по времени
- 5. Сравнить алгоритмы по памяти

Введите команду: 2

Введите количество элементов массива: 7

Введите массив из 7 элементов: 2 1 6 4 3 13 10

Отсортированный массив: 1 2 3 4 6 10 13

Рисунок 4.2 – Демонстрация работы программы при поразрядной сортировке

Меню:

- 0. Выход
- 1. Сортировка бусинами
- 2. Сортировка поразрядная
- 3. Сортировка бинарным деревом
- 4. Сравнить алгоритмы по времени
- 5. Сравнить алгоритмы по памяти

Введите команду: 3

Введите количество элементов массива: 7

Введите массив из 7 элементов: 2 1 6 4 3 13 10

Отсортированный массив: 1 2 3 4 6 10 13

Рисунок 4.3 – Демонстрация работы программы при сортировке бинарным деревом

## 4.3 Результаты замера процессорного времени

Результаты эксперимента замеров по времени приведены в таблицах 4.1, 4.3.

В таблице 4.1 приведены результаты замеров по времени алгоритмов сортировок неотсортированных массивов (это условие наступления лучшего случая сортировки бинарным деревом), размер которых варьируется от 1 до 2500 с шагом 250.

Таблица 4.1 – Результаты замеров времени (неотсортированные массивы)

Размер массива	Время, мс		
	Бусинами	Поразрядная	Бинарным деревом
1	0	0	0
250	17.7333	0.133333	0.666667
500	33.5667	0.2	2.2
750	50.6667	0.3	4.8
1000	69	0.433333	7.8
1250	82.2667	0.533333	11.8667
1500	96.2333	0.8	16.3667
1750	112.467	0.733333	21
2000	131.033	0.833333	26.8333
2250	149.2	0.933333	31.7667
2500	164.267	1.13333	39

По таблице 4.1 был построен график, который иллюстрирует зависимость процессорного времени, затраченного реализациями алгоритмов сортировки, от размера массива — рис. 4.4. Исходя из этих данных можно понять, что лучшего всего работает поразрядная сортировка, а бисерная сортировка работает в 5,6 раз хуже, чем алгоритм сортировки бинарным деревом.

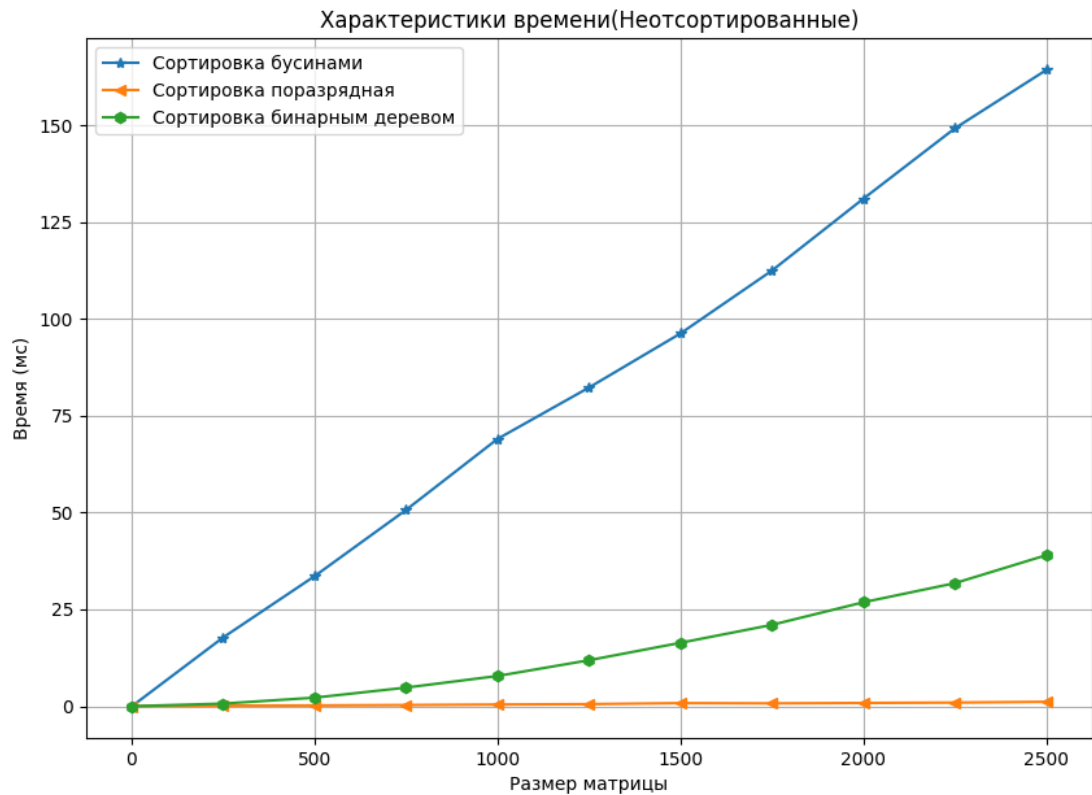


Рисунок 4.4 – Сравнение по времени алгоритмов сортировок на неотсортированных массивах

В таблице 4.2 приведены результаты замеров по времени алгоритмов сортировок отсортированных массивов (это условие наступления худшего случая сортировки бинарным деревом) по возрастанию, размер которых варьируется от 1 до 2500 с шагом 250.

Таблица 4.2 – Результаты замеров времени (отсортированные массивы по возрастанию)

Размер массива	Время, мс		
	Бусинами	Поразрядная	Бинарным деревом
1	0	0	0.0333333
250	17.7333	0.1	0.733333
500	34.9667	0.233333	2.36667
750	50.6667	0.3	5.06667
1000	67.7333	0.4	8.26667
1250	82.2	0.533333	12.2333
1500	96.8	0.633333	17.2667
1750	113.367	0.733333	21.7667
2000	130.833	0.933333	27.7
2250	150.333	0.933333	33.2333
2500	162.067	1.06667	39.1333

По таблице 4.2 был построен график, который иллюстрирует зависимость процессорного времени, затраченного реализациями алгоритмов сортировки, от размера массива — рис. 4.5. Исходя из этих данных можно понять, что лучшего всего работает алгоритм поразрядной сортировки, времени выполнения алгоритма сортировки бинарным деревом увеличивается в 1,2 раза из-за несбалансированного дерева.

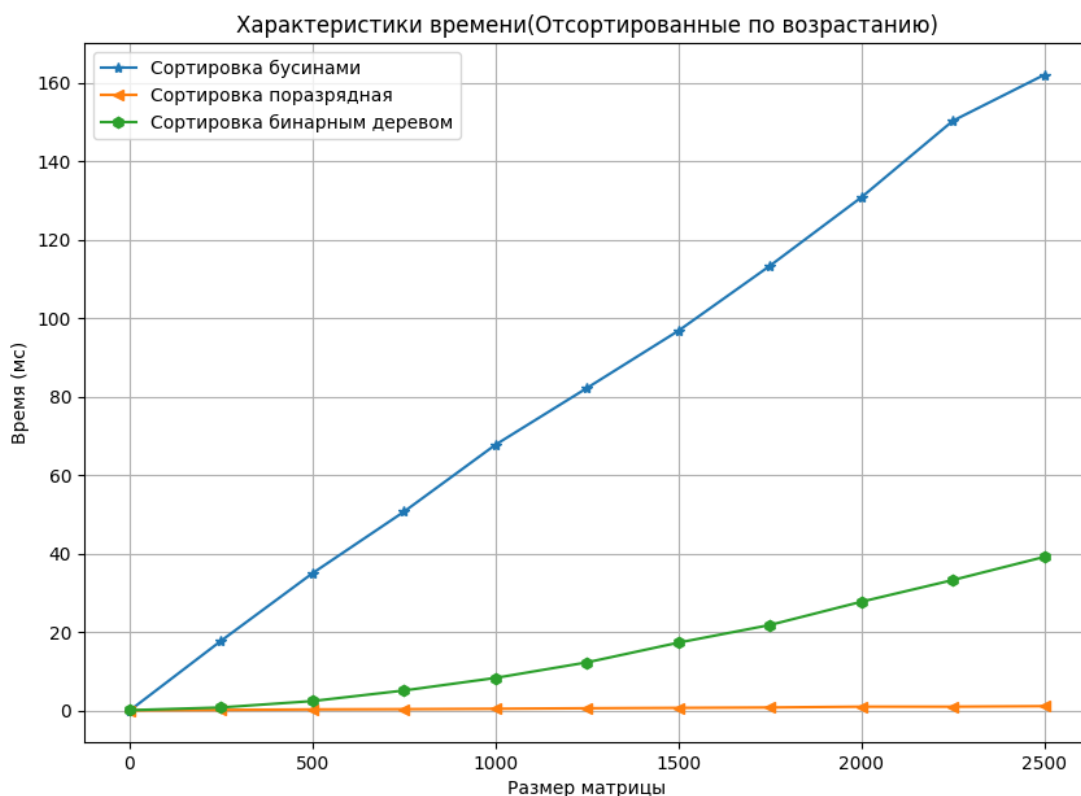


Рисунок 4.5 – Сравнение по времени алгоритмов сортировок на отсортированных массивах по возрастанию

В таблице 4.3 приведены результаты замеров по времени алгоритмов сортировок отсортированных массивов (это условие наступления худшего случая сортировки бинарным деревом) по возрастанию, размер которых варьируется от 1 до 2500 с шагом 250.

Таблица 4.3 – Результаты замеров времени (отсортированные массивы по убыванию)

Размер массива	Время, мс		
	Бусинами	Поразрядная	Бинарным деревом
1	0.0333333	0	0
250	18.0667	0.1	0.666667
500	32.9667	0.3	2.33333
750	50.3333	0.333333	5.06667
1000	68.5	0.433333	8.2
1250	83.4667	0.5	12.1333
1500	97.6667	0.633333	17.2
1750	114.533	0.7	21.6333
2000	130.133	0.833333	27.2
2250	149.067	0.933333	31.9667
2500	162.867	1.03333	39.3333

По таблице 4.3 был построен график, который иллюстрирует зависимость процессорного времени, затраченного реализациями алгоритмов сортировки, от размера массива — рис. 4.6. Исходя из этих данных можно понять, что лучшего всего работает поразрядная сортировка, самая медленная — сортировка бисунами.

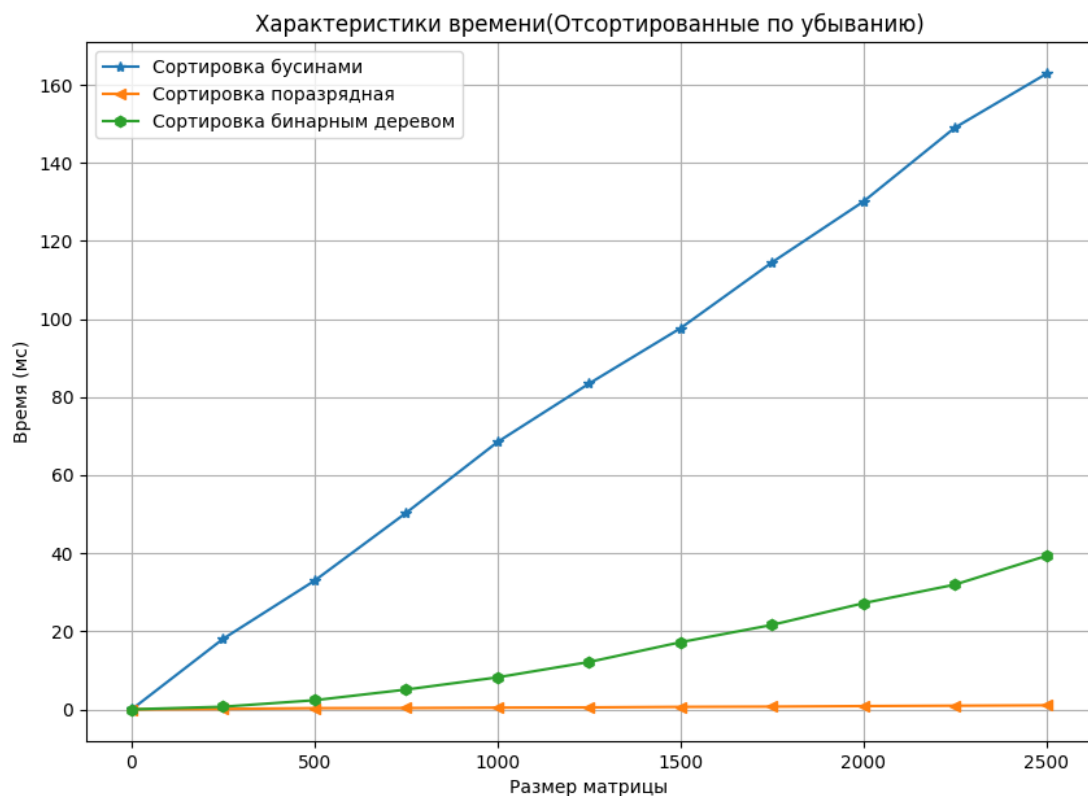


Рисунок 4.6 – Сравнение по времени алгоритмов сортировок на отсортированных массивах по убыванию

## 4.4 Характеристики по памяти

Пусть  $N$  — количество элементов массива,  $M$  — максимальное значение массива, тогда затраты памяти на рассматриваемые алгоритмы будут следующими.

Бисерная сортировка:

— размер массива —  $1 \cdot \text{sizeof}(\text{size\_t});$

- вспомогательные переменные —  $3 \cdot \text{sizeof}(\text{int})$ ;
- вспомогательные массивы —  $(N + M) \cdot \text{sizeof}(\text{int})$ .

Таким образом, общая затраченная память в бисерной сортировке равна  $1 \cdot 8 + 3 \cdot 4 + (N + M) \cdot 4 = 4M + 4N + 20$ .

Поразрядная сортировка:

- размер массива —  $1 \cdot \text{sizeof}(\text{size\_t})$ ;
- вспомогательные переменные —  $2 \cdot \text{sizeof}(\text{int})$ ;
- вспомогательные массивы —  $(N + 10) \cdot \text{sizeof}(\text{int})$ .

Таким образом, общая затраченная память в поразрядной сортировке равна  $1 \cdot 8 + 2 \cdot 4 + (N + 10) \cdot 4 = 4N + 56$ .

Сортировка бинарным деревом:

- размер массива —  $1 \cdot \text{sizeof}(\text{size\_t})$ ;
- размер подов —  $\log(N) \cdot \text{sizeof}(\text{Node})$ ;
- вспомогательный массив —  $N \cdot \text{sizeof}(\text{int})$ .

Таким образом, общая затраченная память в сортировке бинарным деревом равна  $1 \cdot 8 + \log(N) \cdot 12 + N \cdot 4 = 4N + 12 \log(N) + 8$ .

Результаты расчетов памяти, которых представлены в таблице 4.4, где размеры массивов находятся в диапазоне от 1 до 2500 с шагом 250.



Таблица 4.4 – Результаты замеров памяти

Размер массива	Память, кб		
	Бусинами	Поразрядная	Бинарным деревом
1	0.184	0.056	0.008
250	20.972	1.052	1.099
500	21.92	2.052	2.111
750	23	3.052	3.118
1000	24.008	4.052	4.123
1250	25.008	5.052	5.127
1500	26.012	6.052	6.13
1750	27.012	7.052	7.133
2000	27.984	8.052	8.135
2250	28.996	9.052	9.137
2500	30.012	10.052	10.139

По таблице 4.4 был построен график, который иллюстрирует зависимость памяти, затраченной реализациями алгоритмов сортировки, от размера массива — рис. 4.7. Исходя из этих данных можно понять, что бисерная сортировка потребляет больше всего памяти, также алгоритм поразрядной сортировки и алгоритм сортировки бинарным деревом используют примерно одинаковое количество памяти.

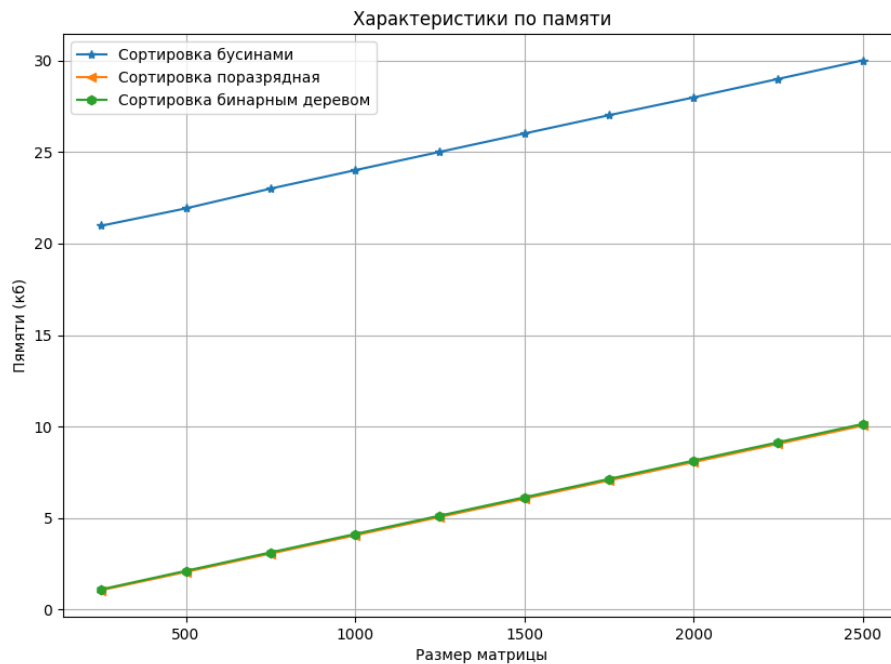


Рисунок 4.7 – Сравнение по памяти алгоритмов сортировок

## 4.5 Вывод

Характер зависимости процессорного времени выполнения реализаций алгоритмов сортировки от размера массива совпал с рассчитанными функциями трудоёмкости. В результате эксперимента было получено, что при размере массива больше 500 поразрядная сортировка работает быстрее сортировки бинарным деревом в 20 раз. При этом бисерная сортировка работает медленнее сортировки бинарным деревом в 8.9 раз. В случае, если массив уже отсортирован, время выполнения сортировки бинарным деревом в 1.3 раза увеличивается из-за несбалансированного дерева.

При размере массива больше 500 бисерная сортировка использует памяти в 4,5 раза больше остальных. При этом поразрядная сортировка и сортировка бинарным деревом используют примерно одинаковое количество памяти.

# Заключение

В результате исследования было получено, что при размере массива больше 500, поразрядная сортировка работает быстрее сортировки бинарным деревом в 20 раз. При этом бисерная сортировка работает медленнее сортировки бинарным деревом в 8,9 раз. В случае, если массив уже отсортирован, время выполнения сортировки бинарным деревом в 1.3 раза увеличивается из-за несбалансированного дерева.

При размере массива больше 500 бисерная сортировка использует памяти в 4,5 раза больше остальных. При этом поразрядная сортировка и сортировка бинарным деревом используют примерно одинаковое количество памяти. Можно сделать вывод, что поразрядная сортировка предпочтительна использовать для сортировки массива.

Цель, которая была поставлена в начале лабораторной работы, была достигнута. В ходе лабораторной работы решены все задачи:

- описаны алгоритмы бисерной сортировки, поразрядной сортировки, сортировки бинарным деревом;
- построены схемы рассматриваемых алгоритмов;
- создано программное обеспечение, реализующее перечисленные алгоритмы;
- проведен сравнительный анализ реализованных алгоритмов по времени и по памяти;
- подготовлен отчет о выполненной лабораторной работе.

## Список использованных источников

1. Вшивков В. А., Маркелова Т. В., Шелехов В. И. Об алгоритмах сортировки в методе частиц в ячейках // Системы анализа и обработки данных. – 2008. – №. 4. – С. 79–92.
2. Кириллов А. С., Кунурбаев К. А., Бичурин Е. А. Исследование способов сортировки значений данных // Автоматизированные технологии и производства. – 2019. – №. 2. – С. 4–8.
3. Тонких А. П. Быстрая поразрядная сортировка на графических процессорах // Информационные технологии в моделировании и управлении: подходы, методы, решения. – 2022. – С. 184–192.
4. Сахибназарова В. Б., Сайгак К. О. Разработка и анализ алгоритма сортировки на основе использования бинарных деревьев // Студенческая наука XXI века. – 2016. – №. 4. – С. 151–152.
5. C library function clock() [Электронный ресурс]. Режим доступа: <https://en.cppreference.com/w/cpp/chrono/c/clock>
6. Matplotlib documentation [Электронный ресурс]. Режим доступа: <https://matplotlib.org/stable/index.html> (дата обращения: 07.11.2023)