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 Трудоемкость сортировок

Студент Морозов Д.В.

Группа ИУ7-52Б

Оценка (баллы)

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

Москва — 2022 г.

Оглавление

Введение

В современном мире приходится работать с большими массивами данных. Часто возникает необходимость в их сортировке — представлении данных в порядке увеличения (или уменьшения) некоторой функции от их значения.

Целью данной лабораторной работы является описание и реализация алгоритмов гномьей сортировки, сортировки перемешиванием и двоичным деревом.

Задачи данной лабораторной:

1) описание алгоритмов гномьей сортировки, сортировки перемешиванием и двоичным деревом;
2) реализация алгоритмов;
3) оценка трудоёмкости алгоритмов;
4) сравнение затрат реализаций алгоритмов по времени выполнения.

1 Аналитическая часть

В этом разделе представлены описания алгоритмов.

1.1 Гномья сортировка

В начеле алгоритма создаётся указатель на второй элемент массива. После этого происходит сравнение текущего и предыдущего элементов. Если порядок соблюден, происходит переход к следующему элементу, если нет, то элементы меняются местами и указатель в цикле переходит к предыдущему элементу. Цикл сортировки заканчивается в тот момент, когда номер указателя становится равным длине массива [1].

1.2 Сортировка перемешиванием

По данному алгоритму в каждой итерации цикла указатель перемещается от левого края массива до правого и при нарушении порядка элементов они меняются местами. После прохода слева направо указатель возвращается к началу массива, так же при нарушении порядка элементов меняя их [1].

1.3 Сортировка двоичным деревом

Сортировка с помощью двоичного дерева — универсальный алгоритм сортировки, заключающийся в построении двоичного дерева поиска по ключам массива (списка), с последующей сборкой результирующего массива путём обхода узлов построенного дерева в необходимом порядке следования ключей. То есть, если при обходе дерева записывать все встречающиеся элементы в массив, получится упорядоченное в порядке возрастания множество [2].

2 Конструкторская часть

В этом разделе приведены схемы алгоритмов.

2.1 Гномья сортировка

На рисунке 2.1 приведена схема алгоритма гномьей сортировки.

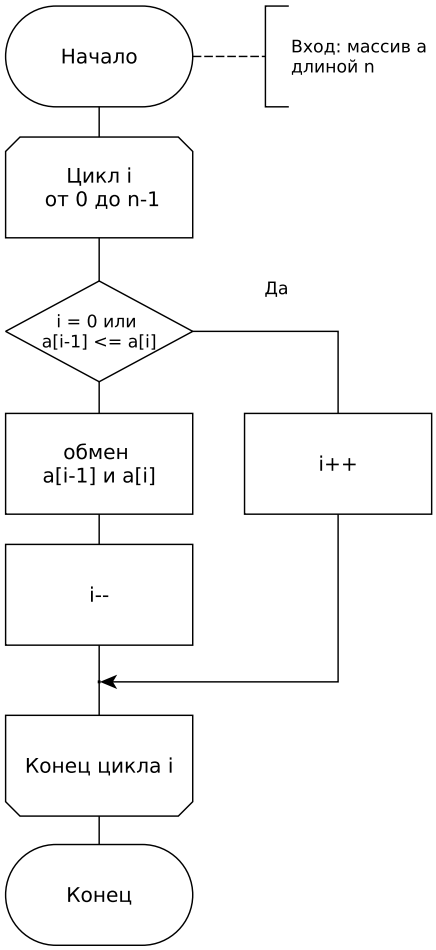

Рисунок 2.1 – Схема алгоритма гномьей сортировки

2.2 Сортировка перемешиванием

На рисунке 2.2 приведена схема алгоритма сортировки перемешиванием.

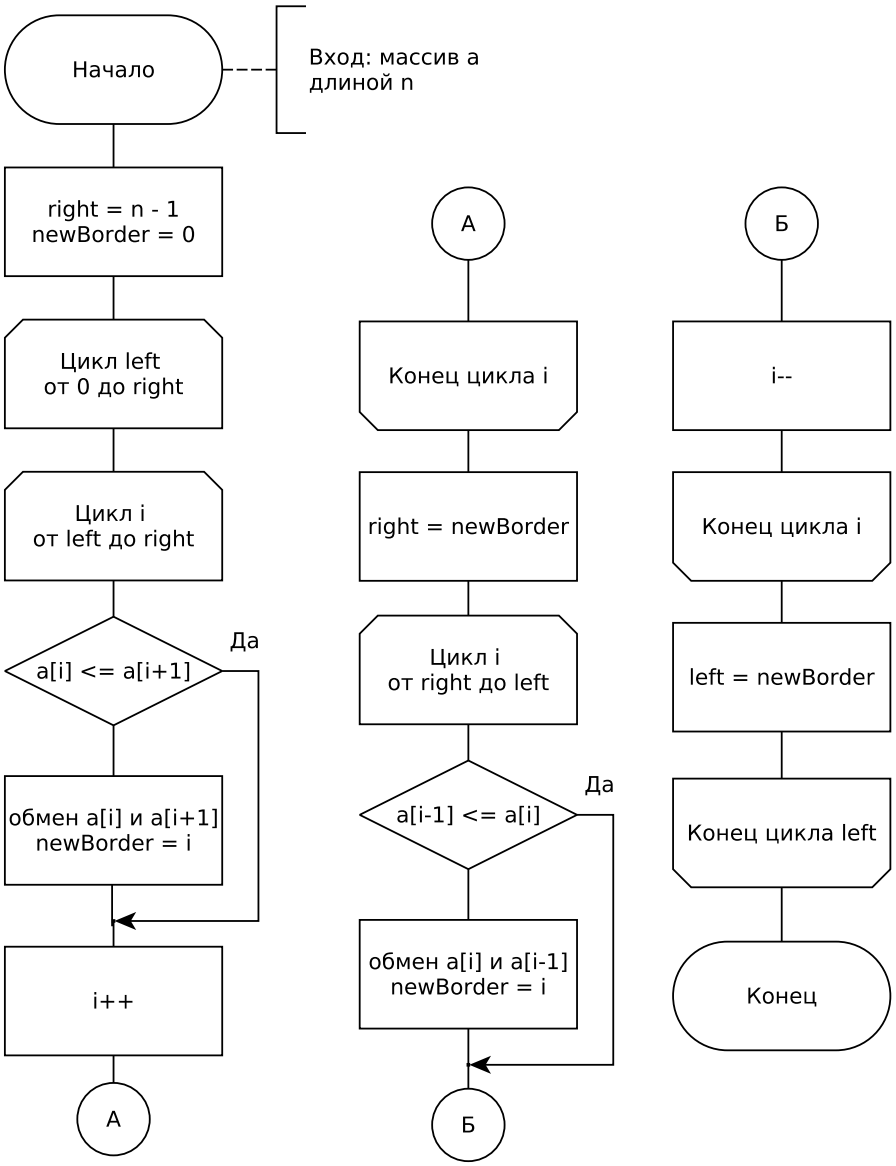

Рисунок 2.2 – Схема алгоритма сортировки перемешиванием

2.3 Сортировка двоичным деревом

На рисунке 2.3 приведена схема алгоритма сортировки двоичным деревом.

Рисунок 2.3 – Схема алгоритма сортировки двоичным деревом

На рисунке 2.4 приведена схема алгоритма вставки в двоичное дерево.

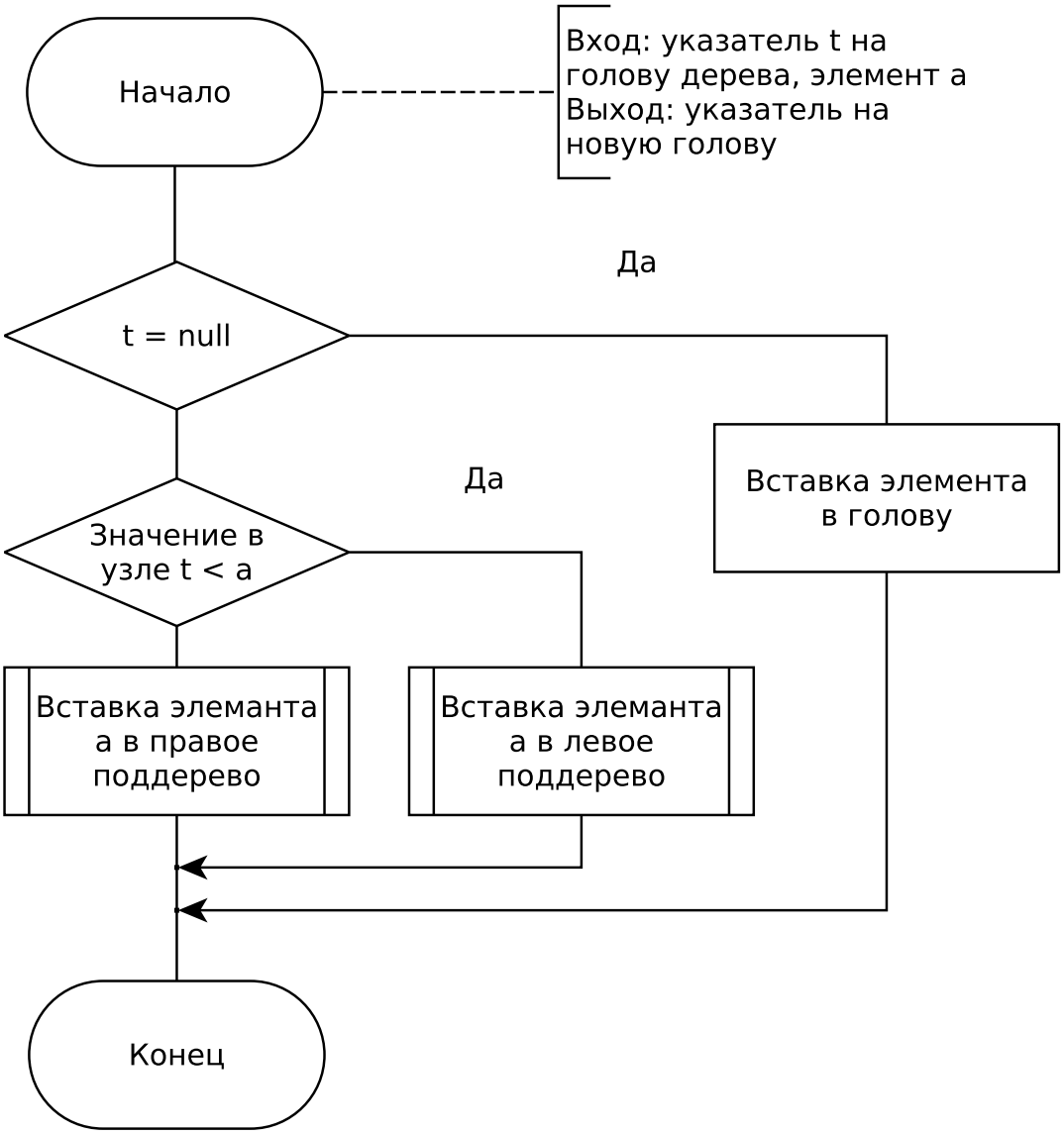

Рисунок 2.4 – Схема алгоритма вставки в двоичное дерево

3 Технологическая часть

В данном разделе представлены требования к программному обеспечению, средства реализации, листинги кода и проведена оценка трудоёмкости.

3.1 Требования к программному обеспечению

Используемое программное обеспечение должно предоставлять возможность измерения процессорного времени.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык программирования C++ [3] и среда разработки CLion, которая позволяет замерять процессорное время с помощью пакета `<ctime>` [4].

3.3 Реализация алгоритмов

В листингах 3.1–3.3 приведены реализации алгоритмов, в листингах 3.4–3.5 — функции вставки в дерево и его обхода.

Листинг 3.1 – Функция сортировки перемешиванием

```
void shakerSort(std::vector<int> &a) {
    int left = 0;
    int right = a.size() - 1;
    int newBorder = left;

    while (right > left) {
        for (int i = left; i < right; i++) {
            if (a[i] > a[i+1]) {
                int tmp = a[i];
                a[i] = a[i+1];
                a[i+1] = tmp;
                newBorder = i;
            }
        }
        right = newBorder;

        for (int i = right; i > left; i--) {
            if (a[i-1] > a[i]) {
                int tmp = a[i];
                a[i] = a[i-1];
                a[i-1] = tmp;
                newBorder = i;
            }
        }
        left = newBorder;
    }
}
```

Листинг 3.2 – Функция гномьей сортировки

```
void gnomeSort(std::vector<int> &a) {
    int n =  a.size();

    for (int i = 0; i < n;) {
        if (i == 0 || a[i-1] <= a[i]) {
            i++;
        } else {
            int tmp = a[i];
            a[i] = a[i-1];
            a[i-1] = tmp;
            i--;
        }
    }
}
```

Листинг 3.3 – Функция сортировки двоичным деревом

```
void treeSort(std::vector<int> &arr) {
    auto tree = std::make_unique<BSTree>();

    for (int &el : arr) {
        tree->insert(el);
    }

    tree->traverse(arr);
}
```

Листинг 3.4 – Функция вставки в дерево

```
BinTreeNode* insert(BinTreeNode *node, int data) {
    if (node == nullptr) {
        node = new BinTreeNode{data, nullptr, nullptr};
    } else if (data < node->data) {
        node->left = insert(node->left, data);
    } else {
        node->right = insert(node->right, data);
    }
    return node;
}
```

Листинг 3.5 – Функция обхода дерева

```
void traverse(BinTreeNode *node, std::vector<int> &arr, int
   &index) {
    if (node->left != nullptr) {
        traverse(node->left,arr, index);
    }

    arr[index] = node->data;
    index++;

    if (node->right != nullptr) {
        traverse(node->right,arr, index);
    }
}
```

3.4 Оценка трудоемкости алгоритмов

3.4.1 Модель вычислений

Для оценки трудоёмкости алгоритмов введём модель вычислений.

1) Операции из списка (3.1) имеют трудоемкость 1.

$$+, -, ++, --, *, /, \%, ==, !=, <, >, <=, >=, [] \tag{3.1}$$

2) Трудоемкость оператора выбора if условие then A else B рассчитывается, как (3.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется;} \\ f_B, & \text{иначе.} \end{cases} \tag{3.2}$$

3) Трудоемкость цикла с N итерациями рассчитывается, как (3.3).

$$\begin{aligned} f_{\text{цикла}} &= f_{\text{инициализации}} + f_{\text{сравнения}} + \\ &+ N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \end{aligned} \tag{3.3}$$

4) Трудоемкость вызова функции равна 0.

3.4.2 Сортировка перемешиванием

Рассчитаем трудоёмкость для худшего случая — массив отсортирован в обратном порядке.

1) Трудоёмкость условного оператора:

$$f_{if} = \underbrace{4}_{cmp} + \underbrace{2+4+3}_{swap} + \underbrace{1}_{newborder} + \underbrace{2}_{inc} + \underbrace{2}_{cmp} = 18. \tag{3.4}$$

2) Трудоёмкость внешнего цикла:

$$f_{out} = \underbrace{4}_{init} + \frac{N}{2} \cdot (\underbrace{1}_{cmp} + \underbrace{2}_{inc} + (N-1) \cdot (\underbrace{2}_{init} + \underbrace{2}_{cmp})). \tag{3.5}$$

3) Общая трудоёмкость в худшем случае:

$$f_{worst} = 4 - 8 \cdot N + 11 \cdot N^2. \tag{3.6}$$

Трудоёмкость алгоритма сортировки пермешиванием в худшем случае оценена как $O(N^2)$.

Рассчитаем трудоёмкость для лучшего случая — массив уже отсортирован.

1) Трудоёмкость условного оператора:

$$f_{if} = \underbrace{4}_{cmp}. \tag{3.7}$$

2) Трудоёмкость первого внутреннего цикла:

$$f_{in1} = \underbrace{2}_{init} + N \cdot (\underbrace{1}_{cmp} + \underbrace{1}_{inc} + f_{if}). \tag{3.8}$$

3) Трудоёмкость второго внутреннего цикла:

$$f_{in2} = \underbrace{2}_{init}. \tag{3.9}$$

4) Трудоёмкость внешнего цикла:

$$f_{out} = \underbrace{4}_{init} + \underbrace{1}_{cmp} + \underbrace{2}_{inc} + f_{in1} + f_{in2}. \tag{3.10}$$

5) Общая трудоёмкость в лучшем случае:

$$f_{best} = 6 \cdot N + 11 \approx 6 \cdot N. \tag{3.11}$$

3.4.3 Гномья сортировка

Рассчитаем трудоёмкость для худшего случая — массив отсортирован в обратном порядке.

1) Трудоёмкость проверки условия:

$$f_{cmp} = 6. \tag{3.12}$$

2) Трудоёмкость условного оператора при истинном условии:

$$f_{if} = f_{cmp} + 1 = 7. \tag{3.13}$$

3) Трудоёмкость условного оператора при ложном условии:

$$f_{else} = f_{cmp} + 10 = 16. \tag{3.14}$$

4) Трудоёмкость цикла:

$$f_{out} = \underbrace{2}_{init} + \frac{N-1}{2} \cdot N \cdot (\underbrace{1}_{cmp} + f_{if} + f_{else}) + N \cdot f_{if}. \tag{3.15}$$

5) Общая трудоёмкость в худшем случае:

$$f_{worst} = 12 \cdot N^2 - 5 \cdot N + 2 \approx 12 \cdot N^2. \tag{3.16}$$

Рассчитаем трудоёмкость для лучшего случая — массив уже отсортирован.

1) Трудоёмкость условного оператора:

$$f_{if} = 7. \tag{3.17}$$

2) Общая трудоёмкость в лучшем случае:

$$f_{best} = N \cdot f_{if} = 7 \cdot N. \tag{3.18}$$

3.4.4 Сортировка двоичным деревом

Трудоёмкость этого алгоритма в лучшем случае, когда вставка каждого элемента производится в сбалансированное дерево, пропорциональна $N \cdot \log N$, в худшем случае — для невозрастающей или неубывающей последовательности — N^2 [2].

4 Исследовательская часть

В данном разделе произведено сравнение алгоритмов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система — Ubuntu 22.04.1 Linux x86_64;

— оперативная память — 8 ГБ;

— процессор — AMD Ryzen 5 3550H [5].

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время замеров ноутбук не был нагружен сторонними приложениями.

4.2 Время выполнения алгоритмов

На рисунке 4.1 представлен график, иллюстрирующий зависимость времени работы алгоритмов от размерности массивов при заполнении их случайными данными.

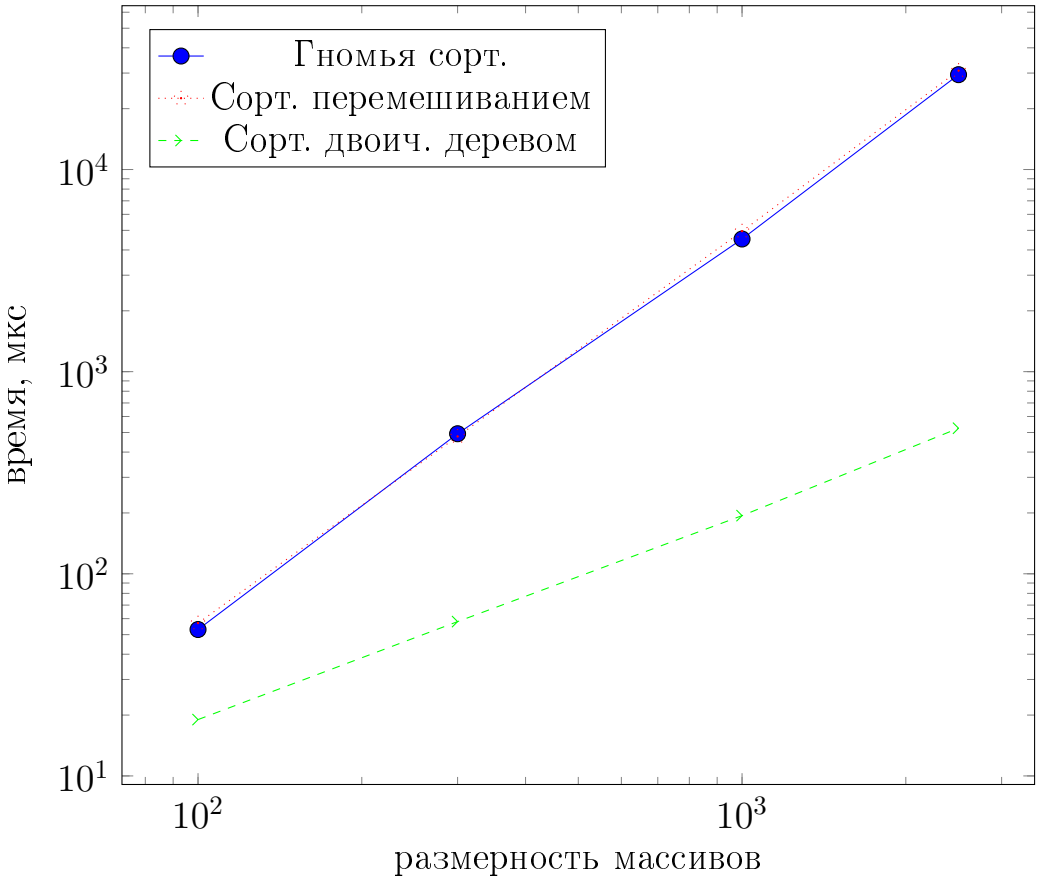

Рисунок 4.1 – Сравнение алгоритмов при случайных данных

На рисунке 4.2 представлен график, иллюстрирующий зависимость времени работы алгоритмов от размерности массивов для лучшего случая.

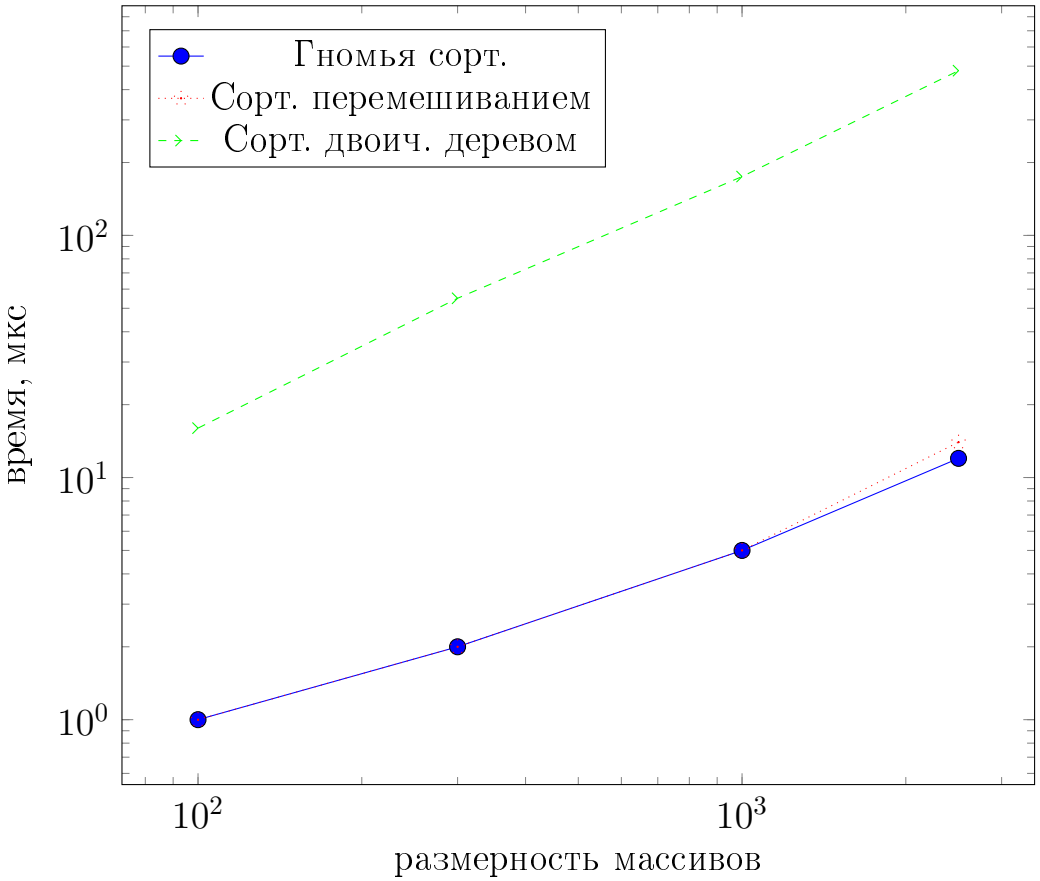

Рисунок 4.2 – Сравнение алгоритмов для лучшего случая

На рисунке 4.3 представлен график, иллюстрирующий зависимость времени работы алгоритмов от размерности массивов для худшего случая.

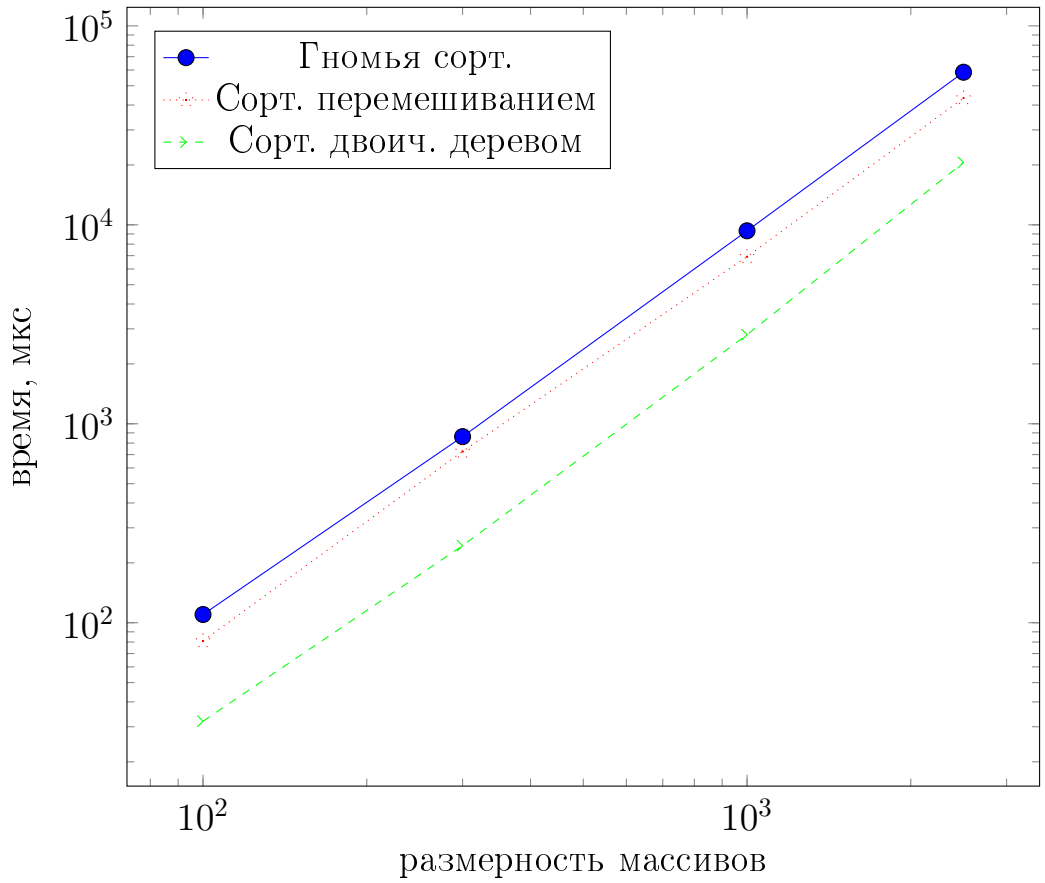

Рисунок 4.3 – Сравнение алгоритмов для худшего случая

Вывод

При заполнении массива случайными данными реализация алгоритма сортировки перемешиванием незначительно отличается по времени работы от реализации алгоритма гномьей сортировки: для всех размерностей разница не более 5 %. Сортировка двоичным деревом оказалась эффективнее других методов: при размерности 100 её реализация быстрее в 3 раза, при размерности 2500 быстрее в 57 раз.

В лучшем случае реализации алгоритмов сортировки перемешиванием и гномьей сортировки также схожи по времени работы, так как не выполняют обмена элементов; реализации этих алгоритмов при размерности 2500 оказались быстрее реализации сортировки двоичным деревом в 40 раз.

В худшем случае реализация сортировки перемешиванием быстрее гномьей на 25 %, но меделеннее реализации сортировки двоичным деревом приблизительно в 2 раза.

Заключение

Цель достигнута: были описаны и реализованы алгоритмы гномьей сортировки, сортировки перемешиванием и двоичным деревом. В ходе выполнения лабораторной работы были решены все задачи:

— описаны алгоритмы гномьей сортировки, сортировки перемешиванием и двоичным деревом;

— приведены реализации алгоритмов;

— оценена трудоёмкость алгоритмов;

— проведено сравнение затрат реализаций алгоритмов по времени выполнения.

При заполнении массива случайными данными реализация алгоритма сортировки перемешиванием незначительно отличается по времени работы от реализации алгоритма гномьей сортировки: для всех размерностей разница не более 5 %. Сортировка двоичным деревом оказалась эффективнее других методов: при размерности 100 её реализация быстрее в 3 раза, при размерности 2500 быстрее в 57 раз.

В лучшем случае реализации алгоритмов сортировки перемешиванием и гномьей сортировки также схожи по времени работы, так как не выполняют обмена элементов; реализации этих алгоритмов при размерности 2500 оказались быстрее реализации сортировки двоичным деревом в 40 раз.

В худшем случае реализация сортировки перемешиванием быстрее гномьей на 25 %, но меделеннее реализации сортировки двоичным деревом приблизительно в 2 раза.

Список использованных источников

1. Прохоренко В. В. Погорелов Д. А. Кострицкий А. С. Сравнительный анализ времени выполнения сортировки по алгоритму Шелла, гномьей сортировки и сортировки перемешиванием // Инновационные аспекты развития науки и техники. 2021.

2. Кнут Д. Э. Искусство программирования. –М.:Мир, 1978. Т. 3. С. 651–675.

3. Документация по языку C++ | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-150` (дата обращения: 30.09.2022).

4. <ctime> | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/en-us/cpp/standard-library/ctime?redirectedfrom=MSDN&view=msvc-160` (дата обращения: 30.09.2022).

5. AMD Ryzen™ 5 3550H [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/apu/amd-ryzen-5-3550h` (дата обращения: 30.09.2022).