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»

на тему: «Трудоёмкость сортировок»

Студент ИУ7-54Б _____ Булдаков М.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.
(Подпись, дата) (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сортировка данных является фундаментальной задачей в области информатики и алгоритмов. Независимо от конкретной области применения, эффективные алгоритмы сортировки существенно влияют на производительность программных систем. От правильного выбора алгоритма зависит как время выполнения программы, так и затраты ресурсов компьютера [1].

Алгоритмы сортировки находят применение в следующих сферах:

— базы данных;

— анализ данных и статистика;

— алгоритмы машинного обучения;

— криптография.

Цель данной лабораторной работы — рассмотреть алгоритмы сортировки. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать алгоритмы гномьей, пирамидальной сортировок и сортировку по алгоритму Шелла;

— разработать программное обеспечение, реализующее алгоритмы сортировок;

— выбрать инструменты для реализации и замера процессорного времени выполнения реализаций алгоритмов;

— проанализировать затраты реализаций алгоритмов по времени.

1 Аналитический раздел

Сортировкой называют перестановку объектов, при которой они располагаются в порядке возрастания или убывания [1].

В данном разделе будут описаны три алгоритма сортировок: гномья, пирамидальная и Шелла.

1.1 Алгоритм гномьей сортировки

Данный алгоритм можно разделить на следующие шаги [2]:

— сравнить текущий и предыдущий элементы;

— если они в правильном порядке, сделать шаг на один элемент вперед, иначе поменять их местами и сделать шаг на один элемент назад;

— если нет предыдущего элемента, сделать шаг вперед;

— если нет следующего элемента, то закончить.

1.2 Алгоритм пирамидальной сортировки

В основе данного алгоритма лежит принцип работы структуры данных куча [3]. Данный алгоритм можно разделить на следующие шаги:

— создать кучу на основе входного массива;

— повторять следующие шаги до тех пор, пока куча не будет содержать только один элемент:

  1) поменять местами корневой элемент кучи (который является самым большим элементом) с последним элементом кучи;
  2) удалить последний элемент кучи (который теперь находится в правильном положении);
  3) сгруппировать оставшиеся элементы в кучу;

— отсортированный массив получается путем изменения порядка элементов во входном массиве.

1.3 Алгоритм Шелла

Данный алгоритм можно разделить на следующие шаги [4]:

— выбрать некоторый интервал (шаг). Обычно начальный шаг выбирают равным половине длины массива;

— сортировка вставками элементов, расположенных на расстоянии заданного шага друг от друга;

— уменьшение шага вдвое и повтор шага 2. Процесс повторяется до тех пор, пока шаг не станет равным 1;

— сортировка завершается с использованием обычной сортировки вставками (шаг равен 1).

Вывод

В данном разделе были описаны три алгоритма сортировок: гномья, пирамидальная и Шелла.

2 Конструкторский раздел

В этом разделе будет представлено описание используемых типов данных, а также схематические изображения алгоритмов сортировок: гномьей, пирамидальной и Шелла.

2.1 Требования к программному обеспечению

Программа должна поддерживать два режима работы: режим массового замера времени и режим сортировки введенного массива.

Режим массового замера времени должен обладать следующей функциональностью:

— генерировать массивы различного размер для проведения замеров;

— осуществлять массовый замер, используя сгенерированные данные;

— результаты массового замера должны быть представлены в виде таблицы и графика.

К режиму сортировки выдвигается следующий ряд требований:

— возможность работать с массивами разного размера, которые вводит пользователь;

— наличие интерфейса для выбора действий;

— на выходе программы, массив отсортированный тремя алгоритмами по возрастанию.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

— целое число представляет количество элементов в массиве;

— массив целых чисел;

— куча представляется с помощью массива целых чисел.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма гномьей сортировки. На рисунке 2.2 представлена схема алгоритма сортировки Шелла. На рисунке 2.3 представлена схема алгоритма пирамидальной сортировки. На рисунке 2.4 представлена схема алгоритма вспомогательной подпрограммы, строящей кучу.

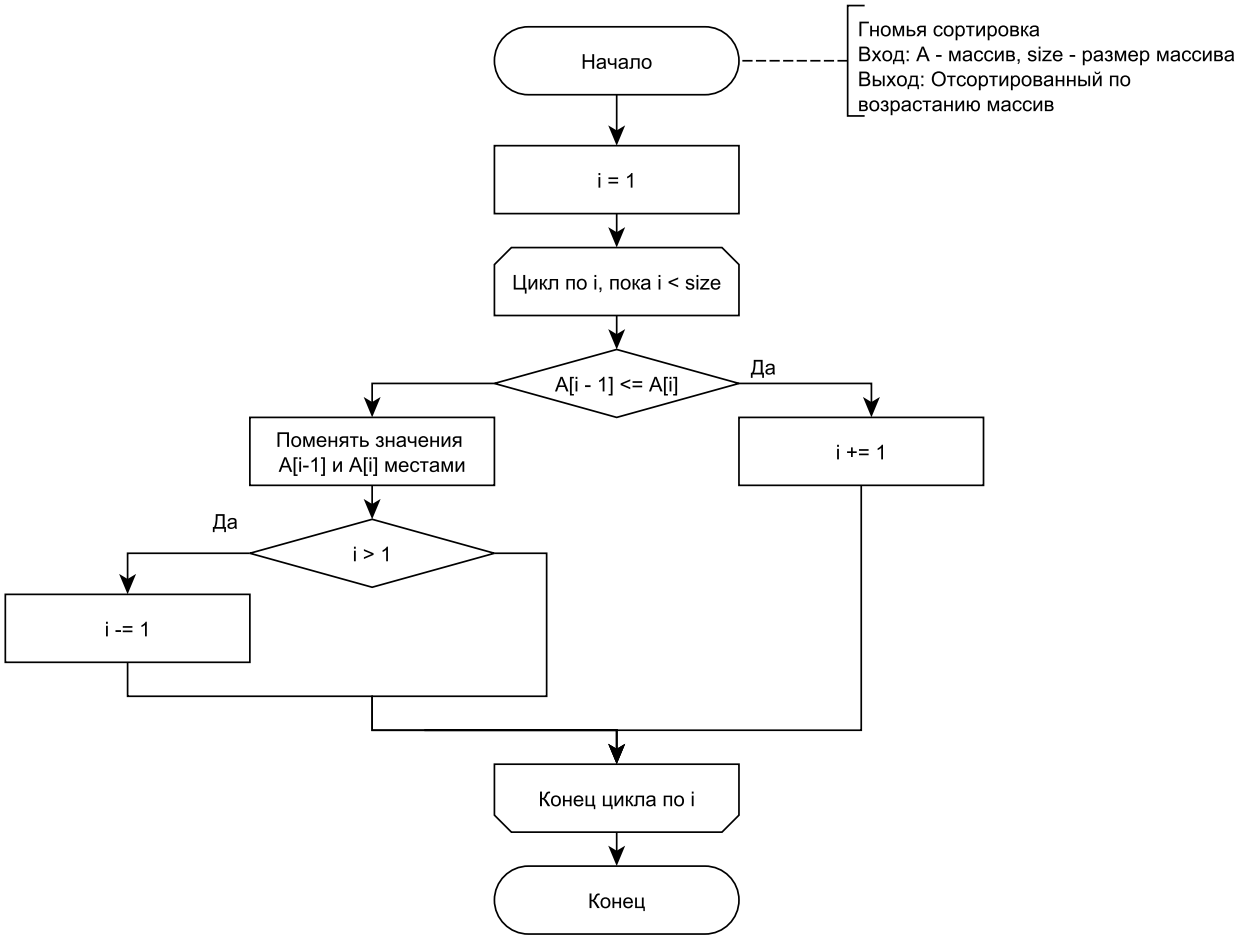

Рисунок 2.1 – Схема алгоритма гномьей сортировки

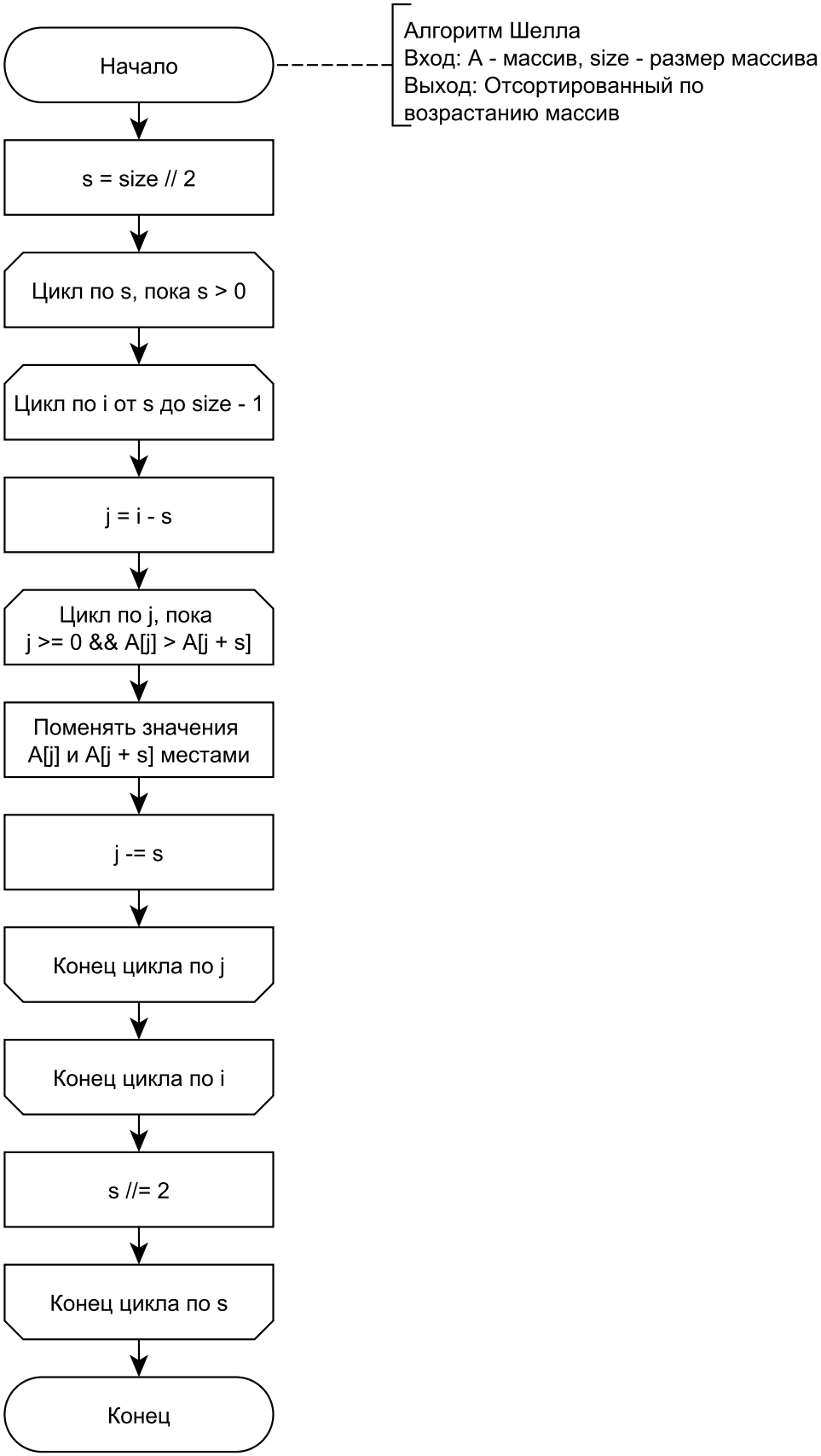

Рисунок 2.2 – Схема алгоритма сортировки Шелла

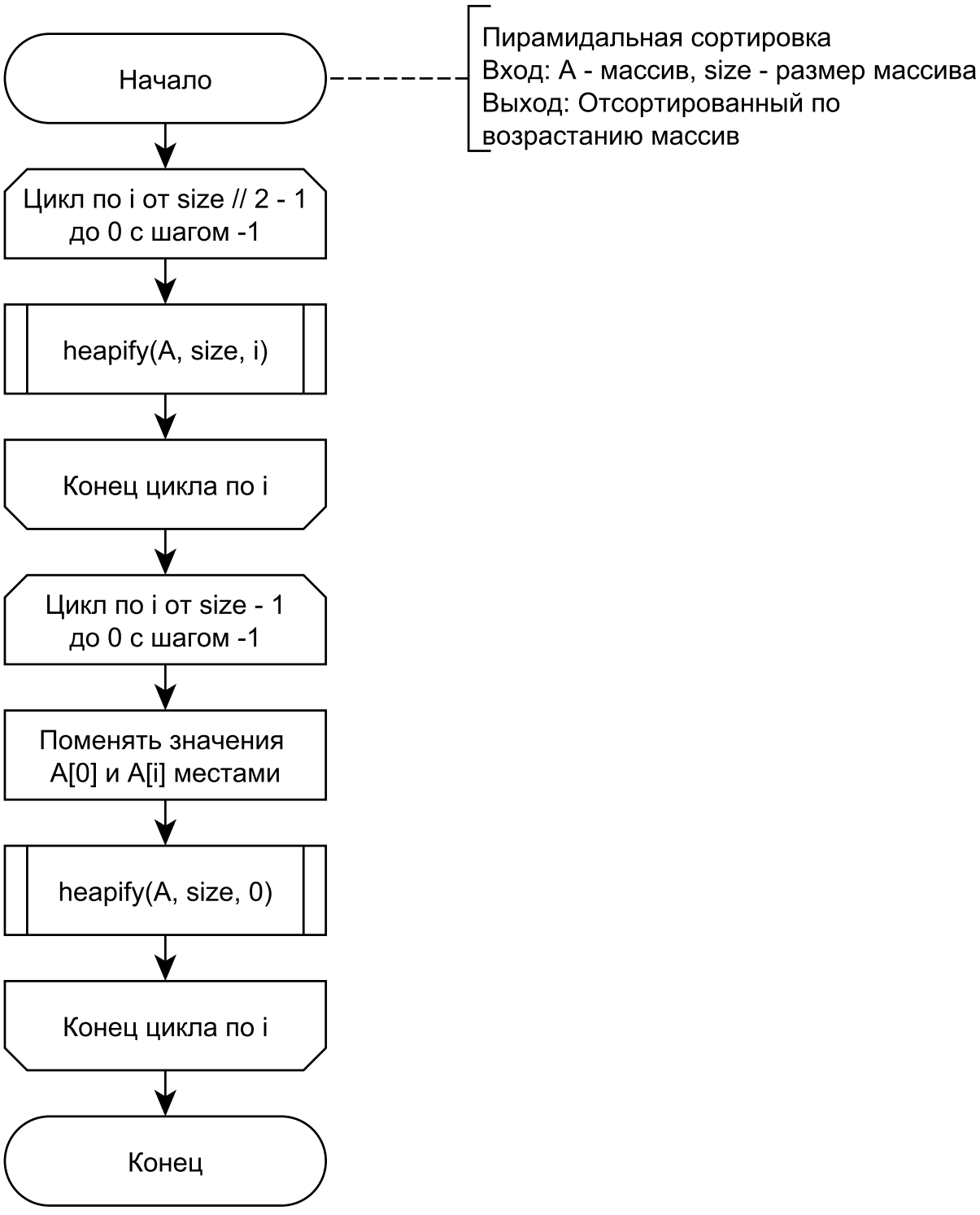

Рисунок 2.3 – Схема алгоритма пирамидальной сортировки

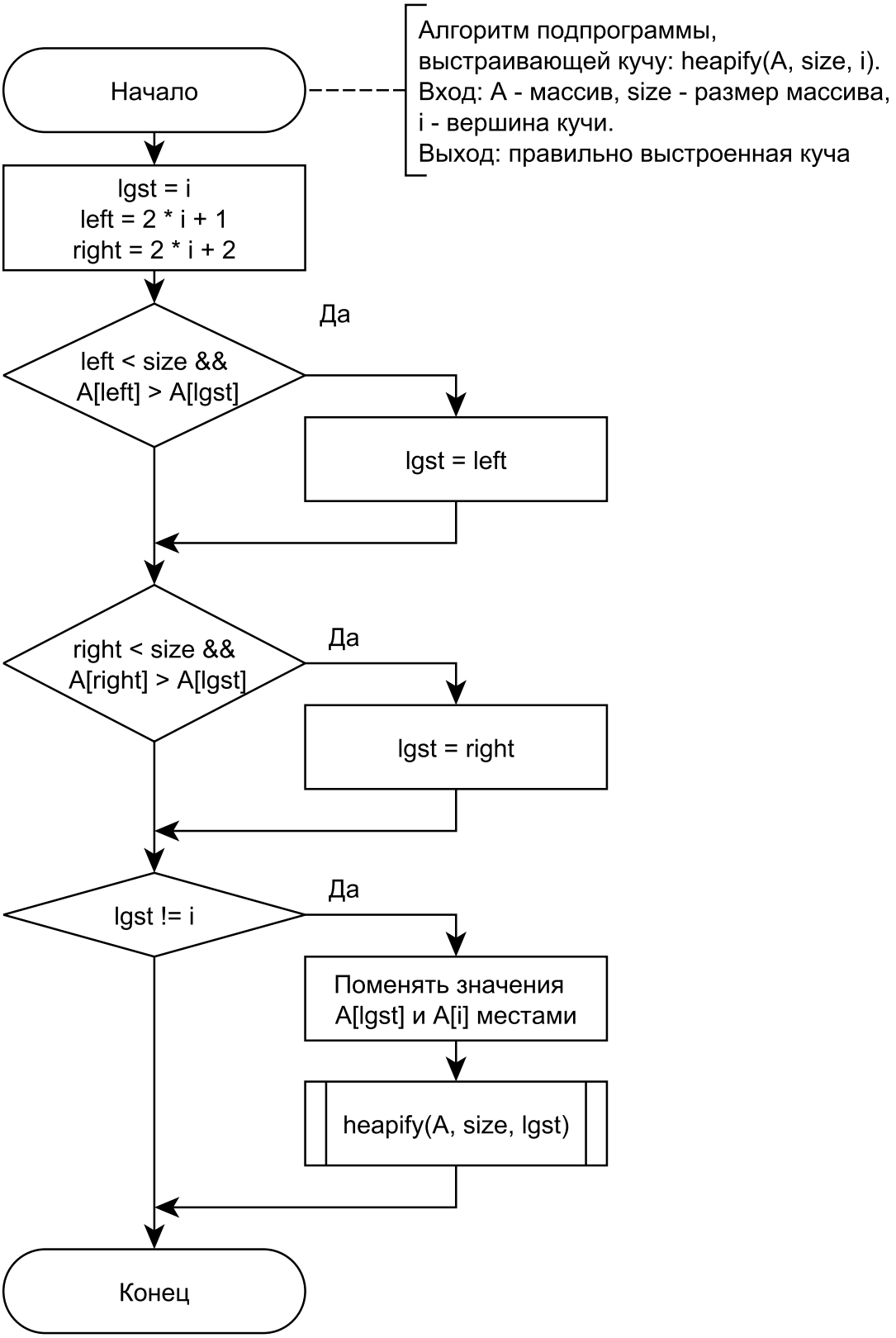

Рисунок 2.4 – Схема алгоритма вспомогательной подпрограммы, выполняющей построение кучи

2.4 Оценка трудоемкости алгоритмов

Модель для оценки трудоемкости алгоритмов состоит из шести пунктов:

1) $+, -, =, + =, - =, ==, ||, \&\&, <, >, <=, >=, <<, >>, []$ — считается, что эти операции обладают трудоемкостью в 1 единицу;

2) $*, /, * =, / =, \%$ — считается, что эти операции обладают трудоемкостью в 2 единицы;

3) трудоемкость условного перехода принимается за 0;

4) трудоемкость условного оператора рассчитывается по формуле (2.1),

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

где f_1 — трудоемкость блока, который вычисляется при выполнении условия, а f_2 — трудоемкость блока, который вычисляется при невыполнении условия;

5) трудоемкость цикла рассчитывается по формуле (2.2),

$$\begin{aligned} f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}); \end{aligned} \tag{2.2}$$

6) вызов подпрограмм и передача параметров принимается за 0.

2.4.1 Трудоемкость алгоритма Шелла

При расчете трудоемкости считается, что $\lfloor \log_2 N \rfloor \approx \log_2 N$. В лучшем случае, когда массив отсортирован трудоемкость алгоритма Шелла рассчитывается по формуле (2.3).

$$f_{best} = 10N \log_2 N - 10N + 5 \log_2 N + 14 \approx O(N \log_2 N) \tag{2.3}$$

В худшем случае, когда переставляются элементы во всех итерациях,

трудоемкость алгоритма Шелла рассчитывается по формуле (2.4).

$$f_{worst} = \frac{32}{3}N^2 + 10N\log_2 N - 26N + 5\log_2 N + \frac{58}{3} \approx O(N^2) \tag{2.4}$$

2.4.2 Трудоемкость алгоритма гномьей сортировки

Для данной сортировки лучшим случаем является отсортированный массив, тогда трудоемкость алгоритма будет рассчитываться по формуле (2.5).

$$f_{best} = 1 + 1 + (N-1)(1+1+4+1) = 7N - 5 \approx O(N) \tag{2.5}$$

Для данного алгоритма худшим является случай, когда массив отсортирован в обратном порядке, тогда трудоемкость рассчитывается по (2.6).

$$\begin{aligned} f_{worst} = 1 + 1 + \frac{(N+1)N}{2}(1+1+4+9+1) + N + \\ + \frac{(N+1)N}{2}(1+1+4+1) = \\ = \frac{23}{2}N^2 + \frac{25}{2}N + 2 \approx O(N^2) \end{aligned} \tag{2.6}$$

2.4.3 Трудоемкость алгоритма пирамидальной сортировки

При расчете трудоемкости считается, что $\lfloor \log_2 N \rfloor \approx \log_2 N$. Трудоемкость подпрограммы **hepify** в лучшем случае, т.е. когда массив упорядочен в виде кучи, равна $f_{hb} = 20$, а в худшем, когда куча не сформирована рассчитывается по формуле (2.7).

$$f_{hw} = (9 + 12 + 8)\log_2 N + 20 = 29\log_2 N + 20 \tag{2.7}$$

Тогда трудоемкость пирамидальной сортировки в лучшем случае, т.е. когда массив упорядочен в виде кучи, рассчитывается по формуле (2.8).

$$\begin{aligned} f_{best} = 4 + 1 + \frac{N}{2}(1+1+20) + 2 + 1 + N(1+1+7+29\log_2 N + 20) = \\ = 29N\log_2 N + 40N + 8 \approx O(N\log_2 N) \end{aligned} \tag{2.8}$$

В худшем случае трудоемкость пирамидальной сортировки оценивается

по формуле (2.9).

$$
\begin{aligned}
f_{worst} = 4 + 1 + \frac{N}{2}(1 + 1 + 29\log_2 N + 20) + 2 + 1 + \\
+ N(1 + 1 + 7 + 29\log_2 N + 20) = \\
= \frac{87}{2} N \log_2 N + 40N + 8 \approx O(N \log_2 N)
\end{aligned}
\tag{2.9}
$$

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов. Была введена модель оценки трудоемкости алгоритма, были рассчитаны трудоемкости алгоритмов в соответствии с этой моделью.

В результате теоретической оценки трудоемкостей алгоритмов выяснилось, что лучшей асимптотической оценкой в худшем случае обладает пирамидальная сортировка $O(N \log_2 N)$. Алгоритм гномьей сортировки оказывается лучше всех прочих сортировок в лучшем случае (т. е. для отсортированных массивов), обладая асимптотической сложностью $O(N)$, но при худшем случае проигрывает по константе алгоритму Шелла, т. о. асимптотическая сложность в худшем случае у гномьей сортировки $O(\frac{23}{2}N^2)$, а у Шелла $O(\frac{32}{3}N^2)$. В лучшем случае асимптотическая сложность алгоритма Шелла $O(10N \log_2 N)$ меньше, чем асимптотическая сложность алгоритма пирамидальной сортировки $O(29N \log_2 N)$.

3 Технологический раздел

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

Для реализации данной работы был выбран язык *Python* [5]. Данный выбор обусловлен следующим:

— язык поддерживает все структуры данных, которые выбраны в результате проектирования;

— язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;

— язык позволяет замерять процессорное время с помощью модуля *time*.

Процессорное время было замерено с помощью функции *process_time()* из модуля *time* [6].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— *main.py* — файл, содержащий функцию *main*;

— *algorithms.py* — файл, содержащий код реализаций всех алгоритмов сортировок;

— *compare_time.py* — файл, в котором содержатся функции для замера и вывода времени выполнения реализаций алгоритмов.

3.3 Реализация алгоритмов

В листингах 3.1 – 3.3 приведены реализации сортировок Шелла, гномьей и пирамидальной соответственно.

Листинг 3.1 – Функция сортировки массива по алгоритму Шелла

```
def shell_sort(arr):
    size = len(arr)
    step = size // 2
    while step > 0:
        for i in range(step, size, 1):
            j = i
            delta = j - step
            while delta >= 0 and arr[delta] > arr[j]:
                arr[delta], arr[j] = arr[j], arr[delta]
                j = delta
                delta = j - step
        step //= 2
    return arr
```

Листинг 3.2 – Функция гномьей сортировки массива

```
def gnome_sort(arr):
    size = len(arr)
    i = 0
    while i < size:
        if i == 0 or arr[i] >= arr[i - 1]:
            i += 1
        else:
            arr[i], arr[i - 1] = arr[i - 1], arr[i]
            i -= 1

    return arr
```

Листинг 3.3 – Реализация пирамидальной сортировки массива

```
def heapify(arr, N, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < N and arr[largest] < arr[l]:
        largest = l

    if r < N and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, N, largest)


def heap_sort(arr):
    N = len(arr)

    for i in range(N // 2 - 1, -1, -1):
        heapify(arr, N, i)

    for i in range(N - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

    return arr
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов сортировок. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Массив	**Ожидаемый результат**	**Фактический результат**
[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]
[ ]	[ ]	[ ]
[1]	[1]	[1]
[4, 1, 2, 3]	[1, 2, 3, 4]	[1, 2, 3, 4]
[2, 1]	[1, 2]	[1, 2]
[31, 57, 24, -10, 59]	[-10, 24, 31, 57, 59]	[-10, 24, 31, 57, 59]

Вывод

Были разработаны и протестированы спроектированные алгоритмы сортировок: Шелла, гномья и пирамидальная.

4 Исследовательский раздел

В данном разделе будут приведены: пример работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного программного обеспечения, а именно показаны результаты сортировки массива $[1, 6, 3, 2, 1, 3, 4]$.

```
Меню:

1 - Отсортировать по возрастанию введенный массив всеми алгоритмами
2 - Провести замерный эксперимент
3 - Вывести функциональные тесты
0 - Выйти

Выберите пункт меню:

1
Вводит целые числа через пробел:
1 6 3 2 1 3 4
--------------------
Исходный массив:  [1, 6, 3, 2, 1, 3, 4]
Результат сортировки алгоритмом Шелла:  [1, 1, 2, 3, 3, 4, 6]
--------------------
Исходный массив:  [1, 6, 3, 2, 1, 3, 4]
Результат гномьей сортировки:  [1, 1, 2, 3, 3, 4, 6]
--------------------
Исходный массив:  [1, 6, 3, 2, 1, 3, 4]
Результат пирамидальной сортировки:  [1, 1, 2, 3, 3, 4, 6]
--------------------

Меню:

1 - Отсортировать по возрастанию введенный массив всеми алгоритмами
2 - Провести замерный эксперимент
3 - Вывести функциональные тесты
0 - Выйти

Выберите пункт меню:
```

Рисунок 4.1 – Демонстрация работы программы при сортировке массива

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, следующие:

— процессор: AMD Ryzen 5 4600H 3 ГГц [7];

— оперативная память: 16 ГБайт;

— операционная система: Windows 10 Pro 64-разрядная система версии 22H2 [8].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.3 Время выполнения реализаций алгоритмов

Результаты замеров времени выполнения реализаций алгоритмов сортировок приведены в таблицах 4.1 – 4.3. Замеры времени проводились на массивах одного размера и усреднялись для каждого набора одинаковых экспериментов.

В таблицах 4.1 – 4.3 используются следующие обозначения:

— Ш — реализация алгоритма сортировки Шелла;

— Г — реализация алгоритма гномьей сортировки;

— П — реализация алгоритма пирамидальной сортировки.

Таблица 4.1 – Время работы реализации алгоритмов на массивах, отсортированных в обратном порядке (в с)

Размер массива	Ш	Г	П
1000	$5.937 \cdot 10^{-4}$	$3.125 \cdot 10^{-4}$	$3.031 \cdot 10^{-3}$
2000	$1.313 \cdot 10^{-3}$	$1.187 \cdot 10^{-3}$	$6.781 \cdot 10^{-3}$
3000	$2.281 \cdot 10^{-3}$	$2.594 \cdot 10^{-3}$	$1.109 \cdot 10^{-2}$
4000	$3.125 \cdot 10^{-3}$	$4.500 \cdot 10^{-3}$	$1.578 \cdot 10^{-2}$
5000	$4.281 \cdot 10^{-3}$	$6.938 \cdot 10^{-3}$	$1.975 \cdot 10^{-2}$
6000	$5.281 \cdot 10^{-3}$	$9.938 \cdot 10^{-3}$	$2.409 \cdot 10^{-2}$
7000	$6.094 \cdot 10^{-3}$	$1.341 \cdot 10^{-2}$	$2.859 \cdot 10^{-2}$
8000	$6.875 \cdot 10^{-3}$	$1.734 \cdot 10^{-2}$	$3.347 \cdot 10^{-2}$
9000	$8.531 \cdot 10^{-3}$	$2.234 \cdot 10^{-2}$	$3.806 \cdot 10^{-2}$

Таблица 4.2 – Время работы реализации алгоритмов на отсортированных массивах (в с)

Размер массива	Ш	Г	П
1000	$5.937 \cdot 10^{-4}$	$9.375 \cdot 10^{-5}$	$3.063 \cdot 10^{-3}$
2000	$1.500 \cdot 10^{-3}$	$2.188 \cdot 10^{-4}$	$6.938 \cdot 10^{-3}$
3000	$2.250 \cdot 10^{-3}$	$2.812 \cdot 10^{-4}$	$1.109 \cdot 10^{-2}$
4000	$3.187 \cdot 10^{-3}$	$4.062 \cdot 10^{-4}$	$1.528 \cdot 10^{-2}$
5000	$4.313 \cdot 10^{-3}$	$5.000 \cdot 10^{-4}$	$1.984 \cdot 10^{-2}$
6000	$5.125 \cdot 10^{-3}$	$5.625 \cdot 10^{-4}$	$2.409 \cdot 10^{-2}$
7000	$6.031 \cdot 10^{-3}$	$6.563 \cdot 10^{-4}$	$2.853 \cdot 10^{-2}$
8000	$6.594 \cdot 10^{-3}$	$8.125 \cdot 10^{-4}$	$3.369 \cdot 10^{-2}$
9000	$8.500 \cdot 10^{-3}$	$9.063 \cdot 10^{-4}$	$3.825 \cdot 10^{-2}$

Таблица 4.3 – Время работы реализации алгоритмов на случайно упорядоченных массивах (в с)

Размер массива	Ш	Г	П
1000	$1.969 \cdot 10^{-3}$	$4.687 \cdot 10^{-4}$	$8.484 \cdot 10^{-3}$
2000	$4.484 \cdot 10^{-3}$	$1.359 \cdot 10^{-3}$	$1.891 \cdot 10^{-2}$
3000	$7.547 \cdot 10^{-3}$	$2.687 \cdot 10^{-3}$	$3.008 \cdot 10^{-2}$
4000	$1.003 \cdot 10^{-2}$	$4.469 \cdot 10^{-3}$	$4.127 \cdot 10^{-2}$
5000	$1.358 \cdot 10^{-2}$	$6.734 \cdot 10^{-3}$	$5.317 \cdot 10^{-2}$
6000	$1.648 \cdot 10^{-2}$	$9.312 \cdot 10^{-3}$	$6.509 \cdot 10^{-2}$
7000	$1.914 \cdot 10^{-2}$	$1.198 \cdot 10^{-2}$	$7.742 \cdot 10^{-2}$
8000	$2.178 \cdot 10^{-2}$	$1.555 \cdot 10^{-2}$	$8.947 \cdot 10^{-2}$
9000	$2.667 \cdot 10^{-2}$	$1.956 \cdot 10^{-2}$	$1.035 \cdot 10^{-1}$

На рисунках 4.2 – 4.4 изображены графики зависимостей времени выполнения реализаций сортировок от размеров массивов.

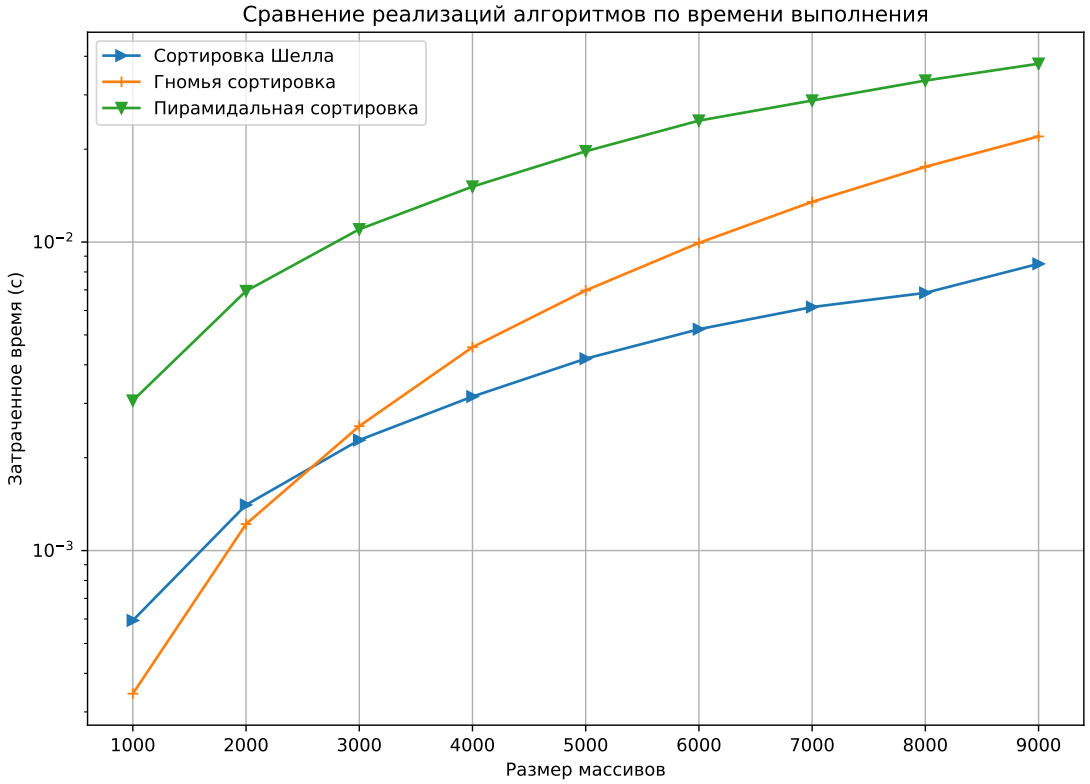

Рисунок 4.2 – Сравнение реализаций алгоритмов по времени выполнения на массивах, отсортированных в обратном порядке

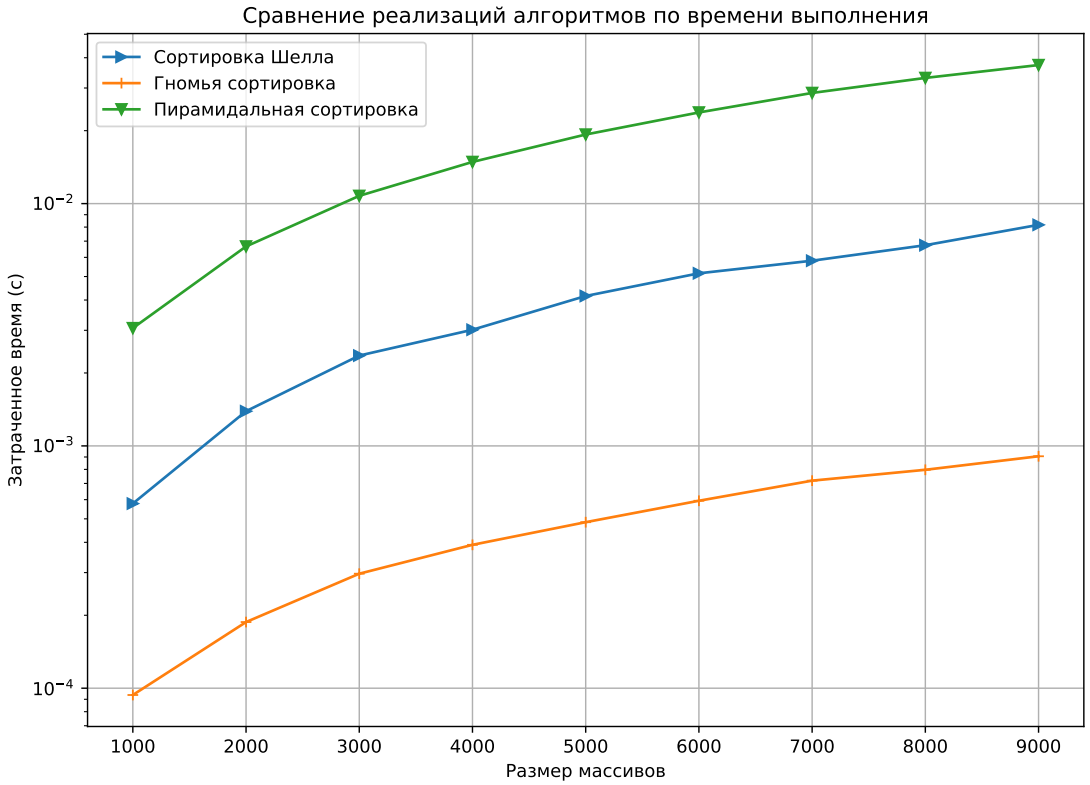

Рисунок 4.3 – Сравнение реализаций алгоритмов по времени выполнения на отсортированных массивах

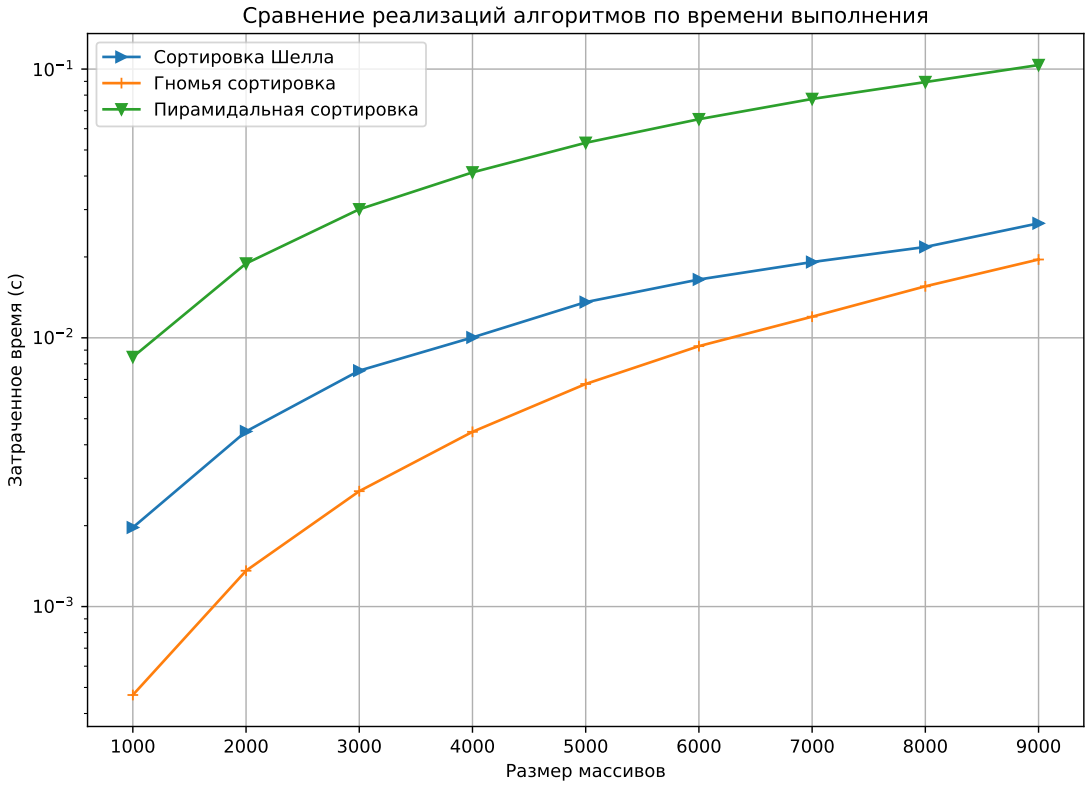

Рисунок 4.4 – Сравнение реализаций алгоритмов по времени выполнения на случайно упорядоченных массивах

4.4 Характеристики по памяти

Введем следующие обозначения:

— n — длина массива, который необходимо отсортировать arr;

— $size()$ — функция, вычисляющая размер в байтах;

— int — целочисленный тип данных;

— $float$ — вещественный тип данных.

Максимальное требование по памяти реализации алгоритма Шелла складывается из 5 локальных переменных типа int, адреса возврата int, возвращаемого значения (ссылки) и рассчитывается по формуле (4.1).

$$f_{memshell} = 6 \cdot size(int) + size(float*). \tag{4.1}$$

Аналогично, реализация алгоритма гномьей сортировки максимально требует памяти под 2 локальные переменные типа int, адрес возврата int, возвращаемое значение (ссылку), т. о. требования по памяти рассчитываются по формуле (4.2).

$$f_{memgnome} = 3 \cdot size(int) + size(float*). \tag{4.2}$$

Максимальное требование реализации алгоритма пирамидальной сортировки по памяти формируется из 3 локальных переменных типа int, адреса возврата int, возвращаемого по ссылке значения, при этом максимальная глубина рекурсии подпрограммы *heapify* равна $\log_2 N$. В подпрограмме *heapify* используются 3 локальных переменных типа int, а для вызова в нее необходимо передать массив по ссылке и 2 переменные типа int. Память требуемая реализацией алгоритма пирамидальной сортировки рассчитывается по формуле (4.3).

$$f_{heap} = 3 \cdot size(int) + size(float*) + \log_2 N(6 \cdot size(int) + size(float*)). \tag{4.3}$$

Вывод

В результате замеров времени выполнения реализаций различных алгоритмов было выявлено, что для массивов длины 9000, отсортированных в обратном порядке, реализация алгоритма Шелла по времени оказалась в 2.6 раза лучше, чем реализация гномьей сортировки, и в 4.5 раза лучше реализации пирамидальной сортировки. В свою очередь, реализация гномьей сортировки оказалась лучше в 1.7 раз по времени выполнения, чем реализация пирамидальной сортировки. Что соответствует теоретической оценке трудоемкости. Поскольку алгоритм Шелла по теоретической оценке трудоемкости в худшем случае выигрывает по константе гномью сортировку, $O(\frac{32}{3}N^2)$ и $O(\frac{23}{2}N^2)$ соответственно, при этом алгоритм пирамидальной сортировки, обладая теоретической оценкой трудоемкости $O(\frac{87}{2}N\log_2 N)$, из-за большой константы проигрывает остальным, хоть и обладает меньшей скоростью роста.

Для отсортированных массивов длинной 9000 реализация гномьей сортировки оказалась лучше по времени в 9 раза, чем реализация алгоритма Шелла, и в 42 раза лучше, чем реализация пирамидальной сортировки. В свою очередь, реализация пирамидальной сортировки на отсортированных массивах, оказалась хуже в 4.5 раза, чем реализации алгоритма Шелла по времени выполнения. Что соответствует теоретической оценке трудоемкости. Поскольку в лучшем случае алгоритм гномьей сортировки обладает наименьшей асимптотической оценкой и малой константой $O(7N)$. А алгоритм Шелла выигрывает алгоритм пирамидальной сортировки по константе, $O(10N\log_2 N)$ и $O(29N\log_2 N)$ соответственно.

Для случайно упорядоченных массивов длинной 9000 реализация гномьей сортировки оказалась лучше по времени в 1.4 раза, чем реализация алгоритма Шелла, и в 5.3 раза лучше, чем реализация пирамидальной сортировки. В свою очередь, реализация пирамидальной сортировки на случайно упорядоченных массивах, оказалась хуже в 3.9 раз, чем реализации алгоритма Шелла по времени выполнения.

При этом меньше всего памяти требует реализация гномьей сортировки, а больше всего — реализация пирамидальной сортировки.

Стоит заметить, что для обратно упорядоченных массивов длинной менее 2500, реализация гномьей сортировки показывала лучшие результаты. На случайно упорядоченных массивах, гномья сортировка хоть и оказалась

эффективней, но обладая большей скоростью роста, при больших длинах массивов она окажется менее эффективной, чем сортировка Шелла и пирамидальная. То же касается и пирамидальной сортировки, за счет большой константы, она показала худший результат во всех случаях, но поскольку асимптотическая оценка этого алгоритма меньше остальных для случайно упорядоченных и обратно упорядоченных массивов, данная реализация покажет лучшую эффективность по времени при гораздо больших длинах массивов.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута, а именно были рассмотрены алгоритмы сортировок.

Для достижения поставленной цели были выполнены следующие задачи.

— описаны алгоритмы гномьей, пирамидальной сортировок и сортировка по алгоритму Шелла;

— разработано программное обеспечение, реализующее алгоритмы сортировок;

— выбраны инструменты для реализации алгоритмов и замера процессорного времени их выполнения;

— проведен анализ затрат реализаций алгоритмов по времени.

В результате исследования реализаций различных алгоритмов было получено, что для массивов длины 9000, отсортированных в обратном порядке, реализация алгоритма Шелла по времени оказалась в 2.6 раза лучше, чем реализация гномьей сортировки, и в 4.5 раза лучше реализации пирамидальной сортировки. В свою очередь, реализация гномьей сортировки оказалась лучше в 1.7 раз по времени выполнения, чем реализация пирамидальной сортировки. Для отсортированных массивов длинной 9000 реализация гномьей сортировки оказалась лучше по времени в 9 раза, чем реализация алгоритма Шелла, и в 42 раза лучше, чем реализация пирамидальной сортировки. В свою очередь, реализация пирамидальной сортировки на отсортированных массивах, оказалась хуже в 4.5 раза, чем реализации алгоритма Шелла по времени выполнения. Для случайно упорядоченных массивов длинной 9000 реализация гномьей сортировки оказалась лучше по времени в 1.4 раза, чем реализация алгоритма Шелла, и в 5.3 раза лучше, чем реализация пирамидальной сортировки. В свою очередь, реализация пирамидальной сортировки на случайно упорядоченных массивах, оказалась хуже в 3.9 раз, чем реализации алгоритма Шелла по времени выполнения.

Для обратно упорядоченных массивов длинной менее 2500, реализация гномьей сортировки показывала лучшие результаты. На случайно упорядоченных массивах, гномья сортировка хоть и оказалась эффективней, но обладая

большей скоростью роста, при больших длинах массивов она окажется менее эффективной, чем сортировка Шелла и пирамидальная. То же касается и пирамидальной сортировки, за счет большой константы, она показала худший результат во всех случаях, но поскольку асимптотическая оценка этого алгоритма меньше остальных для случайно упорядоченных и обратно упорядоченных массивов, данная реализация покажет лучшую эффективность по времени при гораздо больших длинах массивов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Э. К. Д.* Искусство программирования, том 3. Сортировка и поиск, 2-е изд. //. Т. 832. — Пер. с англ. М.: ООО 'И. Д. Вильямс', 2007.
2. Гномья сортировка: [Электронный ресурс]. — Режим доступа: `https://kvodo.ru/gnome-sorting.html` (дата обращения: 30.10.2023).
3. Heap Sort – Data Structures and Algorithms Tutorials: [Электронный ресурс]. — Режим доступа: `https://www.geeksforgeeks.org/heap-sort/` (дата обращения: 30.10.2023).
4. Сортировка Шелла: [Электронный ресурс]. — Режим доступа: `https://kvodo.ru/sortirovka-shella.html` (дата обращения: 31.10.2023).
5. The official home of the Python Programming Language [Электронный ресурс]. — Режим доступа: `https://www.python.org/` (дата обращения: 19.09.2023).
6. time — Time access and conversions [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/library/time.html` (дата обращения: 19.09.2023).
7. Amd [Электронный ресурс]. — Режим доступа: `https://www.amd.com/en.html` (дата обращения: 28.09.2023).
8. Windows 10 Pro 22h2 64-bit [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows10` (дата обращения: 28.09.2023).