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 Динь Вьет Ань

Группа ИУ7И-54Б

Оценка (баллы) _____

Преподаватель Волкова Л.Л., Строганов Ю.В.

Оглавление

Введение

Сортировка — процесс перегруппировки заданной последовательности объектов в определенном порядке. Это одна из главных процедур обработки структурированной информации.

Существует множество различных методов сортировки данных, однако любой алгоритм сортировки имеет:

— сравнение, определяющее упорядочность пары элементов;

— перестановка, меняющая местами пару элементов;

— алгоритм сортировки, использующий сравнение и перестановки.

Алгоритмы сортировки имеют большое значение, так как позволяют эффективнее проводить работу с последовательностью данных. Например, возьмем задачу поиска элемента в последовательности — при работе с отсортированным набором данных время, которое нужно на нахождение элемента, пропорционально логарифму количства элементов. Последовательность, данные которой расположены в хаотичном порядке, занимает время, которое пропорционально количеству элементов, что куда больше логарифма.

Задачи данной лабораторной работы:

1) описать и реализовать три алгоритма сортировки: гномья сортировка, сортировка вставками и сортировка бинарным деревом;

2) провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

3) провести сравнительный анализ алгоритмов на основе экспериментальных данных по времени выполнения программы;

4) описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов гномьей сортировки, сортировки вставками и сортировки двоичным деревом поиска.

1.1 Гномья сортировка

Гномья сортировка [1] — алгоритм сортировки, который использует только один цикл, что является редкостью. В этой сортировке массив просматривается слева-направо, при этом сравниваются и, если нужно, меняются соседние элементы. Если происходит обмен элементов, то происходит возвращение на один шаг назад. Если обмена не было — алгоритм продолжает просмотр массива в поисках неупорядоченных пар.

1.2 Сортировка вставками

Сортировка вставками [2] — алгоритм сортировки, котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помешается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к входным данным алгоритма.

1.3 Сортировка бинарным деревом

Сортировка бинарным деревом [1] — универсальный алгоритм сортировки, заключающийся в построении двоичного дерева поиска по клю-

чам массива, с последующей сборкой результирующего массива путём обхода узлов построенного дерева в необходимом порядке следования ключей.

Шаги алгоритма:

1) построить двоичное дерево поиска по ключам массива;
2) собрать результирующий массив путём обхода узлов дерева поиска в необходимом порядке следования ключей;
3) вернуть, в качестве результата, отсортированный массив.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: гномьей сортировки, сортировки вставками и сортировки двоичным деревом поиска. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

В этом разделе будут приведены требования к вводу и прогамме, а также схемы алгоритмов и вычисления трудоемкости данных алгоритмов.

2.1 Требования к вводу

На вход подаются массив чисел и функция, которая позвояет сравнить два числа в масссиве между собой. При вводе пустого масссива программа не должна аварийно завершатся.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 и 2.4 представлены схемы алгоритмов гномьей сортировки, сортировки вставками, построения двоичного дерева поиска и сортировки двоичным деревом соответственно.

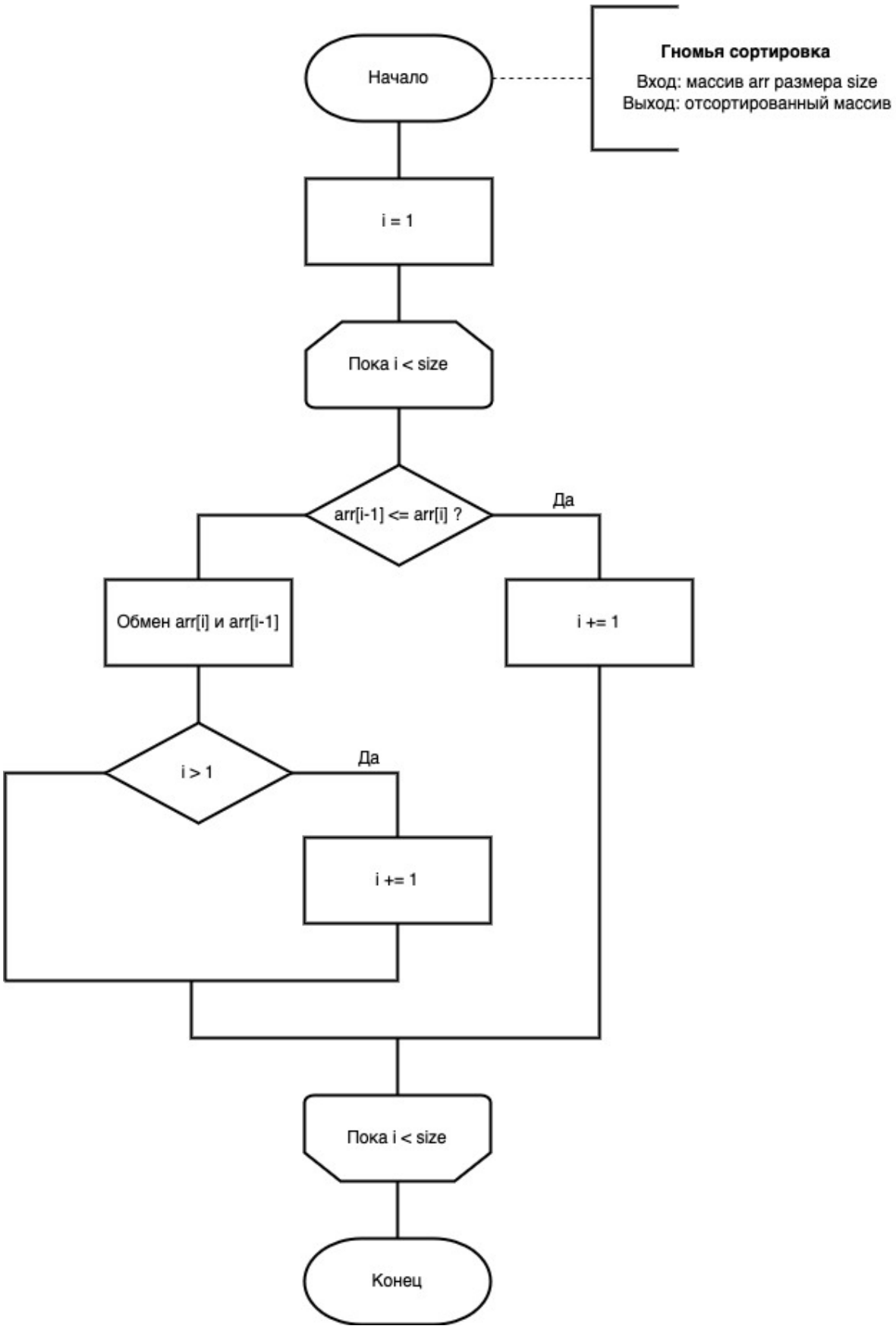


Рисунок 2.1 – Схема алгоритма гномьей сортировки

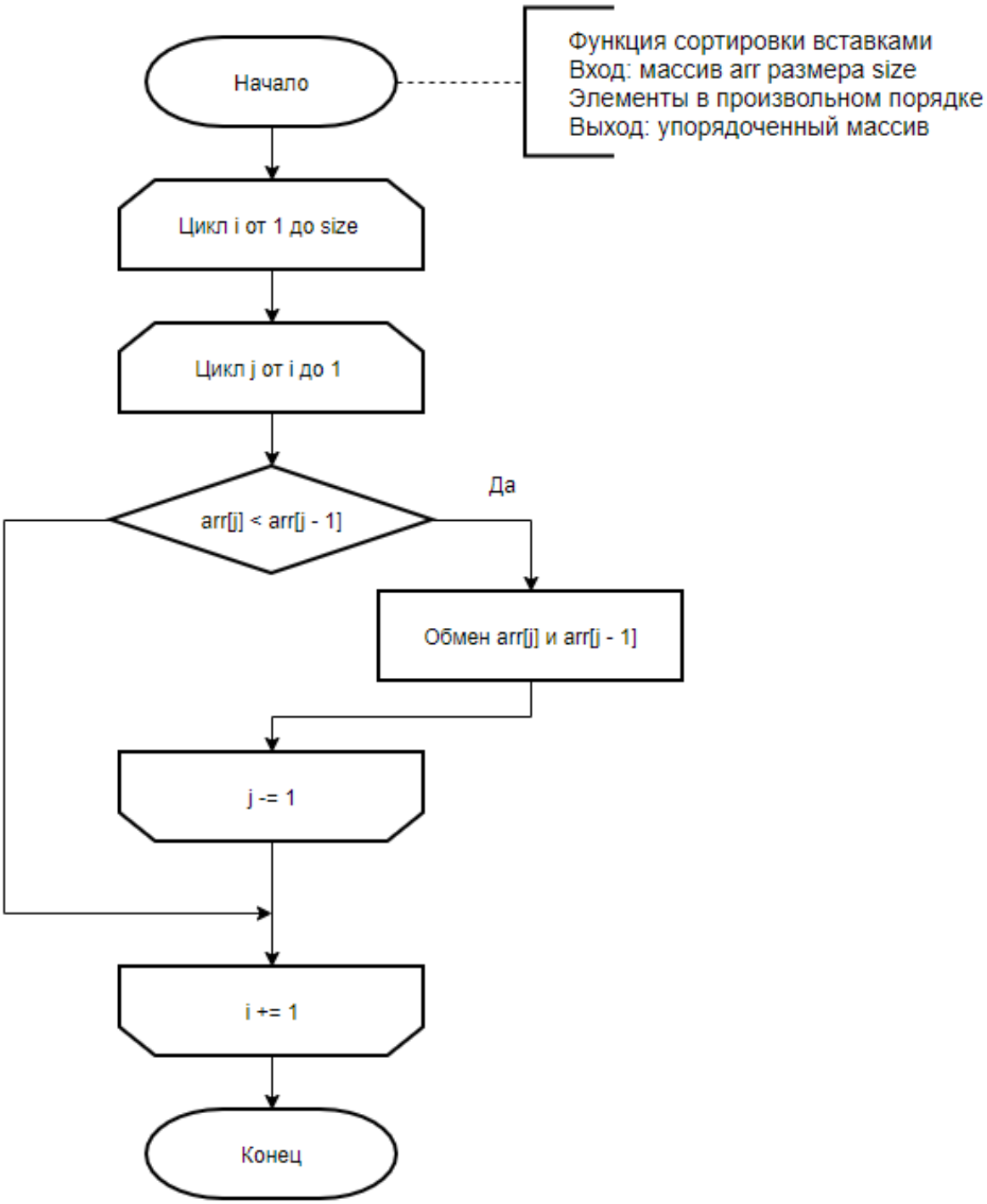


Рисунок 2.2 – Схема алгоритма сортировки вставками

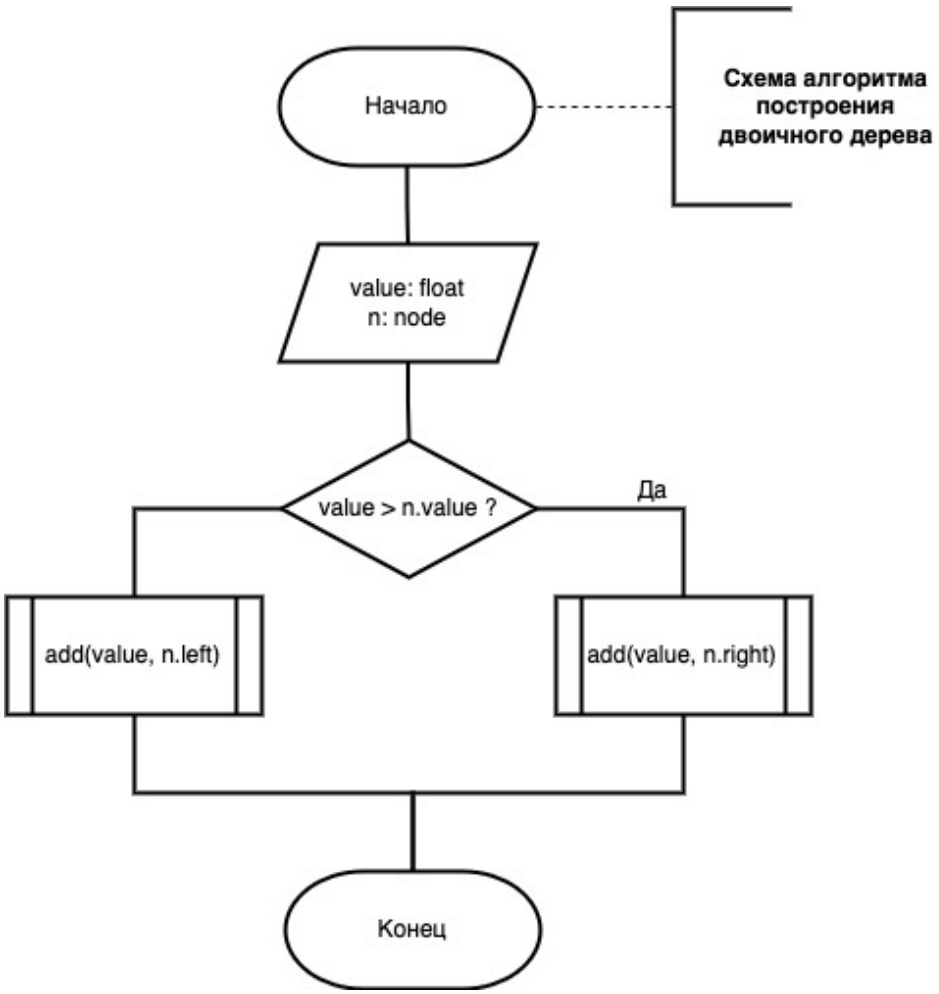


Рисунок 2.3 – Схема алгоритма построения двоичного дерева поиска

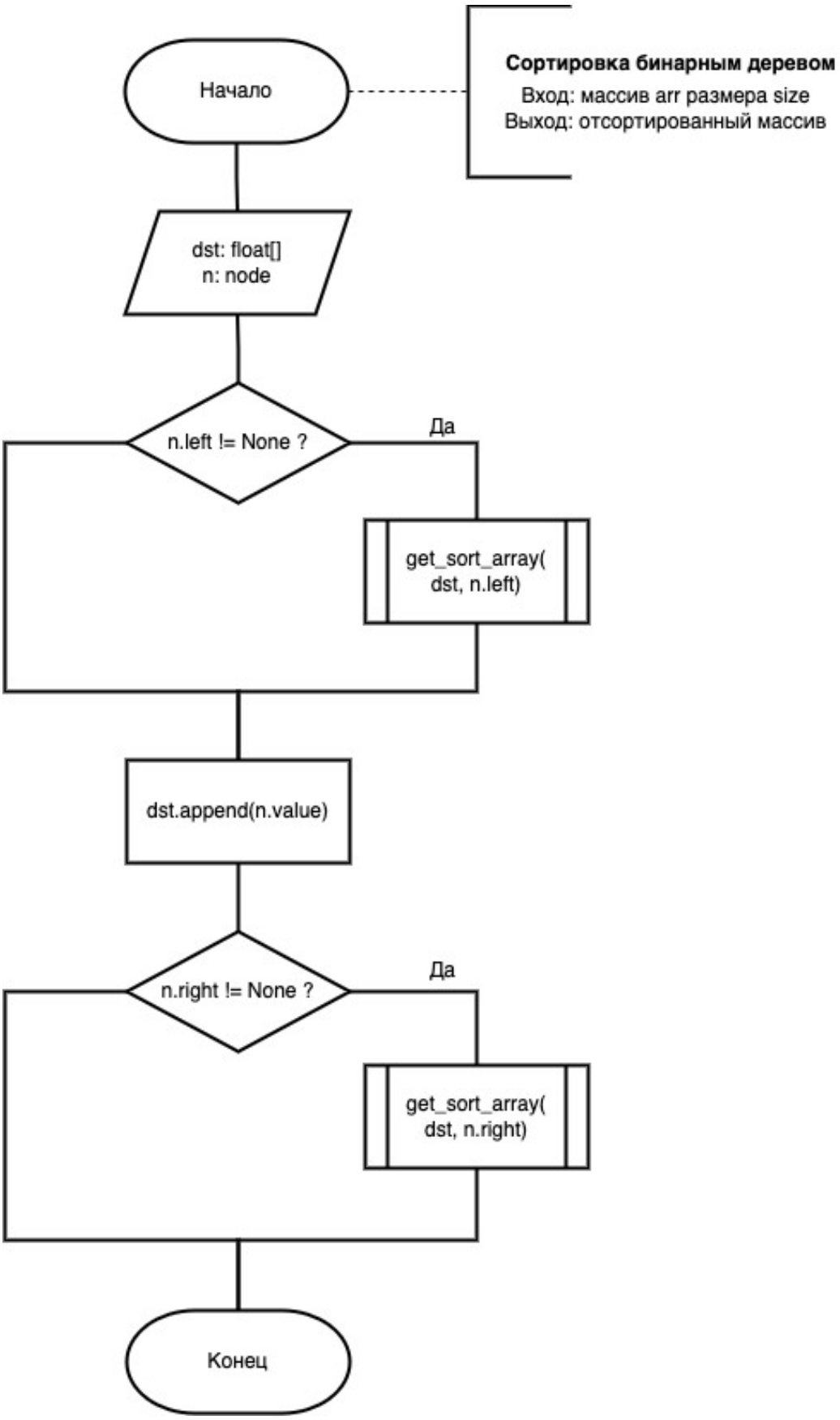


Рисунок 2.4 – Схема алгоритма сортировки двоичным деревом поиска

2.3 Модель вычислений

Введем модель вычислений, которая потребуется для определния трудоемкости каждого отдельно взятого алгоритма сортировки:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, *, \%, =, ==, != , <, >, <=, >=, [] \tag{2.1}$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.3}$$

4. трудоемкость вызова функции равна 0.

2.4 Трудоёмкость алгоритмов

Определим трудоемкость выбранных алгоритмов сортировки по коду. Во всех последующих вычислениях обозначим размер массивов как N.

2.4.1 Алгоритм гномьей сортировки

Трудоёмкость в лучшем случае (при уже отсортированном массиве):

$$f_{best} = 1 + N(4 + 1) = 5N + 1 = O(N) \tag{2.4}$$

Трудоёмкость в худшем случае (при массиве, отсортированном в обратном порядке):

$$f_{worst} = 1 + N(4 + (N-1) * (7+2)) = 9N^2 - 5N + 1 = O(N^2) \quad (2.5)$$

2.4.2 Алгоритм сортировки вставками

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

$$f = 2 + (N-1)(2+2+4+3) = 13N - 11 = O(N) \quad (2.6)$$

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

$$f = 2+(N-1)(2+2+6+\frac{N+2}{2}(5+4)+3) = \frac{9}{2}N^2+\frac{35}{2}N-20 = O(N^2) \quad (2.7)$$

2.4.3 Алгоритм сортировки бинарным деревом

Трудоемкость алгоритма сортировки бинарным деревом состоит из:

— Трудоемкости построения бинарного дерева, которая равна:

$$f_{make\_tree} = (5 \cdot \log(N) + 3) * N = N \cdot \log(N) \quad (2.8)$$

— Трудоемкости восстановления порядка элементов массива, которая равна:

$$f_{main\_loop} = 7N \quad (2.9)$$

Таким образом общая трудоемкость алгоритма выражается как:

$$f_{total} = f_{make\_tree} + f_{main\_loop} \tag{2.10}$$

Трудоемкость алгоритма в лучшем случае:

$$f_{best} = O(N \log(N)) \tag{2.11}$$

Трудоемкость алгоритма в худшем случае:

$$f_{worst} = O(N \log(N)) \tag{2.12}$$

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению (ПО), средства реализации и листинги кода.

3.1 Требования к ПО

К программа принимает на вход массив сравнимых элементов. В качестве результата возвращается массив в отсортированном порядке. Программа не должна аварийно завершаться при некорректном вводе.

3.2 Средства реализации

В качестве языка программирования (ЯП) для реализации данной лабораторной работы был выбран ЯП Python [3].

Данный язык имеет все небходимые инструменты для решения поставленной задачи.

Процессорное время работы реализаций алгоритмов было замерено с помощью функции process_time() из библиотеки time [4].

3.3 Сведения о модулях программы

Программа состоит из четырех модулей:

1) algorithms.py - хранит реализацию алгоритмов сортировок;
2) unit_tests.py - хранит реализацию тестирующей системы и тесты;
3) time_memory.py - хранит реализацию системы замера памяти и времени;
4) tools.py - хранит реализацию вспомогательных функций.

3.4 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов сортировок (гномьей, вставками и бинарным деревом).

Листинг 3.1 – Алгоритм гномьей сортировки

```
def gnome_sort(data):
    i, j, size = 1, 2, len(data)
    while i < size:
        if data[i - 1] <= data[i]:
            i, j = j, j + 1
        else:
            data[i - 1], data[i] = data[i], data[i - 1]
            i -= 1
            if i == 0:
                i, j = j, j + 1
    return data
```

Листинг 3.2 – Алгоритм сортировки вставками

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        j = i - 1
        key = arr[i]

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

Листинг 3.3 – Алгоритм сортировки бинарным деревом поиска

```
class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

    def add(self, val):
        if self.val > val:
            if self.left is None:
```

```
10                  self.left = Node(val)
11              else:
12                  self.left.add(val)
13          else:
14              if self.right is None:
15                  self.right = Node(val)
16              else:
17                  self.right.add(val)
18
19
20  def _one_node_sort(node, dst_list):
21      if node.left is not None:
22          _one_node_sort(node.left, dst_list)
23
24      dst_list.append(node.val)
25
26      if node.right is not None:
27          _one_node_sort(node.right, dst_list)
28
29
30  def binary_tree_sort(arr):
31      if len(arr) == 0:
32          return arr
33
34      head = Node(arr[0])
35      for i in range(1, len(arr)):
36          head.add(arr[i])
37
38      dst = []
39      _one_node_sort(head, dst)
40      return dst
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
[]	[]	[]
[1]	[1]	[1]
[0, 1]	[0, 1]	[0, 1]
[1, 0]	[0, 1]	[0, 1]
[2, −2]	[−2, 2]	[−2, 2]
[2, −2, 2]	[−2, 2, 2]	[−2, 2, 2]
[0, 0, 1, 2]	[0, 0, 1, 2]	[0, 0, 1, 2]
[5, 2, 1, 8, 9, 10]	[1, 2, 5, 8, 9, 10]	[1, 2, 5, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 5, 6, 7]	[1, 2, 3, 4, 5, 6, 7]
[9, 2, 1]	[1, 2, 9]	[1, 2, 9]
[9, −10000, −20000]	[−20000, −10000, 9]	[−20000, −10000, 9]
[5, 4, 4, 3]	[3, 4, 4, 5]	[3, 4, 4, 5]

Вывод

В этом разделе была представлена реализация алгоритмов сортировок. Тестирование показало, что алгоритмы реализованы правильно и работают корректно.

4 Исследовательская часть

В данном разделе приводятся результаты замеров затрат реализаций алгоритмов по процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система Window 10 Home Single Language;

— Память 8 Гб;

— Процессор 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц, 4 ядра.

Во время замера устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой замера.

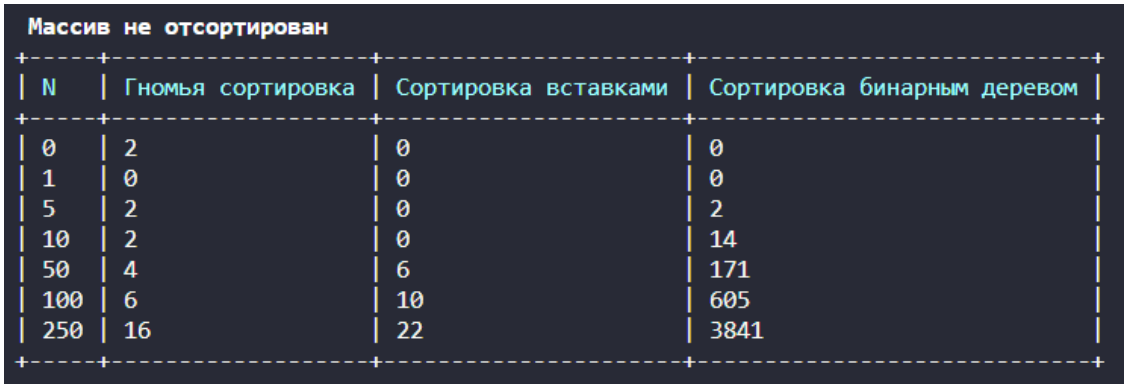
4.2 Время выполнения реализаций алгоритмов

Процессорное время реализаций алгоритмов замерялось при помощи функции process_time() из библиотеки time языка Python. Данная функция возвращает количество секунд, прошедших с начала эпохи, типа float.

Контрольная точка возвращаемого значения не определна, поэтому допустима только разница между результатами последовательных вызовов.

Замеры времени для каждой длины массива проводились 1000 раз. В качестве результата взято среднее время работы алгоритма на данной длине массива. При каждом запуске алгоритма, на вход подавались случайно сгенерированные массивы. Тестовые пакеты создавались до начала замера времени.

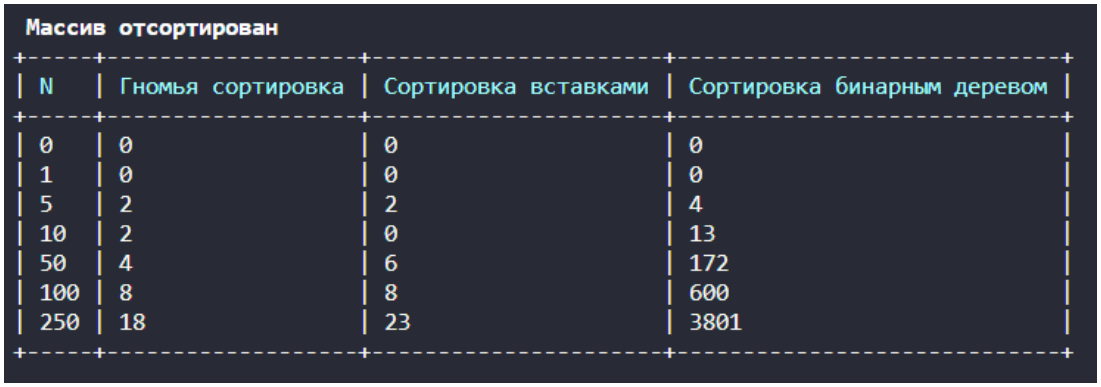
Результаты замеров приведены на рисунках 4.1, 4.2, 4.3 (в микросекундах).



Массив не отсортирован

N	Гномья сортировка	Сортировка вставками	Сортировка бинарным деревом
0	2	0	0
1	0	0	0
5	2	0	2
10	2	0	14
50	4	6	171
100	6	10	605
250	16	22	3841

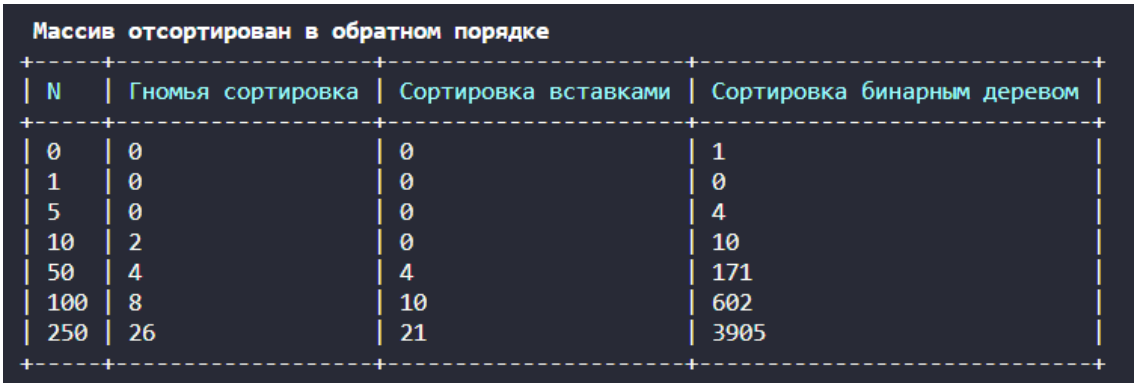
Рисунок 4.1 – Результаты замеров времени для неотсортированного массива (в микросекундах)



Массив отсортирован

N	Гномья сортировка	Сортировка вставками	Сортировка бинарным деревом
0	0	0	0
1	0	0	0
5	2	2	4
10	2	0	13
50	4	6	172
100	8	8	600
250	18	23	3801

Рисунок 4.2 – Результаты замеров времени для отсортированного массива (в микросекундах)



Массив отсортирован в обратном порядке

N	Гномья сортировка	Сортировка вставками	Сортировка бинарным деревом
0	0	0	1
1	0	0	0
5	0	0	4
10	2	0	10
50	4	4	171
100	8	10	602
250	26	21	3905

Рисунок 4.3 – Результаты замеров времени для отсортированного в обратном порядке массива (в микросекундах)

Вывод

Алгоритмы гномьей сортировки и сортировки вставками работает быстрее алгоритма сортировки бинарным деревом независимо от того, как расположены объекты в сортируемом массиве. Напротив, сортировка бинарным деревом проигрывает двум другим алгоритмам по времени, так как использует рекурсию.

Заключение

В ходе выполнения лабораторной работы были решены все задачи:

— описаны и реализованы 3 алгоритма сортировки: гномья, вставками, бинарным деревом;

— проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

— проведен сравнительный анализ алгоритмов на основе экспериментальных данных;

— подготовлен отчет о лабораторной работе.

Теоретически и экспериментально было подтверждено различие во временной эффективности различных алгоритмов сортировок, в частности алгоритма гномьей сортировки, алгоритма сортировки вставками и алгоритма сортировки бинарным деревом. Было получено, что гномья сортировка наиболее быстрая среди этих трех алгоритмов (работает в 4-5 раз быстрее), а сортировка бинарным деревом наиболее медленная (проигрывает в 3-4 раза).

Список использованных источников

1. Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования*. с. 834.

2. Сортировка вставками [Электронный ресурс]. Режим доступа: `https://kvodo.ru/sortirovka-vstavkami-2.html` (дата обращения: 02.03.2021).

3. Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 02.03.2023).

4. time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 02.03.2023).