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 Косарев А.А.

Группа ИУ7-51Б

Оценка (баллы)

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

Москва — 2022 г.

Содержание

Введение

В программировании алгоритмы сортировок занимают отдельное важное место. Их существует огромное количество, и все они обладают различными свойствами, такими как трудоемкость и объем требуемой памяти.

Упорядочивание элементов в массиве необходимо для решения множества задач в разных сферах, связанных с математикой, физикой, компьютерной графикой и т. д.

В данном отчете будут рассмотрены следующие методы сортировки:

— пирамидальная сортировка;

— сортировка выбором;

— сортировка перемешиванием (шейкерная).

1 Аналитическая часть

1.1 Цель и задачи

Цель — изучение и исследование трудоемкости алгоритмов сортировки.

Для достижения поставленной цели следует решить следующие задачи:

— описать алгоритмы сортировки (пирамидальную, выбором и перемешиванием);

— реализовать описанные алгоритмы;

— вывести оценку трудоемкости алгоритмов;

— выполнить замеры процессорного времени работы реализаций алгоритмов;

— провести сравнительный анализ заданных алгоритмов сортировки по затраченному времени работы реализаций.

1.2 Пирамидальная сортировка

Пирамидальная сортировка [1] — сортировка, основанная на структуре, называемой двоичной кучей или бинарным сортирующим деревом. Для этого дерева обязательно должны быть выполнены следующие условия:

— каждый лист дерева имеет глубину h, либо $h-1$, где h — максимальная глубина дерева;

— значение в любой вершине не меньше (не больше) значения ее потомков.

Для представления такого дерева обычно используется массив arr, в котором корень дерева хранится в первом элементе массива $arr[0]$, а потомки элемента $arr[i]$ — в $arr[2 \cdot i + 1]$ и $arr[2 \cdot i + 2]$.

Сам алгоритм сортировки включает в себя два этапа:

1) построение бинарного сортирующего дерева, удовлетворяющего перечисленным выше условиям;

2) замена корня на последний элемент кучи, уменьшение ее размера на 1 и построение сортирующего дерева для нового корня.

Второй этап алгоритма повторяется до тех пор, пока размер двоичной кучи не станет равным 1.

1.3 Сортировка выбором

Сортировка выбором [2] — метод сортировки, идея которого состоит в том, чтобы на очередном шаге найти минимум (максимум) и поставить его в начало (конец) неотсортированной части массива.

Тогда можно выделить следующие шаги алгоритма:

1) двигаясь по текущему подмассиву, произвести поиск индекса минимального (максимального) элемента;

2) поменять местами найденный минимум (максимум) и элемент, стоящий в начале (конце) неотсортированной части массива.

Описанные шаги выполняются, пока длина анализируемого подмассива не станет равна 1.

1.4 Сортировка перемешиванием

Сортировка перемешиванием (или шейкерная сортировка) [3] — разновидность сортировки пузырьком, особенностью которой является двунаправленное действие, то есть при движении от начала массива к концу "тонет" максимальный элемент, а при проходе от конца к началу "всплывает" минимальный.

Замечание: алгоритм сортировки пузырьком заключается в выполнении $N - 1$ проходов, в каждом из которых производится последовательное сравнение значений соседних элементов и обмен чисел местами, если предыдущее оказывается больше последующего. В очередном i-м проходе выполняется сравнение каждого из $N - i$ первых элементов с правым соседом.

Вывод

В данном разделе были теоретически разобраны алгоритмы пирамидальной сортировки, сортировки выбором и перемешиванием.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов сортировок, рассмотренных в аналитической части, выбранные типы данных и вывод оценки трудоемкости алгоритмов.

2.1 Алгоритмы сортировки

Схемы алгоритмов сортировки представлены на рисунках 2.1–2.4.

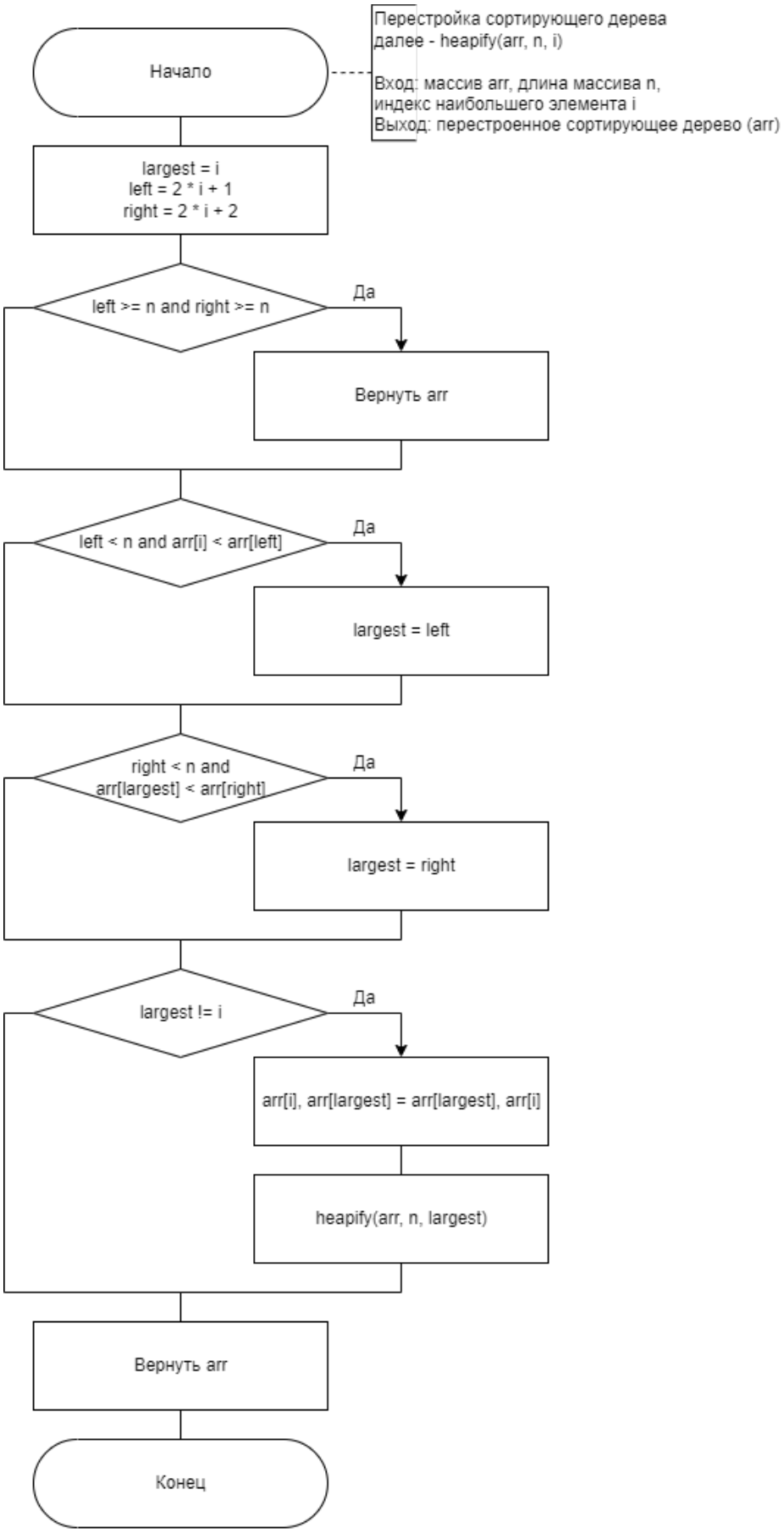

Рисунок 2.1 – Пирамидальная сортировка (перестройка сортирующего дерева)

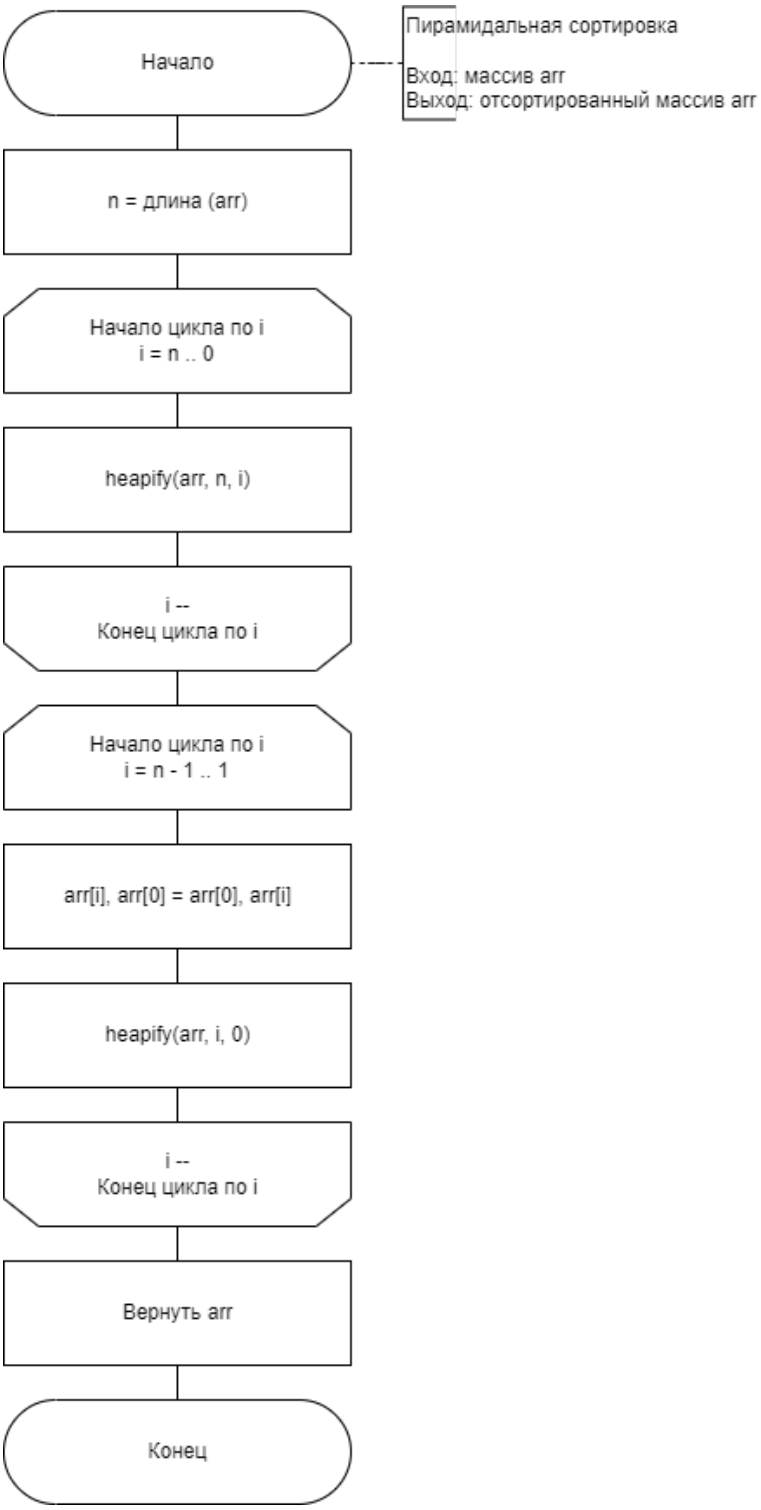

Рисунок 2.2 – Пирамидальная сортировка

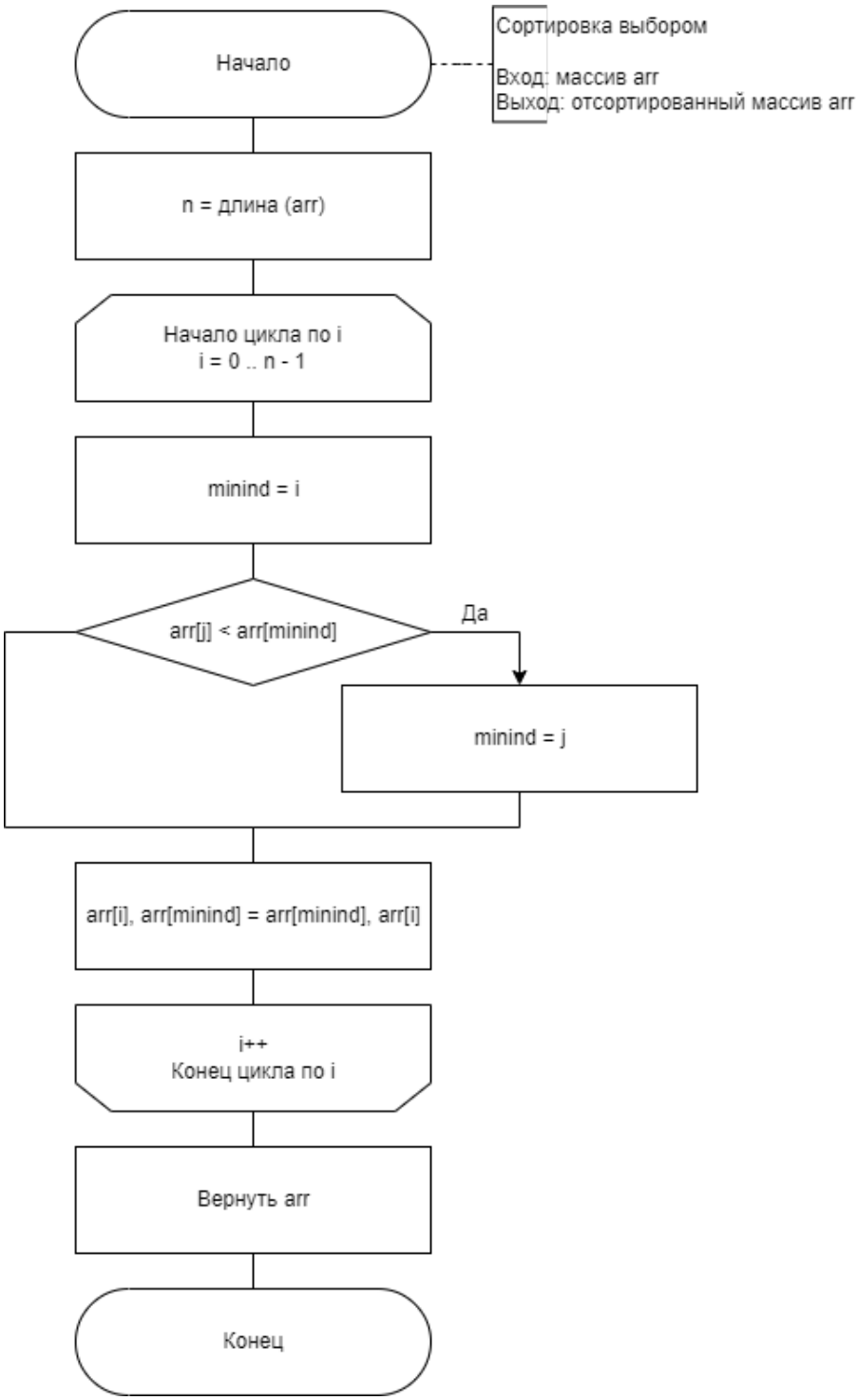

Рисунок 2.3 – Сортировка выбором

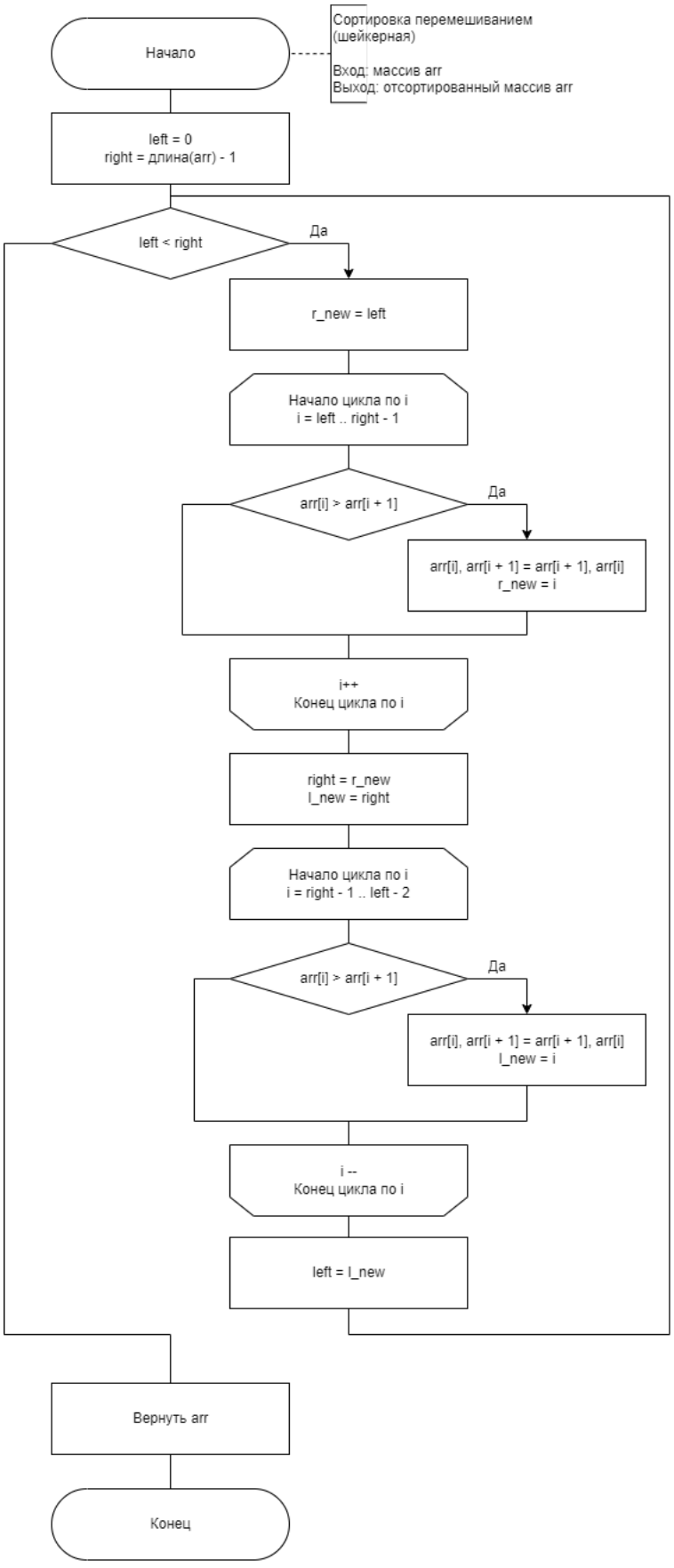

Рисунок 2.4 – Сортировка перемешиванием (шейкерная)

2.2 Описание типов данных

Выбранные типы данных:

— массив типа *int*;

— длина массива — тип *int*.

2.3 Оценка трудоемкости алгоритмов

Для вывода оценки трудоемкости алгоритмов следует описать модель оценки трудоемкости.

1. Трудоемкость базовых операций:

   1.1. /, %, * — 2;

   1.2. =, +=, −=, ++, −−, ==, !=, <, <=, >, >=, +, -, <<, >>, [ ] — 1.

2. Трудоемкость цикла с N итерациями:

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

3. Трудоемкость условного оператора:

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

Вывод оценки трудоемкости алгоритмов заданных сортировок приведен ниже.

Пирамидальная сортировка

В таблице 2.1 представлена построчная оценка трудоемкости пирамидальной сортировки.

Таблица 2.1 – Построчная оценка
трудоемкости пирамидальной сортировки

Строка кода	Вес
largest = i	1
left = 2 * i + 1	4
right = 2 * i + 2	4
if left >= n and right >= n:	2
return	0
if left < n and arr[i] < arr[left]:	4
largest = left	1
if right < n and arr[largest] < arr[right]:	4
largest = right	1
if largest != i:	1
arr[i], arr[largest] = arr[largest], arr[i]	5
heapify(arr, n, largest)	0

Для процесса перестройки бинарного дерева из N элементов в худшем случае потребуется log_2N операций. После N итераций все элементы будут отсортированы, а значит сложность сортировки определяестя формулой $N \cdot log_2N$.

Итого трудоемкость – O($N \cdot log_2N$).

Сортировка выбором

В таблице 2.2 представлена построчная оценка трудоемкости сортировки выбором.

Таблица 2.2 – Построчная оценка
трудоемкости сортировки выбором

Строка кода	Вес
for i in range(len(arr)):	3
minind = i	1
for j in range(i + 1, len(arr)):	4
if arr[j] < arr[minind]:	3
minind = j	1
arr[i], arr[minind] = arr[minind], arr[i]	5
return arr	0

В лучшем случае (когда массив уже отсортирован в порядке возрастания):

f = 2 + N(3 + N(3 + 5 + 2)) = O(N^2).

В худшем случае (массив отсортирован в обратном порядке):

f = 2 + N(3 + N(3 + 6 + 2)) = O(N^2).

Итого трудоемкость – O(N^2).

Сортировка перемешиванием

В таблице 2.3 представлена построчная оценка трудоемкости сортировки перемешиванием.

Таблица 2.3 – Построчная оценка трудоемкости сортировки перемешиванием

Строка кода	Вес
left = 0	1
right = len(arr) - 1	2
while left < right:	1
rnew = left	2
for i in range(left, right):	3
if arr[i] > arr[i + 1]:	4
arr[i], arr[i + 1] = arr[i + 1], arr[i]	7
rnew = i	1
right = rnew	1
lnew = right	1
for i in range(right - 1, left - 1, -1):	5
if arr[i] > arr[i + 1]:	4
arr[i], arr[i + 1] = arr[i + 1], arr[i]	7
lnew = i	1
left = lnew	1
return arr	0

В лучшем случае (массив отсортирован):

f = 4 + N(4 + 5 + 5) = O(N).

В худшем случае (массив отсортирован в обратном порядке):

f = 2 + N(4 + 2 + N(2 + 12) + 2 + N(2 + 12)) = O(N^2).

Итого трудоемкость – O(N^2).

2.4 Исходные файлы программы

Программа состоит из следующих модулей:

— *main.py* – файл с главной функцией, вызывающей меню программы;

— *sorts.py* – файл, содержащий код алгоритмов сортировок;

— *proc_time.py* – файл с функциями подсчета процессорного времени алгоритмов.

Вывод

В этом разделе были представлены схемы алгоритмов пирамидальной сортировки и сортировок выбором и перемешиванием, вывод оценки их трудоемкости и выбранные типы данных.

Проведя анализ трудоемкости алгоритмов, можно сказать, что в большинстве случаев пирамидальная сортировка является более предпочтительной, однако сортировка перемешиванием менее затратна при работе с отсортированным массивом.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации лабораторной работы, представлены листинги алгоритмов сортировки, а также тестовые данные, которые использовались для проверки корректности работы алгоритмов.

3.1 Выбор языка программирования и среды разработки

Для реализации алгоритмов был выбран язык программирования *Python*, так как он предоставляет необходимую функциональность, позволяющую решить поставленные задачи на реализацию алгоритмов и выполнение замеров процессорного времени работы их реализаций, а в качестве среды разработки – *PyCharm*. Для замеров процессорного времени использовалась функция *process_time()* [5] из библиотеки *time*, а для построения графиков – библиотека *matplotlib* [6].

3.2 Реализация алгоритмов

Ниже представлены реализации алгоритмов сортировки перемешиванием, выбором и пирамидальной сортировки (листинги 3.1–3.4).

Листинг 3.1 – Пирамидальная сортировка (перестройка сортирующего дерева)

```
1 def heapify(arr, n, i):
2     largest = i
3     left = 2 * i + 1
4     right = 2 * i + 2
5
6     if left >= n and right >= n:
7         return
8
9     if left < n and arr[i] < arr[left]:
```

```
10          largest = left
11
12      if right < n and arr[largest] < arr[right]:
13          largest = right
14
15      if largest != i:
16          arr[i], arr[largest] = arr[largest], arr[i]
17          heapify(arr, n, largest)
```

Листинг 3.2 – Пирамидальная сортировка

```
1  def heapsort(arr):
2      n = len(arr)
3
4      for i in range(n, -1, -1):
5          heapify(arr, n, i)
6
7      for i in range(n - 1, 0, -1):
8          arr[i], arr[0] = arr[0], arr[i]
9          heapify(arr, i, 0)
10
11     return arr
```

Листинг 3.3 – Сортировка выбором

```
1  def selectionsort(arr):
2      for i in range(len(arr)):
3          minind = i
4          for j in range(i + 1, len(arr)):
5              if arr[j] < arr[minind]:
6                  minind = j
7          arr[i], arr[minind] = arr[minind], arr[i]
8      return arr
```

Листинг 3.4 – Сортировка перемешиванием (шейкерная)

```
1  def shakersort(arr):
2      left = 0
3      right = len(arr) - 1
4      while left < right:
5          r_new = left
```

```
6           for i in range(left, right):
7               if arr[i] > arr[i + 1]:
8                   arr[i], arr[i + 1] = arr[i + 1], arr[i]
9                   r_new = i
10          right = r_new
11          l_new = right
12          for i in range(right - 1, left - 1, -1):
13              if arr[i] > arr[i + 1]:
14                  arr[i], arr[i + 1] = arr[i + 1], arr[i]
15                  l_new = i
16          left = l_new
17      return arr
```

3.3 Тестирование

Классы эквивалентности:

— пустой массив;

— массив из одного элемента;

— массив из одинаковых элементов;

— отсортированный по возрастанию массив;

— отсортированный по убыванию массив;

— неотсортированный массив.

В таблице 3.1 представлены модульные тесты. Все тесты пройдены успешно.

Вывод

В этом разделе были рассмотрены средства реализации лабораторной работы, представлены листинги алгоритмов сортировок перемешиванием, выбором и пирамидальной сортировки, а также модульные тесты.

Таблица 3.1 – Модульные тесты

	Входные данные	Ожидаемый результат		
№	Массив	Пирамидальная	Выбором	Перемешиванием
1	[ ]	[ ]	[ ]	[ ]
2	[5]	[5]	[5]	[5]
3	[8, 8, 8, 8, 8]	[8, 8, 8, 8, 8]	[8, 8, 8, 8, 8]	[8, 8, 8, 8, 8]
4	[1, 5, 8, 10]	[1, 5, 8, 10]	[1, 5, 8, 10]	[1, 5, 8, 10]
5	[10, 7, 3, -1]	[-1, 3, 7, 10]	[-1, 3, 7, 10]	[-1, 3, 7, 10]
6	[8, 0, 12, -4]	[-4, 0, 8, 12]	[-4, 0, 8, 12]	[-4, 0, 8, 12]

4 Исследовательская часть

В данном разделе будут представлены примеры работы программы, проведены замеры процессорного времени и предоставлена информация о технических характеристиках устройства.

4.1 Технические характеристики устройства

Ниже представлены характеристики компьютера, на котором проводилось тестирование программы:

— операционная система Windows 10 Домашняя 21H2;

— оперативная память 16 Гб;

— процессор Intel(R) Core(TM) i7-10870H CPU @ 2.20 ГГц.

Во время тестирования ноутбук был подключен к сети электропитания. Из программного обеспечения были запущены только среда разработки браузер *Chrome* и *PyCharm*.

Загруженность компонентов:

— процессор – 15 %;

— оперативная память – 45 %.

4.2 Примеры работы программы

На рисунках 4.1–4.2 представлен результат работы программы. В каждом примере пользователем введен массив целых чисел и получены результаты его сортировки.

```
Меню
1. Сортировка массива:
     - Пирамидальная
     - Выбором
     - Перемешиванием
2. Замеры процессорного времени
0. Выход
>  1
Введите длину массива: 12
Введите массив (через Enter):
0
3
34
-45
4
-100
234
1
5
-3
6
24

 ================================================================================
Пирамидальная сортировка
Array:  [0, 3, 34, -45, 4, -100, 234, 1, 5, -3, 6, 24]
Sorted array:  [-100, -45, -3, 0, 1, 3, 4, 5, 6, 24, 34, 234]

 ================================================================================
Сортировка выбором
Array:  [0, 3, 34, -45, 4, -100, 234, 1, 5, -3, 6, 24]
Sorted array:  [-100, -45, -3, 0, 1, 3, 4, 5, 6, 24, 34, 234]

 ================================================================================
Сортировка перемешиванием
Array:  [0, 3, 34, -45, 4, -100, 234, 1, 5, -3, 6, 24]
Sorted array:  [-100, -45, -3, 0, 1, 3, 4, 5, 6, 24, 34, 234]
```

Рисунок 4.1 – Пример работы программы №1

```
Введите массив (через Enter):
4
12
93
0
9
-3
-25
0
13
34
-2
-1
23
230
-250
-9
12
10
-90
111
32
7
-5
9
279

 ========================================================================
Пирамидальная сортировка
Array:  [4, 12, 93, 0, 9, -3, -25, 0, 13, 34, -2, -1, 23, 230, -250, -9, 12, 10, -90, 111, 32, 7, -5, 9, 279]
Sorted array:  [-250, -90, -25, -9, -5, -3, -2, -1, 0, 0, 4, 7, 9, 9, 10, 12, 12, 13, 23, 32, 34, 93, 111, 230, 279]

 ========================================================================
Сортировка выбором
Array:  [4, 12, 93, 0, 9, -3, -25, 0, 13, 34, -2, -1, 23, 230, -250, -9, 12, 10, -90, 111, 32, 7, -5, 9, 279]
Sorted array:  [-250, -90, -25, -9, -5, -3, -2, -1, 0, 0, 4, 7, 9, 9, 10, 12, 12, 13, 23, 32, 34, 93, 111, 230, 279]

 ========================================================================
Сортировка перемешиванием
Array:  [4, 12, 93, 0, 9, -3, -25, 0, 13, 34, -2, -1, 23, 230, -250, -9, 12, 10, -90, 111, 32, 7, -5, 9, 279]
Sorted array:  [-250, -90, -25, -9, -5, -3, -2, -1, 0, 0, 4, 7, 9, 9, 10, 12, 12, 13, 23, 32, 34, 93, 111, 230, 279]
```

Рисунок 4.2 – Пример работы программы №2

4.3 Время выполнения реализаций алгоритмов

Для замера процессорного времени использована функция *process_time()* библиотеки *time*. Возвращаемый результат — время в миллисекундах, число типа *float*.

Чтобы получить достаточно точное значение, производилось усреднение времени для серии запусков расчета. В замерах использовались массивы длиной от 200 до 2800 символов, а сортировка каждого массива запускалась по 1000 раз.

В таблицах 4.1–4.3 представлено процессорное время работы алгоритмов сортировки.

Таблица 4.1 – Результаты замеров времени (неотсортированный массив)

Длина массива	Пирамидальная	Выбором	Перемешиванием
200	0.000391	0.000625	0.001234
400	0.000859	0.002859	0.005063
600	0.001422	0.006578	0.012047
800	0.001969	0.011797	0.022141
1000	0.002594	0.018625	0.036766
1200	0.003203	0.026750	0.051250
1400	0.003859	0.039563	0.078734
1600	0.004797	0.048188	0.096422
1800	0.005188	0.060828	0.120266
2000	0.005797	0.074297	0.151266
2200	0.006562	0.090563	0.181062
2400	0.007281	0.108969	0.212422
2600	0.007859	0.129266	0.281609
2800	0.009672	0.164656	0.354766

Таблица 4.2 – Результаты замеров времени (отсортированный по возрастанию массив)

Длина массива	Пирамидальная	Выбором	Перемешиванием
200	0.000766	0.001000	0.000016
400	0.001375	0.003625	0.000016
600	0.002031	0.008438	0.000031
800	0.002812	0.014984	0.000047
1000	0.003625	0.023609	0.000063
1200	0.004484	0.033719	0.000078
1400	0.005406	0.046594	0.000094
1600	0.006281	0.056734	0.000094
1800	0.005672	0.059719	0.000078
2000	0.006250	0.074250	0.000078
2200	0.007078	0.092594	0.000125
2400	0.008703	0.118984	0.000141
2600	0.009547	0.145531	0.000156
2800	0.011969	0.186812	0.000188

Таблица 4.3 – Результаты замеров времени (отсортированный по убыванию массив)

Длина массива	Пирамидальная	Выбором	Перемешиванием
200	0.000484	0.000906	0.002719
400	0.001016	0.003891	0.011469
600	0.001687	0.009000	0.027750
800	0.002406	0.016094	0.051094
1000	0.003141	0.025844	0.082578
1200	0.003906	0.038750	0.109063
1400	0.004047	0.042969	0.138953
1600	0.004719	0.056172	0.189469
1800	0.005500	0.071625	0.231750
2000	0.006141	0.088234	0.298125
2200	0.006906	0.120609	0.402500
2400	0.008797	0.140891	0.434625
2600	0.008672	0.148734	0.483844
2800	0.009141	0.176016	0.612531

На рисунках 4.3-4.5 также приведены результаты замеров процессорного времени.

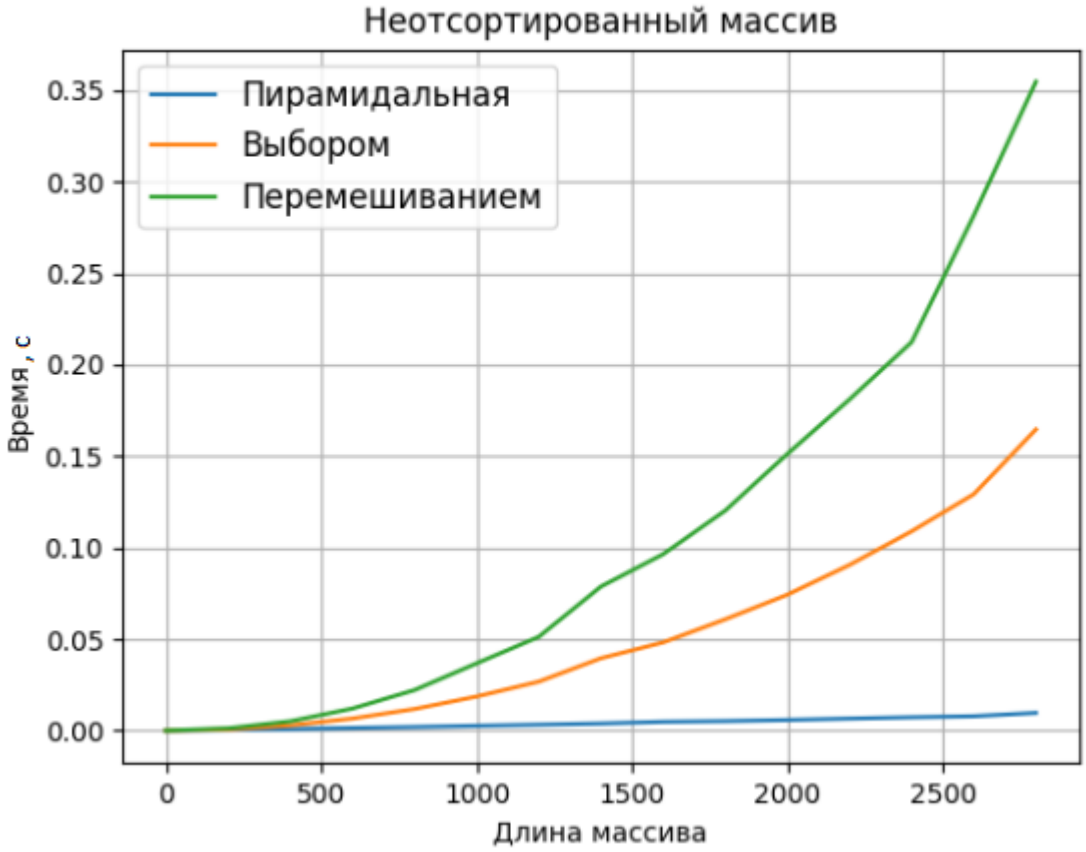

Рисунок 4.3 – Сравнение процессорного времени работы алгоритмов сортировки на неотсортированном массиве

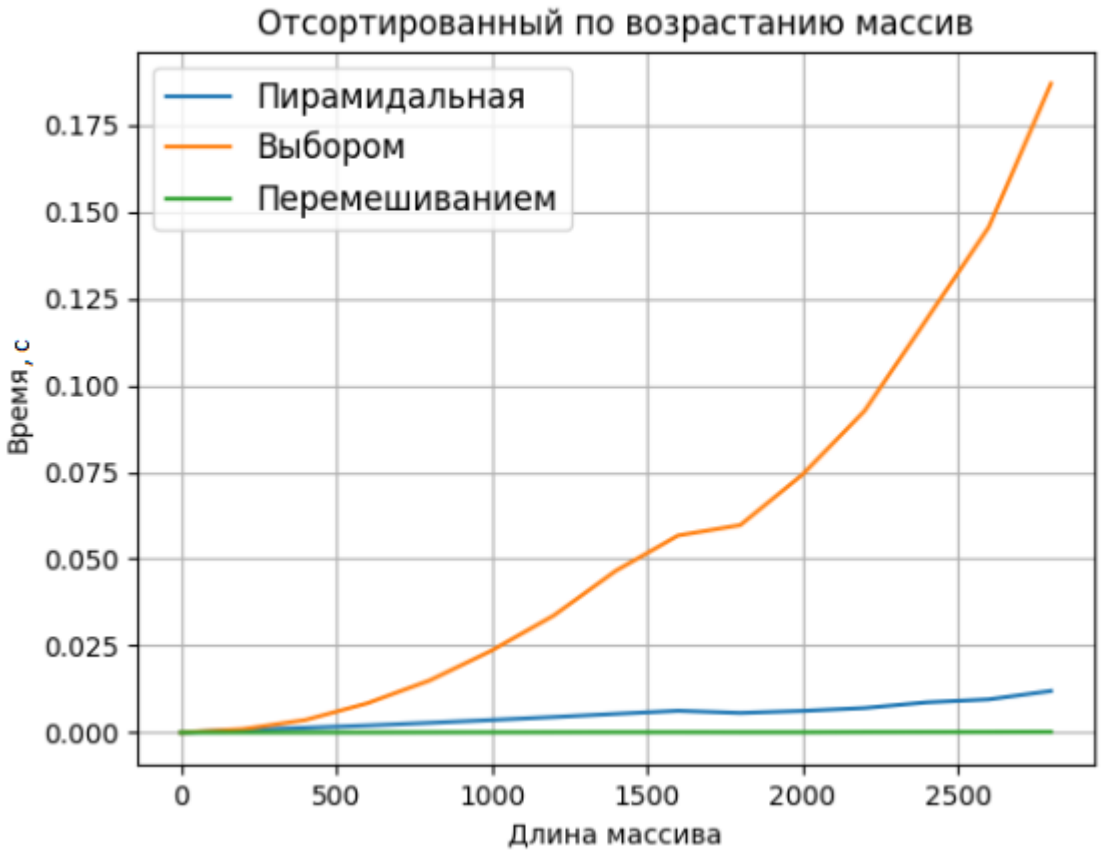

Рисунок 4.4 – Сравнение процессорного времени работы алгоритмов сортировки на отсортированном по возрастанию массиве

Рисунок 4.5 – Сравнение процессорного времени работы алгоритмов сортировки на отсортированном по убыванию массиве

4.4 Вывод

В результате замеров процессорного времени выделены следущие аспекты:

— на неотсортированном массиве наименее затратной по времени сортировкой является пирамидальная, а самой медленной оказалась сортировка перемешиванием;

— в случае, если массив уже отсортирован, сортировка выбором не предпочтительна к использованию, а сортировка перемешиванием, наоборот, тратит меньше всего времени.

Заключение

Цель, которая была поставлена в начале лабораторной работы, была достигнута: изучены и исследованы трудоемкости алгоритмов сортировки.

Решены все поставленные задачи:

— описаны и реализованы алгоритмы сортировок выбором, перемешиванием и пирамидальной сортировки;

— выведена оценка трудоемкости алгоритмов;

— выполнены замеры процессорного времени работы реализаций алгоритмов;

— проведен сравнительный анализ заданных алгоритмов сортировки по затраченному времени работы реализаций.

В результате исследования было определено, что при увеличении длины входного массива время работы реализаций алгоритмов сортировки увеличивается по разным законам.

Трудоемкости (в худшем случае):

— пирамидальная – $O(N \cdot log_2 N)$;

— выбором – $O(N^2)$;

— перемешиванием – $O(N^2)$.

Теоретическая оценка трудоемкости алгоритмов подтвердилась при замерах в ходе экспериментов.

Стоит отметить, что сортировка перемешиванием является самой быстродействующей среди рассмотренных трех при входном отсортированном массиве, однако в остальных случаях она проигрывает двум другим.

Исходя из проведенных экспериментов, можно сделать вывод: для упорядочивания элементов в массивах общего вида лучше использовать пирамидальную сортировку.

Список использованных источников

1. Пирамидальная сортировка [Электронный ресурс]. – URL: http://algolist.ru/sort/pyramid_sort.php (дата обращения: 10.10.2022)

2. Сортировка выбором [Электронный ресурс]. – URL: http://algolist.ru/sort/select_sort.php (дата обращения: 10.10.2022)

3. Сортировка перемешиванием [Электронный ресурс]. – URL: https://kvodo.ru/shaker-sort.html (дата обращения: 10.10.2022)

4. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск. — СПб.: Диалектика, 2019. — 832 с.

5. time — Time access and conversions [Электронный ресурс]. – URL: https://docs.python.org/3/library/time.html#time.process_time (дата обращения: 10.10.2022)

6. Matplotlib documentation [Электронный ресурс]. – URL: https://matplotlib.org/stable/index.html (дата обращения: 10.10.2022)

7. Windows 10 Home [Электронный ресурс]. – URL: https://www.microsoft.com/en-us/d/windows-10-home/d76qx4bznwk4?activetab=pivot:overviewtab (дата обращения: 10.10.2022)

8. Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz [Электронный ресурс]. – URL: https://ark.intel.com/content/www/ru/ru/ark/products/208018/intel-core-i710870h-processor-16m-cache-up-to-5-00-ghz.html (дата обращения: 10.10.2022)