

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 14

Выполнила:
Рудникова В.О.
К3125

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Название задачи [N баллов]	3
Дополнительные задачи	4
Задача №1. Название задачи [N баллов]	4
Вывод	5

Задачи по варианту

Задача №1. Сортировка вставкой

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

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 10^3$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
def insertion_sort(l):
    for i in range(len(l)):
        for j in range(i):
            if l[j] > l[i]:
                l[j], l[i] = l[i], l[j]
    return l
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(' '.join(str(i) for i in insertion_sort(a)))
```

Тест:

```
import unittest
from task1 import insertion_sort
class MyTestCase(unittest.TestCase):
    def test_min(self):
        a1 = [i for i in range(11)]
        a1[-1], a1[-2] = a1[-2], a1[-1]
        a2 = [i for i in range(101)]
        a2[-1], a2[-2] = a2[-2], a2[-1]
        a3 = [i for i in range(1001)]
        a3[-1], a3[-2] = a3[-2], a3[-1]
        self.assertEqual(insertion_sort(a1), sorted(a1))
        self.assertEqual(insertion_sort(a2), sorted(a2))
        self.assertEqual(insertion_sort(a3), sorted(a3))
    def test_average(self):
        a1 = [i for i in range(6)]
        a1.extend([i for i in range(10, 5, -1)])
        a2 = [i for i in range(6)]
        a2.extend([i for i in range(10, 5, -1)])
        a3 = [i for i in range(6)]
        a3.extend([i for i in range(10, 5, -1)])
        self.assertEqual(insertion_sort(a1), sorted(a1))
        self.assertEqual(insertion_sort(a2), sorted(a2))
        self.assertEqual(insertion_sort(a3), sorted(a3))
    def test_max(self):
        a1 = [i for i in range(10, 0, -1)]
        a2 = [i for i in range(100, 0, -1)]
        a3 = [i for i in range(1000, 0, -1)]
        self.assertEqual(insertion_sort(a1), sorted(a1))
        self.assertEqual(insertion_sort(a2), sorted(a2))
        self.assertEqual(insertion_sort(a3), sorted(a3))
if __name__ == '__main__':
    unittest.main()
```

Для каждого элемента списка $a[i]$ просматриваются все элементы до него, и при нахождении такого $a[j]$, который больше рассматриваемого $a[i]$, эти два элемента меняются местами.

Результат работы кода на примерах из текста задачи:

input.txt	
1	6
2	31 41 59 26 41 58
output.txt	
1	26 31 41 41 58 59

Результат работы кода на максимальных и минимальных значениях:

input.txt	
1	1000 ✓
2	1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986 985 984 983 982 981
output.txt	
1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 ✓

input.txt	
1	8
2	1 2 3 4 5 6 8 7
output.txt	
1	1 2 3 4 5 6 7 8

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.09375	1076 В
Пример из задачи	0.109375	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.078125	28.9 KiB

Вывод по задаче: в данной задаче я использовала алгоритм сортировки вставкой.

Задача №4. Линейный поиск

Рассмотрим задачу поиска.

- Формат входного файла. Последовательность из n чисел $A = a_1, a_2, \dots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 10^3$, $-10^3 \leq a_i$, $V \leq 10^3$
- Формат выходного файла. Одно число - индекс i , такой, что $V = A[i]$, или значение -1 , если V в отсутствует.
- 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.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    a = [int(i) for i in f.readline().split()]
    v = int(f.readline())
```

```

def linear_search(l, x):
    idxs = []
    for i in range(len(l)):
        if l[i] == x:
            idxs.append(i)
    return idxs
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    if len(linear_search(a, v)) == 0:
        g.write('-1')
    elif len(linear_search(a, v)) == 1:
        g.write(str(linear_search(a, v)[0]))
    else:
        g.write(str(len(linear_search(a, v))) + '\n' + ',
'.join(str(i) for i in linear_search(a, v)))

```

Tecr:

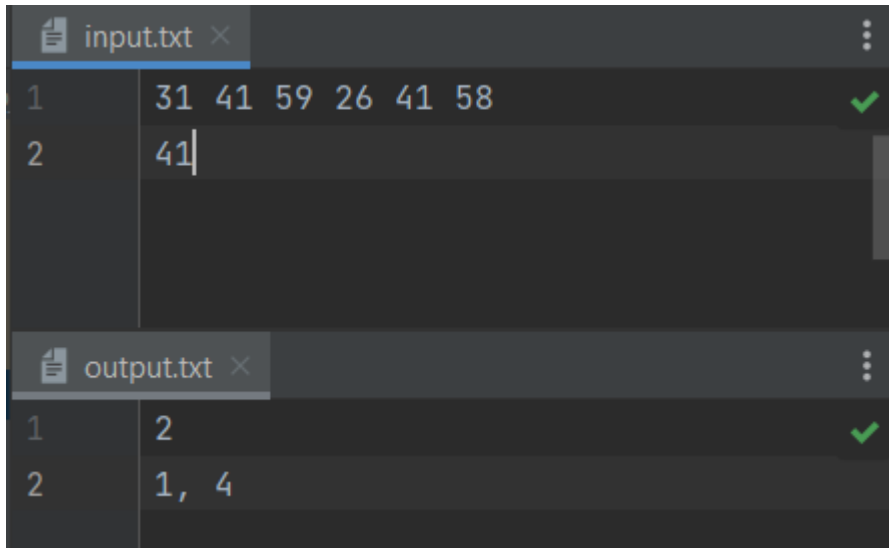
```

import unittest
from task4 import linear_search
def search(l, v):
    idxs = []
    while v in l:
        idxs.append(l.index(v))
        l[l.index(v)] = v + 1
    return idxs
class MyTestCase(unittest.TestCase):
    def test(self):
        a1 = [i for i in range(11)]
        v1 = 1000
        a2 = [i for i in range(101)]
        v2 = 13
        a3 = [i for i in range(901)]
        a3.extend([100] * 100)
        v3 = 100
        self.assertEqual(linear_search(a1, v1), search(a1, v1))
        self.assertEqual(linear_search(a2, v2), search(a2, v2))
        self.assertEqual(linear_search(a3, v3), search(a3, v3))
if __name__ == '__main__':
    unittest.main()

```

В цикле проходимся по списку и сравниваем каждый его элемент с искомым. Если нужное значение найдено, сохраняем индекс элемента в отдельный список и считаем количество раз, которое встретилось нужное нам число.

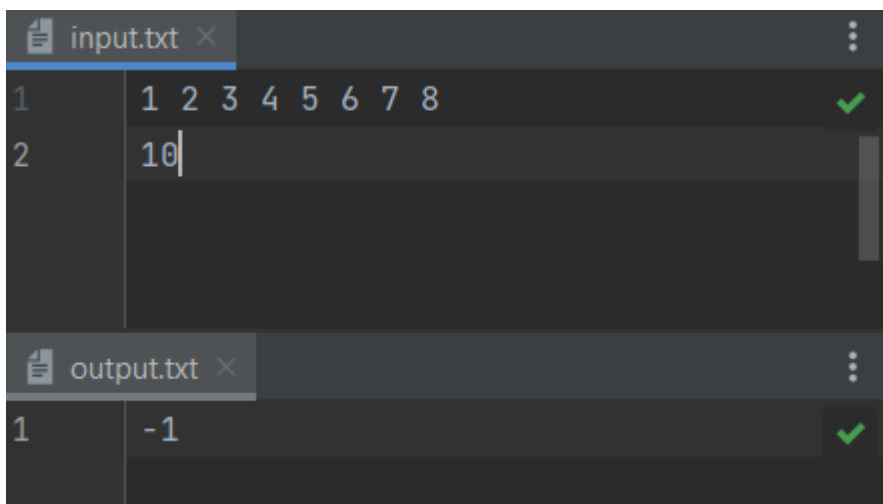
Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with two files: `input.txt` and `output.txt`. The `input.txt` file contains two lines: the first line has the numbers 31, 41, 59, 26, 41, 58, and the second line has the number 41. The `output.txt` file contains two lines: the first line has the number 2, and the second line has the numbers 1, 4. Green checkmarks are visible at the end of each line in both files, indicating successful execution.

File	Line	Content
input.txt	1	31 41 59 26 41 58
	2	41
output.txt	1	2
	2	1, 4

Результат работы кода на максимальных и минимальных значениях:



The screenshot shows a code editor with two files: `input.txt` and `output.txt`. The `input.txt` file contains two lines: the first line has the numbers 1, 2, 3, 4, 5, 6, 7, 8, and the second line has the number 10. The `output.txt` file contains one line: the number -1. Green checkmarks are visible at the end of each line in both files, indicating successful execution.

File	Line	Content
input.txt	1	1 2 3 4 5 6 7 8
	2	10
output.txt	1	-1


```

input.txt
1 999 999 999 999 999 999 999 999 99 ✓
2 999

output.txt
1 1000 ✓
2 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.078125	1076 В
Пример из задачи	0.09375	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.078125	35.9 KiB

Вывод по задаче: в задаче я реализовала алгоритм линейного поиска.

Задача №6. Пузырьковая сортировка

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq \dots \leq A'[n]$, где A' - выход процедуры Bubble Sort, n - длина массива A . Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой. Формат входного и выходного файла и ограничения - как в задаче 1.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('input.txt') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
def bubble_sort(l):
    for i in range(len(l)):
        for j in range(len(l) - 1, i, -1):
            if l[j - 1] > l[j]:
                l[j - 1], l[j] = l[j], l[j - 1]
    return l
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('output.txt', 'w+') as g:
    g.write(' '.join(str(i) for i in bubble_sort(a)))
# доказательство корректности
for i in range(len(a) - 1):
    print(f'a[{i}] <= a[{i + 1}] - {bubble_sort(a)[i] <=
bubble_sort(a)[i + 1]}')
```

Тест:

```
import unittest
from task6 import bubble_sort
class MyTestCase(unittest.TestCase):
    def test_min(self):
        a1 = [i for i in range(11)]
        a1[-1], a1[-2] = a1[-2], a1[-1]
        a2 = [i for i in range(101)]
        a2[-1], a2[-2] = a2[-2], a2[-1]
        a3 = [i for i in range(1001)]
        a3[-1], a3[-2] = a3[-2], a3[-1]
        self.assertEqual(bubble_sort(a1), sorted(a1))
        self.assertEqual(bubble_sort(a2), sorted(a2))
        self.assertEqual(bubble_sort(a3), sorted(a3))
    def test_average(self):
        a1 = [i for i in range(6)]
        a1.extend([i for i in range(10, 5, -1)])
        a2 = [i for i in range(6)]
        a2.extend([i for i in range(10, 5, -1)])
        a3 = [i for i in range(6)]
```

```

        a3.extend([i for i in range(10, 5, -1)])
        self.assertEqual(bubble_sort(a1), sorted(a1))
        self.assertEqual(bubble_sort(a2), sorted(a2))
        self.assertEqual(bubble_sort(a3), sorted(a3))

    def test_max(self):
        a1 = [i for i in range(10, 0, -1)]
        a2 = [i for i in range(100, 0, -1)]
        a3 = [i for i in range(1000, 0, -1)]
        self.assertEqual(bubble_sort(a1), sorted(a1))
        self.assertEqual(bubble_sort(a2), sorted(a2))
        self.assertEqual(bubble_sort(a3), sorted(a3))

if __name__ == '__main__':
    unittest.main()

```

Для каждого из элементов списка пересматриваются все оставшиеся в обратном порядке. Если выбранный элемент больше своего правого соседа, они меняются местами.

Результат работы кода на примерах из текста задачи:

```

input.txt ×
1 6 ✓
2 31 41 59 26 41 58

output.txt ×
1 26 31 41 41 58 59 ✓

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt ×
1 10 ✓
2 1 2 3 4 5 6 7 8 10 9

output.txt ×
1 1 2 3 4 5 6 7 8 9 10 ✓

```

```

input.txt x
1 1000 ✓
2 1000 999 998 997 996 995 994 993 992

output.txt x
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 1 ✓

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.078125	1070 B
Пример из задачи	0.078125	1070 B
Верхняя граница диапазона значений входных данных из текста задачи	0.078125	28.9 KiB

Вывод по задаче: В этой задаче я использовала алгоритм сортировки пузырьком. В среднем и худшем случаях время выполнения сортировки пузырьком и вставками отличается незначительно.

Дополнительные задачи

Задача №2. Сортировка вставкой +

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.

- Формат выходного файла (input.txt). В первой строке выходного файла выведите n чисел. При этом i -ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i -ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
    b = a.copy()
idxs = [i for i in range(len(a))]
for i in range(len(a)):
    for j in range(i):
        if a[j] > a[i]:
            a[j], a[i] = a[i], a[j]
for i in range(len(b)):
    for j in range(len(a)):
        if a[i] == b[j]:
            idxs[j] = i
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(' '.join(str(i + 1) for i in idxs) + '\n' + ' '.join(str(i) for i in a))
```

Реализуется алгоритм сортировки вставками (см. задачу 1). Для определения новых индексов элементов создаётся копия изначального списка и производится линейный поиск каждого элемента нового списка в копии, индекс сохраняется.

Результат работы кода на примерах из текста задачи:

```
input.txt ×
1 10 ✓
2 1 8 4 2 3 7 5 6 9 0

output.txt ×
1 2 9 5 3 4 8 6 7 10 1 ✓
2 0 1 2 3 4 5 6 7 8 9
```

Результат работы кода на максимальных и минимальных значениях:

```
input.txt ×
1 1000 ✓
2 1000 999 998 997 996 995 994 993 992

output.txt ×
1 1000 999 998 997 996 995 994 993 9 ✓
2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1
```

```
input.txt ×
1 10 ✓
2 1 2 3 4 5 6 7 8 10 9

output.txt ×
1 1 2 3 4 5 6 7 8 10 9 ✓
2 1 2 3 4 5 6 7 8 9 10
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375	2208 В
Пример из задачи	0.078125	2208 В
Верхняя граница диапазона значений входных данных из текста задачи	0.734375	28.9 KiB

Вывод по задаче: в этой задаче я использовала сортировку вставками и линейный поиск.

Задача №3. Сортировка вставкой по убыванию

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

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
for i in range(len(a)):
    for j in range(i):
        if a[j] < a[i]:
            a[j], a[i] = a[i], a[j]
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(' '.join(str(i) for i in a))
```

Используется алгоритм сортировки вставкой (см. задачу 1), но знак “>” при сравнении значений меняется на “<”.

Результат работы кода на примерах из текста задачи:

```

input.txt x
1 6 ✓
2 31 41 59 26 41 58

output.txt x
1 59 58 41 41 31 26 ✓

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt x
1 1000 ✓
2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1

output.txt x
1 1000 999 998 997 996 995 994 993 9 ✓

```

```

input.txt x
1 10 ✓
2 1 2 3 4 5 6 7 8 10 9

output.txt x
1 10 9 8 7 6 5 4 3 2 1 ✓

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.09375	1076 В

входных данных из текста задачи		
Пример из задачи	0.078125	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.328125	28.9 KiB

Вывод по задаче: в задаче был использован алгоритм сортировки вставкой.

Задача №5. Сортировка выбором

Рассмотрим сортировку элементов массива, которая выполняется следующим образом. Сначала определяется наименьший элемент массива, который ставится на место элемента $A[1]$. Затем производится поиск второго наименьшего элемента массива A , который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n - 1$ элементов массива A . Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой. Формат входного и выходного файла и ограничения - как в задаче 1.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
def selection_sort(l):
    minn = 10 ** 9 + 1
    for i in range(len(l)):
        for j in range(i, len(l)):
            if l[j] < minn:
                minn = l[j]
            l[j], l[i] = l[i], l[j]
    minn = 10 ** 9 + 1
```

```

    return l
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(' '.join(str(i) for i in a))

```

Tecr:

```

import unittest
from task5 import selection_sort
class MyTestCase(unittest.TestCase):
    def test_min(self):
        a1 = [i for i in range(11)]
        a1[-1], a1[-2] = a1[-2], a1[-1]
        a2 = [i for i in range(101)]
        a2[-1], a2[-2] = a2[-2], a2[-1]
        a3 = [i for i in range(1001)]
        a3[-1], a3[-2] = a3[-2], a3[-1]
        self.assertEqual(selection_sort(a1), sorted(a1))
        self.assertEqual(selection_sort(a2), sorted(a2))
        self.assertEqual(selection_sort(a3), sorted(a3))
    def test_average(self):
        a1 = [i for i in range(6)]
        a1.extend([i for i in range(10, 5, -1)])
        a2 = [i for i in range(6)]
        a2.extend([i for i in range(10, 5, -1)])
        a3 = [i for i in range(6)]
        a3.extend([i for i in range(10, 5, -1)])
        self.assertEqual(selection_sort(a1), sorted(a1))
        self.assertEqual(selection_sort(a2), sorted(a2))
        self.assertEqual(selection_sort(a3), sorted(a3))
    def test_max(self):
        a1 = [i for i in range(10, 0, -1)]
        a2 = [i for i in range(100, 0, -1)]
        a3 = [i for i in range(1000, 0, -1)]
        self.assertEqual(selection_sort(a1), sorted(a1))
        self.assertEqual(selection_sort(a2), sorted(a2))
        self.assertEqual(selection_sort(a3), sorted(a3))
if __name__ == '__main__':
    unittest.main()

```

Линейным поиском находится минимальный элемент списка и ставится на первое место. Далее находится второй минимальный элемент, становится на второе место и так далее, пока очередь не дойдёт до последнего элемента.

Результат работы кода на примерах из текста задачи:

```
input.txt
1 6
2 31 41 59 26 41 58

output.txt
1 26 31 41 41 58 59
```

Результат работы кода на максимальных и минимальных значениях:

```
input.txt
1 1000
2 1000 999 998 997 996 995 994 993 992

output.txt
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 1
```

```
input.txt
1 10
2 1 2 3 4 5 6 7 8 10 9

output.txt
1 1 2 3 4 5 6 7 8 9 10
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375	1076 В
Пример из задачи	0.0625	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.109375	28.9 KiB

Вывод по задаче: в задаче я реализовала алгоритм сортировки выбором.

Задача №7. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет n , где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем. Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n . Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i , содержится в ячейке $M[i]$. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

- Формат входного файла (input.txt). Первая строка входного файла содержит число жителей n ($3 \leq n \leq 9999$, n нечетно). Вторая строка содержит описание массива M , состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения имеют точность не более двух знаков после запятой и не превышают 10^6 .

- Формат выходного файла (output.txt). В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt', 'r') as f:
    n = int(f.readline())
    a = [float(i) for i in f.readline().split()]
    idxs = [i for i in range(len(a))]
    for i in range(len(a)):
        for j in range(i):
            if a[j] > a[i]:
                a[j], a[i] = a[i], a[j]
                idxs[j], idxs[i] = idxs[i], idxs[j]
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(str(idxs[0] + 1) + ' ' + str(idxs[n // 2] + 1) + ' ' + str(idxs[n-1] + 1))
```

Имеется два списка: в одном хранятся элементы, в другом - их изначальные индексы. Основной список сортируется (я использую сортировку вставкой), каждый элемент списка индексов становится на место, равное новому индексу соответствующего элемента. Из списка индексов выводится первый, средний и последний элемент.

Результат работы кода на примерах из текста задачи:

```
input.txt x
1 5 ✓
2 10.00 8.70 0.01 5.00 3.00
output.txt x
1 3 4 1 ✓
```

Результат работы кода на максимальных и минимальных значениях:

```
input.txt x
1 1000 ✓
2 1000 999 998 997 996 995 994 993 992
output.txt x
1 1000 500 1 ✓
```

```
input.txt x
1 10 ✓
2 1 2 3 4 5 6 7 8 10 9
output.txt x
1 1 6 9 ✓
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.125	1076 В
Пример из задачи	0.109375	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.40625	31.9 KiB

Вывод по задаче: в задаче я использовала алгоритм сортировки вставкой, применяя его одновременно для списка элементов и списка индексов.

Задача №8. Секретарь Своп

Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своп — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своп смог проверить Вашу работу.

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($3 \leq n \leq 5000$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9 . Числа могут совпадать друг с другом.
- Формат выходного файла (output.txt). В первых нескольких строках выведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат: Swap elements at indices X and Y . Здесь X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \leq X, Y \leq n$). Мистер Своп любит порядок, поэтому сделайте так, чтобы $X < Y$. После того, как все нужные перестановки выведены, выведите следующую фразу: No more swaps needed.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt') as f:
    n = int(f.readline())
    a = [int(i) for i in f.readline().split()]
g = open('../output.txt', 'w+')
for i in range(len(a)):
    for j in range(len(a)-1, i, -1):
        if a[j-1] > a[j]:
            a[j-1], a[j] = a[j], a[j-1]
            g.write(f'Swap elements at indices {j-1} and {j}.\n')
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
g.write('No more swaps needed.')
g.close()
```

Используется алгоритм сортировки пузырьком, при этом при каждом обмене элементов в файл выводится сообщение об этом с указанием индексов меняющихся элементов.

Результат работы кода на примерах из текста задачи:

```
input.txt x
1 5 ✓
2 3 1 4 2 2

output.txt x
1 Swap elements at indices 2 and 3. ✓
2 Swap elements at indices 0 and 1.
3 Swap elements at indices 3 and 4.
4 Swap elements at indices 1 and 2.
5 Swap elements at indices 2 and 3.
6 No more swaps needed.
```

Результат работы кода на максимальных и минимальных значениях:

```
input.txt x
1 1000 ✓
2 1000 999 998 997 996 995 994 993 992
3

output.txt x
The file size (19,47 MB) exceeds the configured limit (2,56 MB)
499492 Swap elements at indices 997 and 998 ✓
499493 Swap elements at indices 996 and 997
499494 Swap elements at indices 995 and 996
499495 Swap elements at indices 998 and 997
499496 Swap elements at indices 997 and 996
499497 Swap elements at indices 996 and 995
499498 Swap elements at indices 998 and 997
499499 Swap elements at indices 997 and 996
499500 Swap elements at indices 998 and 997
499501 No more swaps needed.
```

The screenshot shows a code editor with two files: `input.txt` and `output.txt`.
 In `input.txt`, line 1 contains the number `10` and line 2 contains the sequence `1 2 3 4 5 6 7 8 10 9`.
 In `output.txt`, line 1 contains the text `Swap elements at indices 8 and 9.` and line 2 contains `No more swaps needed.`.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.078125	9555 B
Пример из задачи	0.125	9555 B
Верхняя граница диапазона значений входных данных из текста задачи	0.71875	28.9 KiB

Вывод по задаче: в задаче я использовала алгоритм сортировки пузырьком с выводом сообщения для каждого элемента.

Задача №9. Сложение двоичных чисел

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.

- Формат входного файла (`input.txt`). В одной строке содержится два n -битовых двоичных числа, записанные через пробел ($1 \leq n \leq 103$)

- Формат выходного файла (output.txt). Одна строка - двоичное число, которое является суммой двух чисел из входного файла.
- Оцените асимптотическое время выполнения вашего алгоритма.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt') as f:
    a = [int(i) for i in f.readline().split()]
    b = [int(i) for i in f.readline().split()]
def binary_sum(l1, l2):
    res = [0] * (len(l1) + 1)
    for i in range(len(l1) - 1, -1, -1):
        res[i + 1] += l1[i] + l2[i]
        if res[i + 1] > 1:
            res[i + 1] -= 2
            res[i] += 1
    return res
print(process_time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(' '.join(str(i) for i in binary_sum(a,
b)))
```

Тест:

```
import unittest
from task9 import binary_sum
class MyTestCase(unittest.TestCase):
    def test1(self):
        a = [int(i) for i in bin(5)[2:]]
        print(a)
        b = [int(i) for i in bin(7)[2:]]
        print(b)
        self.assertEqual(binary_sum(a, b), [int(i) for i in
bin(5 + 7)[2:]])
    def test2(self):
```

```

        a = [int(i) for i in bin(56)[2:]]
        print(a)
        b = [int(i) for i in bin(33)[2:]]
        print(b)
        self.assertEqual(binary_sum(a, b), [int(i) for i in
bin(56 + 33)[2:]])
    def test3(self):
        a = [int(i) for i in bin(3001)[2:]]
        print(a)
        b = [int(i) for i in bin(2076)[2:]]
        print(b)
        self.assertEqual(binary_sum(a, b), [int(i) for i in
bin(3001 + 2076)[2:]])
    def test4(self):
        a = [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,

```

```

1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0,
0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0]
b = [1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,

```

```

0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0,
1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0,
1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0,
0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1,
1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1,
0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,

```

```

c = [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1,
1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,

```

```

1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1]

self.assertEqual(binary_sum(a, b), c)

if __name__ == '__main__':
    unittest.main()

```

В одном цикле перебираются с конца оба списка, третий заполняется следующим образом: его i -й элемент приравнивается к сумме соответствующих элементов изначальных списков, и от него отнимается 2, если необходимо (при этом, в свою очередь, $i-1$ -й элемент увеличивается на 1).

Результат работы кода на примерах из текста задачи:

```

input.txt x
1 1 1 0 1 1 1 1 0 ✓
2 1 0 0 0 0 0 1 1

output.txt x
1 1 0 1 1 0 0 0 1 ✓

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt x
1 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 ✓
2 1 1 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0 0 0

output.txt x
1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 ✓

```

```

input.txt x
1 1 ✓
2 1

output.txt x
1 1 0 ✓

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из	0.09375	1076 В

текста задачи		
Пример из задачи	0.09375	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.09375	8800 В

Вывод по задаче: в этой задаче я вспомнила принцип сложения в столбик и работы с двоичными числами и реализовала это в программе. Асимптотическая сложность такой программы - $O(n)$.

Задача №10. Палиндром

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.

- Формат входного файла (input.txt). В первой строке входных данных содержится число n ($1 \leq n \leq 100000$). Во второй строке задается последовательность из n больших латинских букв (буквы записаны без пробелов).
- Формат выходного файла (output.txt). В единственной строке выходных данных выдайте искомый палиндром.

Код:

```
from time import process_time
from tracemalloc import *
start()
with open('../input.txt') as f:
    n = int(f.readline())
    s = [i for i in f.readline()]
def palindrome(s1):
    s2 = ''
    s3 = ''
```

```

minn = 'a'
for i in range(len(s)):
    for j in range(i, len(s)):
        if s[j] < minn:
            minn = s[j]
            s[j], s[i] = s[i], s[j]
    minn = 'a'
    for i in range(len(s1)):
        if s1.count(s1[i]) % 2 != 0 and s2.count(s1[i]) <
1:
            s2 += s1[i]
        if s2 != '':
            for i in s1[::-1]:
                if s1.count(i) % 2 != 0 and s2[0] != i:
                    s1.remove(i)
            s1.remove(s2[0])
        for i in range(len(s1)):
            if s3.count(s1[i]) < s1.count(s1[i]) // 2:
                s3 += s1[i]
        s4 = s3[::-1]
        if s2 != '':
            s3 += s2[0]
        s3 += s4
    return s3
print(process time())
snapshot = take_snapshot()
for stat in snapshot.statistics('lineno'):
    print(stat)
with open('../output.txt', 'w+') as g:
    g.write(palindrome(s))

```

Входная строка превращается в список символов и сортируется. После этого отбираются все символы, встретившиеся нечётное количество раз (при наличии), из них выбирается наименьший (он будет стоять в середине палиндрома). Далее в новую строку сохраняются все символы, каждый в изначальном количестве, нацело поделённом на 2 – формируется первая половина палиндрома. К первой половине прибавляется наименьший нечётный символ и вторая половина (развёрнутая первая).

Результат работы кода на примерах из текста задачи:

```
input.txt x
1 3 ✓
2 AAB

output.txt x
1 ABA ✓
```

```
input.txt x
1 6 ✓ 1 ^ v
2 QAZQAZ

output.txt x
1 AQZZQA ✓ 1 ^ v
```

```
input.txt x
1 6 ✓ 1 ^ v
2 ABCDEF

output.txt x
1 A ✓
```

Результат работы кода на максимальных и минимальных значениях:

The image shows two screenshots of a code editor interface. The first screenshot displays two files: 'input.txt' and 'output.txt'. 'input.txt' contains two lines: '1' followed by '1000' and '2' followed by a long string of uppercase letters 'XNJLUKUEARQUAMCNYGOWQWXLEPJXKCRNQCITJ'. 'output.txt' contains one line: '1' followed by a long string of 'A's followed by 'B's. The second screenshot shows the same editor with 'input.txt' containing '1' followed by '1' and '2' followed by 'A'. 'output.txt' contains one line: '1' followed by 'A'.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0625	1076 В
Пример из задачи	0.078125	1076 В
Пример из задачи	0.09375	1076 В
Пример из задачи	0.109375	1076 В
Верхняя граница диапазона значений входных данных из текста задачи	0.09375	8800 В

Вывод по задаче: в задаче я использовала сортировку выбором и вспомнила некоторые принципы работы со строками.

Вывод

В этой лабораторной я вспомнила реализацию сортировок вставками, выбором и пузырьком, а также научилась писать тесты к задачам на питоне.