

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Сортировка вставками, выбором, пузырьковая
Вариант 12

Выполнила:
Мкртчян К.Г.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024г.

Содержание отчета

Содержание отчета	2
Задачи по варианту.....	3
Задача №1. Сортировка вставкой [N баллов].....	3
Задача №3. Сортировка вставкой по убыванию [N баллов].....	5
Задача №9. Сложение двоичных чисел [N баллов]	8
Дополнительные задачи	9
Задача №2. Сортировка вставкой + [N баллов]	9
Задача №4. Линейный поиск [N баллов]	11
Задача №10. Палиндром [N баллов]	13
Вывод.....	16

Задачи по варианту

Задача №1. Сортировка вставкой [N баллов]

Текст задачи: Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$.

Решение:

```
-----
from lab1.src.verifications import data_verification1

@data_verification1
def insertion_sort(n, m):
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if m[i] < m[j]:
                m[i], m[j] = m[j], m[i]
                i, j = j, i
    return m

file = open("input1.txt")
test_n = int(file.readline())
test_m = list(map(int, file.readline().split(" ")))
open("output1.txt", "w").write(" ".join(map(str, insertion_sort(test_n,
test_m))))

-----

def data_verification1(func):
    def check(n, m, repeat=0):
        if 1 <= n <= 10**3 and all(abs(x) <= 10**9 for x in m) and len(m)
== n:
            return func(n, m)
        else:
            if repeat >= 2:
                return "Invalid data"
            else:
                print("Enter data again")
                n = int(input())
                m = [int(i) for i in input().split()]
                return check(n, m, repeat+1)
    return check

-----
```

Объяснение решения:

Функция `insertion_sort` принимает два параметра – количество чисел (n) и сами числа (m). Она проходится по всем элементам массива – от второго до последнего ($m[i]$) и сравнивает их с предыдущими элементами ($m[j]$), где j от $i-1$ до 0 , если $m[i] < m[j]$, то элементы и их индексы меняются местами: $m[i] = m[j]$ и наоборот, и $i = j$, а $j = i$. Индексы меняются местами для того, чтобы мы перемещали именно взятый изначально элемент (элемент с индексом i).

Для проверки значений – числа n и всех элементов списка m – напомним функцию (декоратор) – `data_verification1`. Она принимает на вход функцию (например `insertion_sort`), а также включает в себя функцию `check`, которая

будет проверять значение и вызывать `insertion_sort`, если данные удовлетворяют ограничениям, или просить пользователя ввести данные еще раз. Если пользователь уже в третий раз ввёл неподходящие значения, то функция вернет сообщение "Invalid data" и прекратит работу. Мы применим эту функцию для заданий 1, 2, 3, 5, 6, так как у всех этих заданий одинаковый формат ввода данных и одинаковые ограничения для значений.

Результат работы кода на примерах из текста задачи:

```

input1.txt x  output1.txt
6
31 41 59 26 41 58
put1.txt  output1.txt
26 31 41 41 58 59
  
```

Результат работы кода на минимальных значениях:

```

input1.txt x
1 1
2 0
put1.txt  output1.txt
0
  
```

На максимальных:

```

input1.txt x  output1.txt  task1.py  verifications.py
1000
771640048 -624537449 527489086 -831331990 249104921 350044689
-998669847 -997171016 -994714136 -992557071 -988264355 -986104339 -984158969 -981960835 -978395694 -972613485
  
```

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.0000038	19.5
6 31 41 59 26 41 58	0.0000083	19.89
1000 -749706382 149187416 -921224543 - 684411683...	0.0231	19.74

Вывод по задаче: Мы научились писать алгоритм сортировки вставкой, проверять значения, переданные в функцию, при помощи декораторов, а также писать тесты для проверки алгоритма сортировки.

Задача №3. Сортировка вставкой по убыванию [N баллов]

Текст задачи: Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swar. Формат входного и выходного файла и ограничения - как в задаче 1.

Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Решение:

```
-----
from lab1.src.verifications import data_verification1

def swap(a, b):
    c = b
    b = a
    a = c
    return a, b

@data_verification1
def insertion_sort(n, m):
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if m[i] > m[j]:
                m[i], m[j] = swap(m[i], m[j])
                i, j = j, i
    return m

file = open("input3.txt")
test_n = int(file.readline())
test_m = list(map(int, file.readline().split(" ")))
open("output3.txt", "w").write(" ".join(map(str, insertion_sort(test_n,
test_m))))
-----
```

Объяснение решения:

Функция `insertion_sort` работает также, как и функция из первой задачи, мы меняем только знак ($m[i] > m[j]$). Также мы используем функцию `swap()`, которая запишет в $m[i]$ - $m[j]$ и наоборот. Эта функция запишет значение второй переменной в вспомогательную ($c = b$), присвоит второй переменной значение первой ($b = a$) и присвоит первой значение вспомогательной ($a = c$). В Python нет необходимости писать такую функцию, так как можно поменять значения переменных местами через множественное/позиционное присваивание ($a, b = b, a$).

Также сортировку вставкой можно реализовать при помощи рекурсии:

```
-----
import sys
sys.setrecursionlimit(10**8)

def swap(a, b):
    c = b
```

```

b = a
a = c
return a, b

def insertion_sort_rec(n, m, index=1):
    if index == n:
        return m
    else:
        for j in range(index - 1, -1, -1):
            if m[index] > m[j]:
                m[index], m[j] = swap(m[index], m[j])
                index, j = j, index
        return insertion_sort_rec(n, m, index + 1)

file = open("input_rec3.txt")
test_n = int(file.readline())
test_m = list(map(int, file.readline().split(" ")))
open("output_rec3.txt", "w").write(" ".join(map(str, insertion_sort_rec(test_n, test_m))))

```

В этом решении у нас будет только один цикл для перебора элементов с индексами от $\text{index} - 1$ до 0. Мы также перебираем элементы со второго до последнего ($1 \leq \text{index} < n$). Для того, чтобы рассмотреть следующий элемент мы, вместо перехода на другую итерацию, снова вызовем функцию и передадим $\text{index} + 1$.

Результат работы кода на примерах из текста задачи:

Two screenshots showing the execution of the insertion sort algorithm on a list of 6 elements. In both cases, the input list is [31, 41, 59, 26, 41, 58] and the output list is [59, 58, 41, 41, 31, 26].

Результат работы кода на минимальных значениях:

A screenshot showing the execution of the insertion sort algorithm on a list of 1 element. The input is 1 and [1], and the output is 1 and [1].

Результат работы кода на максимальных значениях:

A screenshot showing the execution of the insertion sort algorithm on a list of 6 large numbers. The input is 1000 and [848272261, 949656085, 695332360, -616603306, 628878694]. The output is the same list sorted in descending order: [997793901, 996272290, 995818516, 994641448, 993051744].

```

input_rec3.txt x  output3.txt  input3.txt
1 1000
2 848272261 949656085 695332360 -616603306
output_rec3.txt x  input_rec3.txt  output3.txt
997793901 996272290 995818516 994641448 993051744

```

Для способа со вложенным циклом:

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.000004	18.68
6 31 41 59 26 41 58	0.0000104	18.48
1000 -749706382 149187416 -921224543 - 684411683...	0.027	18.76

Для способа с использованием рекурсии:

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.0000015	18.7
6 31 41 59 26 41 58	0.0000094	18.62
1000 -749706382 149187416 -921224543 - 684411683...	2.6553	19.64

Вывод по задаче: В этой задаче мы поняли, что принцип сортировки вставкой можно использовать не только для сортировки по возрастанию, но и для сортировки по убыванию. Поняли, как работает множественное/позиционное присваивание и реализовали его через функцию `swar ()`. Убедились, что сортировку вставкой можно реализовать

с помощью рекурсии, но такой метод значительно уступает по времени (в 100 раз дольше по времени при длине массива в 1000 элементов) методу с использованием вложенного цикла.

Задача №9. Сложение двоичных чисел [N баллов]

Текст задачи: рассмотрим задачу сложения двух n -битовых двоичных целых чисел, хранящихся в n -элементных массивах A и B . Сумму этих двух чисел необходимо занести в двоичной форме в $(n + 1)$ -элементный массив C . Напишите скрипт для сложения этих двух чисел.

Решение:

```
-----
from lab1.src.verifications import data_verification9

@data_verification9
def sum_dv(a, b):
    a = [int(i) for i in a]
    b = [int(i) for i in b]
    n = len(a)
    c = [int(i) for i in "0"*(n+1)]
    for i in range(1, n+1):

        s = a[-i] + b[-i] + c[-i]

        c[-i - 1] = s // 2
        c[-i] = s % 2
    result = "".join(map(str, c))
    result = result[result.index("1"):]
    return result
-----

def data_verification9(func):
    def check(a, b, repeat=0):
        if 1 <= len(a) == len(b) <= 10**3 and all((x in '01') for x in a)
and all(y in '01' for y in b):
            return func(a, b)
        else:
            if repeat >= 2:
                return "Invalid data"
            else:
                print("Enter data again")
                a, b = input().split()
                return check(a, b, repeat+1)
    return check
-----
```

Объяснение решения:

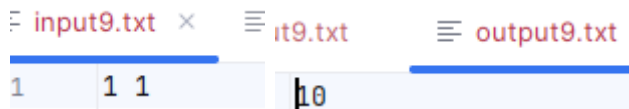
В функции `sum_dv` мы преобразуем два числа (a , b) в списки из нулей и единиц. Затем создаем список из нулей – c , его длина равна $n + 1$, где n – длина списка a (a и b одинаковой длины по условию, поэтому можем измерить длину любого из получившихся списков). В цикле мы суммируем значения с конца массивов и записываем это число в переменную s . В правый элемент c мы записываем остаток от деления s на 2, а в левый элемент c (следующий) мы записываем результат целочисленного деления

с на 2. Преобразуем с в строку, запишем её в переменную result и срежем все нули, которые стоят в начале строки. Для этого найдем индекс первого вхождения единицы и сделаем от него срез до конца строки.

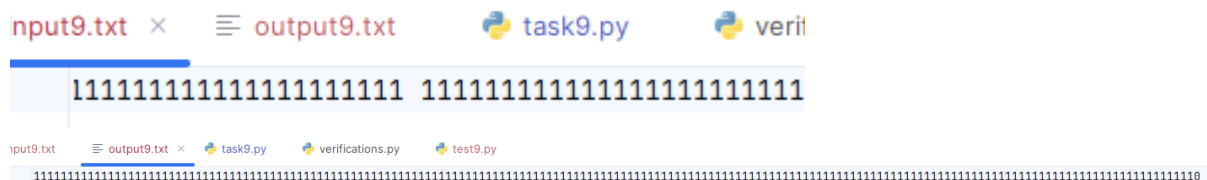
Сложность алгоритма – $O(n)$.

Функция data_verification9 проверяет длину чисел в двоичной записи. Длины должны быть равны и больше единицы, но меньше 1000 (включая границы). А символы в двоичной записи должны быть 0 или 1.

Результат работы кода на минимальных значениях:



Результат работы кода на максимальных значениях:



Время выполнения и затраты памяти:

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 1	0.00001	19.72
111000 010101	0.000018	19.95
1010101... 1001011...	0.000407	19.96

Вывод по задаче: мы написали алгоритм для сложения двух чисел в двоичной системе счисления, повторили, как измерять время выполнения и затраты памяти, а также, как определять асимптотическую сложность алгоритма.

Дополнительные задачи

Задача №2. Сортировка вставкой + [N баллов]

Текст задачи: измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

Решение:

```
from lab1.src.verifications import data_verification1
```

```
@data_verification1
def insertion_sort(n, m):
```

```

indexes = [1]
for i in range(1, n):
    for j in range(i, -1, -1):
        if m[i] < m[j]:
            m[i], m[j] = m[j], m[i]
            i, j = j, i
    indexes.append(i+1)
return indexes, m

file_input = open("input2.txt")
file_output = open("output2.txt", "w")

test_n = int(file_input.readline())
test_m = list(map(int, file_input.readline().split(" ")))

res_ind, res_m = insertion_sort(test_n, test_m)

file_output.write(" ".join(map(str, res_ind)))
file_output.write("\n")
file_output.write(" ".join(map(str, res_m)))

```

Объяснение решения:

Функция сортирует массив тем же способом, что и функции из задач 1 и 3. Помимо сортировки мы создадим массив с индексами – `indexes`. Он изначально содержит один элемент – единицу – это индекс для первого элемента, для следующих элементов мы будем добавлять индекс `i+1`.

Результат работы на минимальных значениях:

input2.txt ×	output2.txt ×
1	1
0	0

Результат работы на значениях из условия:

input2.txt ×	output2.txt	u1z.txt	= outputz.txt ^
10	1 8 4 2 3 7 5 6 9 0	1 2 2 2 3 5 5 6 9 1	0 1 2 3 4 5 6 7 8 9

Время выполнения и затраты памяти:

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.0000078	19.75
10 1 8 4 2 3 7 5 6 9 0	0.000017	19.59
1000 209323393 -541998062 -128173336	0.0186	19.82

Вывод по задаче: мы дополнили алгоритм сортировки вставкой – реализовали запись индексов переставленных элементов.

Задача №4. Линейный поиск [N баллов]

Текст задачи:

- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V.
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

Решение:

```
-----
from lab1.src.verifications import data_verification4

@data_verification4
def lineal_search(m, x):
    res = []
    for i in range(len(m)):
        if m[i] == x:
            res.append(i)
    count = len(res)
    if count == 0:
        return str(-1)
    elif count == 1:
        return str(res[0])
    else:
        return str(count) + ' ' + ", ".join(map(str, res))

file = open("input4.txt")
A = list(map(int, file.readline().split(" ")))
V = int(file.readline())
open("output4.txt", "w").write(lineal_search(A, V))

-----

def data_verification4(func):
    def check(M, V, repeat=0):
        if len(M) > 0 and type(M[0]) == str:
            return func(M, V)
        else:
            if len(M) <= 10**3 and all(abs(x) <= 10**3 for x in M) and
abs(V) <= 10**3 :
                return func(M, V)
            else:
                if repeat >= 2:
                    return "Invalid data"
                else:
                    print("Enter data again")
                    M = [int(i) for i in input().split()]
```

```

V = int(input())
return check(M, V, repeat+1)

return check

```

Объяснение решения:

Функция `lineal_search` принимает в качестве аргументов список и искомый элемент, затем проходится в цикле по списку и сравнивает i -ый элемент с искомым, если элементы равны, то функция добавляет этот индекс в массив `res`. Если длина этого массива равна нулю, то есть элемент не найден, то функция вернет -1. Если длина равна 1, то функция вернет только единственный подходящий индекс. Если мы нашли несколько таких элементов, то есть длина `res` больше единицы, то выведем, сколько раз встречается элемент, и все индексы его вхождения.

Функция `data_verification4` проверяет значение аргументов. Для того, чтобы проверить работу функции на примере списка животных, напишем условие: если длина списка больше нуля и первый элемент – строка, то мы не будем проверять список на ограничения. Если это не так, что длина списка должна быть меньше или равна 1000 и все элементы по модулю также не превосходят 1000.

Результат работы на минимальных значениях:

The screenshot shows a code editor with two files: `input4.txt` and `output4.txt`. `input4.txt` contains the numbers 0 and 4 on two lines. `output4.txt` contains the number 1 on the first line.

Результат работы на примере:

The screenshot shows a code editor with two files: `input4.txt` and `output4.txt`. `input4.txt` contains the list of numbers 98 5 3 -9 0 4 5 6 7 2 5 6 -4 on the first line and the number 5 on the second line. `output4.txt` contains the indices 3 1, 6, 10 on the first line.

Результат работы на максимальных значениях:

The screenshot shows a code editor with two files: `input4.txt` and `output4.txt`. `input4.txt` contains the list of numbers 136 692 -973 -565 399 on the first line and the number 1000 on the second line. `output4.txt` contains the indices 427, 449 on the first line.

Время выполнения и затраты памяти:

	Время выполнения (в секундах)	Затраты памяти (в МБ)

... 0	0.0000075	19.78
Рак Кабан Козел Лемминг... Свинья	0.000035	19.60
136 692 -973 -565 399 - 813... 1000	0.00009	19.93

Вывод по задаче: мы научились писать алгоритм линейного поиска, проанализировали его время работы и затраты памяти, а также убедились в корректности его работы на тестах, убедились, что алгоритм за небольшое время справляется с поиском всех вхождений элемента в массиве из 1000 элементов.

Задача №10. Палиндром [N баллов]

Текст задачи: палиндром — это строка, которая читается одинаково как справа налево, так и слева направо. На вход программы поступает набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы. Требуется из данных букв по указанным правилам составить палиндром наибольшей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

Решение:

```
-----
from lab1.src.verifications import data_verification10
```

```
def merge_sort(string):
    n = len(string)
    m = n // 2
    arr1 = string[:m]
    arr2 = string[m:]
    n1, n2 = m, n-m
    if n1 > 1:
        arr1 = merge_sort(arr1)
    if n2 > 1:
        arr2 = merge_sort(arr2)
    return merge_list(arr1, arr2, n1, n2)
```

```
def merge_list(arr1, arr2, n1, n2):
    res_str = ""
    i1, i2 = 0, 0
    while i1 < n1 and i2 < n2:
        if arr1[i1] >= arr2[i2]:
```

```

        res_str += arr1[i1]
        i1 += 1
    else:
        res_str += arr2[i2]
        i2 += 1
    res_str += arr1[i1:] + arr2[i2:]
    return res_str

@data_verification10
def palindrome(n: int, s: str, index=0):
    s = merge_sort(s)
    count1 = ""
    count2 = ""
    while index < n:
        k = s[index]
        count = 1
        index += 1
        while index < n:
            if s[index] == k:
                count += 1
                index += 1
            else:
                break
        count1 = k * (count % 2) + count1
        count2 += k * (count // 2) * 2
    return palindrome2(count1, count2)

def palindrome2(count1, count2):
    if len(count1) > 0:
        res_str = count1[0]
    else:
        res_str = ""
    pairs_count = len(count2)
    for i in range(0, pairs_count, 2):
        res_str = count2[i] + res_str + count2[i+1]
    return res_str
-----

def data_verification10(func):
    def check(n, s, repeat=0):
        if 1 <= n <= 10**5 and all(65 <= ord(x) <= 90 for x in s) and
len(s) == n:
            return func(n, s)
        else:
            if repeat >= 2:
                return "Invalid data"
            else:
                print("Enter data again")
                n = int(input())
                s = input()
                return check(n, s, repeat+1)
    return check
-----

```

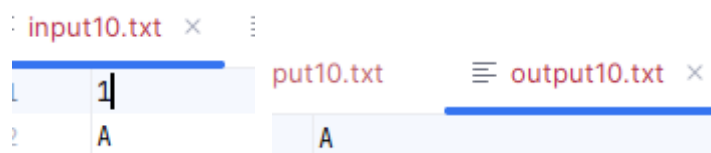
Объяснение решения:

Для того, чтобы собрать полином в алфавитном порядке, отсортируем строку. Напишем для этого сортировку слиянием: в функции `merge_sort` будем делить список пополам, пока не останется один элемент, а в функции `merge_list` будем сливать эти два списка так, чтобы получилась строка, отсортированная в обратном алфавиту порядке. Это нужно для того, чтобы

при формировании палиндрома по краям были первые буквы алфавита, а в середине – последние. В функции `palindrome` мы разберем отсортированную строку на две строки: в `count1` запишем буквы, для которых не нашлось пары, а в `count2` будем записывать все пары букв. Проходимся в цикле по строке и записываем элемент с индексом `= index` в переменную `k` (количество таких элементов – переменная `count`, изначально равна единице). Затем проходимся далее по массиву во вложенном цикле. Если мы нашли еще один элемент равный `k`, тогда увеличиваем индекс и количество на единицу. Если мы дошли до элемента не равного `k`, тогда цикл прекращается и все пары записываются в `count2`, а оставшийся элемент в `count1`. Функция `palindrome2` формирует палиндром: если у нас остались одиночные элементы, то первый из них (в алфавитном порядке) мы запишем в переменную `res_str`. Затем с обеих сторон строки с результатом будем присоединять пары, отсортированные в обратном алфавитном порядке. В итоге функция возвращает переменную `res_str`.

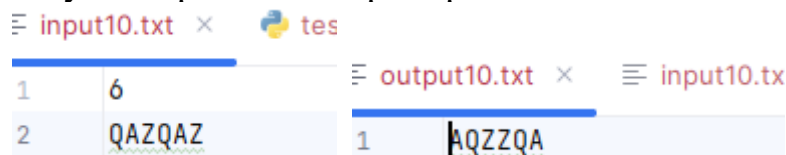
Также введенные значения проверяет функция `data_verification10`: она проверяет, чтобы длина строки была равна `n` и `n <= 10^5`, а также каждый символ из строки в таблице ASCII должен иметь номер от 65 (`=ord("A")`) до 90 (`=ord("Z")`).

Результат работы при минимальных значениях:



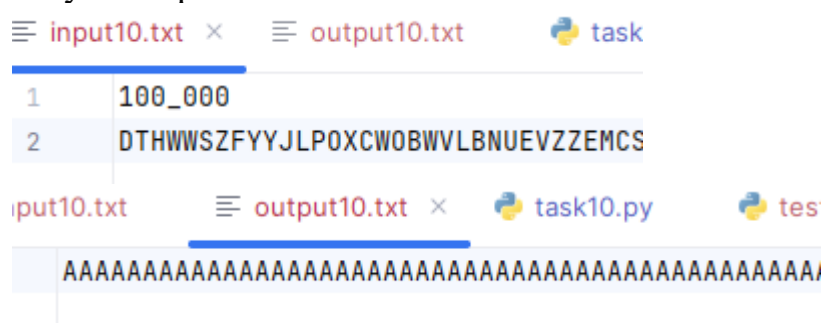
The screenshot shows a code editor with three tabs: `input10.txt`, `put10.txt`, and `output10.txt`. The `input10.txt` tab is active and shows two lines: `1` and `A`. The `output10.txt` tab shows the result `A`.

Результат работы на примере из задания:



The screenshot shows a code editor with three tabs: `input10.txt`, `output10.txt`, and `input10.tx`. The `input10.txt` tab is active and shows two lines: `6` and `QAZQAZ`. The `output10.txt` tab shows the result `AQZZQA`.

Результат работы на максимальных значениях:



The screenshot shows a code editor with three tabs: `input10.txt`, `output10.txt`, and `task10.py`. The `input10.txt` tab is active and shows two lines: `100_000` and a long string of letters: `DTHWWSZFYYJLPXCWOBWVLBNUEVZZEMCS`. The `output10.txt` tab shows a long string of 'A's.

Время выполнения и затраты памяти:

	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 A	0.000013	19.71

6 ABCDEF	0.000018	19.71
100000 DTHWWSZFYJLP OXCWOBWVLBNUEVZZ...	0.27694	20.43

Вывод по задаче: мы научились сортировать строки с помощью сортировки слиянием, а также написали эффективный алгоритм для составления палиндрома из букв английского алфавита.

Вывод

В ходе лабораторной работы мы вспомнили, как работать с файлами, как писать модульные тесты, как измерять затраты памяти и время работы алгоритма, а также изучили и реализовали простые алгоритмы сортировки и поиска элементов в массиве, научились решать нестандартные задачи и использовать декораторы.