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 1

Выполнил:

Толстухина Ксения Александровна

K3139

Проверил:

Афанасьев А.В.

Санкт-Петербург

<mark>2024 г</mark>

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №2. Сортировка вставкой +	6
Задача №3. Сортировка вставкой по убыванию	7
Задача №4. Линейный поиск	9
Задача №5. Сортировка выбором	12
Задача №6. Пузырьковая сортировка	14
Задача №7. Знакомство с жителями Сортлэнда	16
Задача №8. Секретарь Своп	18
Задача №9. Сложение двоичных чисел	20
Задача №10. Палиндром	22
Вывод	25

Задачи по варианту

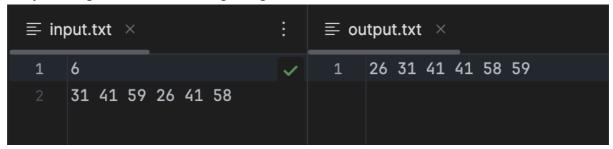
Задача №1. Сортировка вставкой

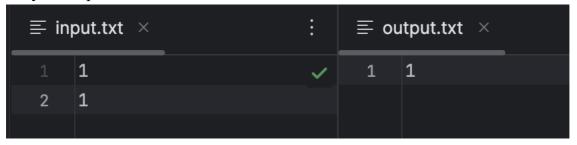
Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива A = [31, 41, 59, 26, 41, 58]

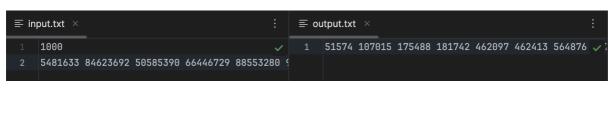
```
Листинг кода.
import time
import random
import tracemalloc
tracemalloc.start()
start = time.perf counter()
with open('input.txt', 'w') as f:
  f.write(str(1000) + \n'n')
  f.write(' '.join(map(str, random.sample(range(100000000), 1000))))
file = open('input.txt')
n = file.readline()
data = [int(i) for i in file.readline().split()]
file.close()
def insertion sort(arr):
  for i in range(1, len(arr)):
     item to insert = arr[i]
     i = i - 1
     while j \ge 0 and item to insert < arr[j]:
       arr[i + 1] = arr[i]
       j = j - 1
     arr[j + 1] = item to insert
  return arr
with open('output.txt', 'w') as file:
  file.write(''.join(map(str, insertion sort(data))))
print(time.perf counter() - start)
print(tracemalloc.get traced memory()[0] / 1024, 'MB')
```

Для начала я импортирую три библиотеки: time - для того, чтобы отследить время выполнения кода, гапdom - чтобы записать в входной файл максимально допустимое количество чисел в рандомном порядке(не вручную), tracemalloc - для отслеживания памяти, затрачиваемое в программе. Далее я записываю в файл различные числа в случайном порядке(использую этот блок кода только для проверки верхней границы). Далее я считываю данные из входного файла. Функция insertion_sort() отвечает за сортировку данных вставкой. Сортировку я начинаю со второго элемента, так как первый один - уже отсортирован. Item_to_insert - текущий элемент, j - индекс предыдущего элемента. В цикле, при условии, что j не отрицательный и до тех пор пока текущий элемент меньше предыдущего, мы сдвигаем элемент и соответственно уменьшаем j на 1, далее вставляем текущий элемент. Полученный результат записываю в выходной файл и затем вывожу время работы программы и затраченную память.

Результат работы кода на примерах из текста задачи:







Время выполнения Затра	ты памяти
------------------------	-----------

Нижняя граница диапазона значений входных данных из текста задачи	0.000868124989210628	1.404296875 MB
Пример из задачи	0.0011316250020172447	1.435546875 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.0879293339967262	38.5400390625 MB

Вывод по задаче: Я вспомнила написание одного из существующих алгоритмов сортировки и протестировала его работу на различных данных, убедилась, что скорость выполнения на данных максимальных значениях не превышает 2 секунд.

Задача №2. Сортировка вставкой +

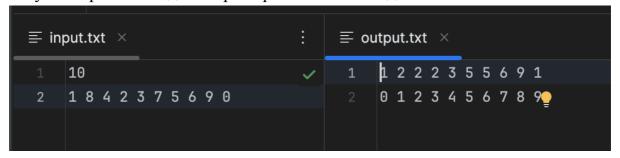
Текст задачи.

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке п чисел, которые обозначают новый индекс элемента массива после обработки.

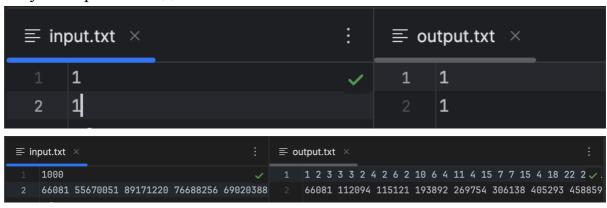
```
import time
import tracemalloc
start = time.perf counter()
tracemalloc.start()
file = open('input.txt')
n = file.readline()
a = list(map(int, file.readline().split()))
def insertion sort(arr):
  indx = [1]
  for i in range(1, len(arr)):
     item to insert = arr[i]
    j = i - 1
     while j \ge 0 and item to insert < arr[j]:
       arr[j+1] = arr[j]
       j = j - 1
     arr[i+1] = item to insert
     indx.append(arr.index(item to insert) + 1)
  return indx, arr
with open('output.txt', 'w') as f:
  fun = insertion sort(a)
  f.write(' '.join(map(str, fun[0])) + '\n')
  f.write(' '.join(map(str, fun[1])))
print(time.perf counter() - start)
print(tracemalloc.get traced memory()[0] / 1024, 'MB')
```

Проделываю все тоже самое, что и в задаче 1, только добавляю новый массив, в котором будут храниться индексы элементов.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0008054169884417206	6.4404296875 MB
Пример из задачи	0.0007612500048708171	6.65234375 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.09543116699205711	62.9697265625 MB

Вывод по задаче: задача работает так же

Задача №3. Сортировка вставкой по убыванию

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедур Swap.

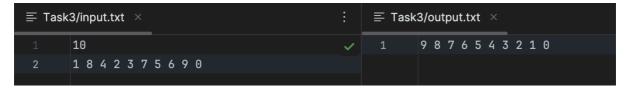
Листинг кода

```
import time
import random
import tracemalloc
tracemalloc.start()
start = time.perf counter()
with open('input.txt', 'w') as f:
  f.write(str(1000) + \n'n')
  f.write(''.join(map(str, random.sample(range(100000000), 1000))))
file = open('input.txt')
n = file.readline()
a = list(map(int, file.readline().split()))
def insertion sort(a):
  for i in range(1, len(a)):
     item to insert = a[i]
     i = i - 1
     while j \ge 0 and item to insert \ge a[j]:
       a[i+1] = a[i]
       j = j - 1
     a[j + 1] = item_to_insert
  return a
with open('output.txt', 'w') as file:
  file.write(''.join(map(str, insertion sort(a))))
print(time.perf counter() - start)
print(tracemalloc.get traced memory()[0] / 1024, 'MB')
```

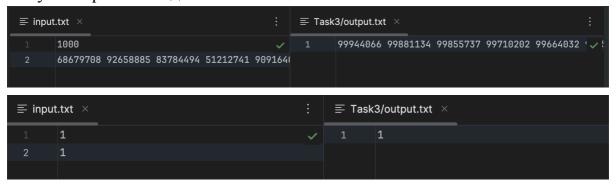
Текстовое объяснение решения

Рассуждения аналогичны задаче номер 1, единственное отличие - смена знака < на >, чтобы как раз таки отсортировать данные в противоположном порядке.

Результат работы кода на примере:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти					
Нижняя граница диапазона значений входных данных из текста задачи	0.0012026250042254105	1.677734375 MB					
Пример из задачи	0.0015490410005440935	1.7724609375 MB					
Верхняя граница диапазона значений входных данных из текста задачи	0.08343283300928306	38.7744140625 MB					

Вывод: в обратную сторону работает, время +- одинаковое

Задача №4. Линейный поиск

Текст задачи.

Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V

Если число встречается несколько раз, то выведете, сколько раз встречается число и все индексы і через запятую

```
import time
import tracemallocq
start = time.perf counter()
tracemalloc.start()
file = open('input.txt')
a = list(map(int, file.readline().split()))
v = int(file.readline())
ind = []
for i in range(len(a)):
  if a[i] == v:
     ind.append(i)
with open('output.txt', 'w') as file:
  if len(ind) == 0:
     file.write(str(-1))
        file.write(str(len(ind)) + ' ' + ', '.join(map(str, ind))) if len(ind) > 1 else
file.write(str(ind[0]))
print(time.perf counter() - start)
print(tracemalloc.get traced memory()[0] / 1024, 'MB')
```

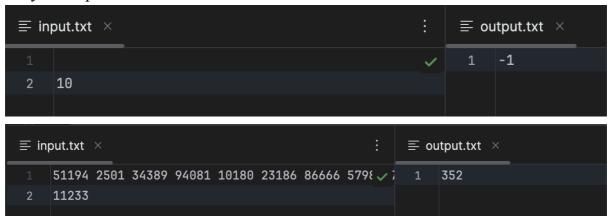
Сначала считываем данные из файла, потом прохожу по всему списку и записываю в список индексов все вхождения числа V. Записываю данные в выходной файл.

Результат работы кода на примерах из текста задачи:

```
      ≡ input.txt ×
      ⋮
      ≡ output.txt ×

      1
      9 9 1 4 2 6 7 4 6 8 9 2 12 36 12 34
      ✓
      1
      2

      2
      1
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
      □
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005055830115452409	1.6513671875 MB
Пример из задачи	0.000281250016996637	1.8896484375 MB
Пример из задачи	0.00042220798786729574	1.8896484345 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.005153624981176108	37.5732421875 MB

Задача №5. Сортировка выбором

Текст задачи.

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 π - 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.

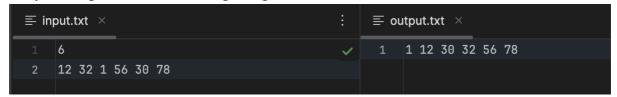
Формат входного и выходного файла и ограничения - как в задаче 1.

```
import time
import random
import tracemalloc
start = time.perf counter()
tracemalloc.start()
file = open('input.txt')
n = int(file.readline())
arr = list(map(int, file.readline().split()))
for i in range(len(arr) - 1):
  min el = i
  for j in range(i + 1, len(arr)):
     if arr[j] < arr[min el]:</pre>
        \min el = i
  arr[i], arr[min_el] = arr[min_el], arr[i]
with open('output.txt', 'w') as file:
  file.write(' '.join(map(str, arr)))
print(time.perf counter() - start)
print(tracemalloc.get_traced_memory()[0] / 1024, 'Mb')
```

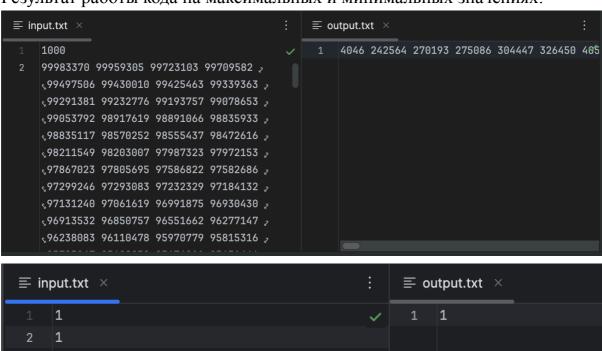
Я импортирую необходимые библиотеки, читаю данные из файла.

Прохожу циклом по массиву длиной len(arr) - 1, нахожу наименьший элемент и меняю его местами с текущим, таким образом получаю отсортированную последовательность и записываю ее в файл.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Проверка задачи на (openedu, астр и тд при наличии в задаче). (скрин)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0014942090201657265	1.4638671875 Mb
Пример из задачи	0.0014448750007431954	1.4951171875 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.235761125019053	37.4599609375 Mb

Вывод по задаче: в наихудшем случае, то есть когда последовательность отсортирована в обратном порядке, алгоритм срабатывает за 0.235761125019053 секунды, в то время как алгоритм из номера 1 за 0.23828616700484417, что на 0,002525041986 больше чем время данного.

Задача №6. Пузырьковая сортировка

Текст задачи.

Напишите код пузырьковой сортировка на Python и докажите корректность.

Листинг кода.

```
import time
import tracemalloc

tracemalloc.start()
start = time.perf_counter()

file = open('input.txt')
n = int(file.readline())
a = list(map(int, file.readline().split()))

for i in range(len(a) - 1):
    for j in range(len(a) - i - 1):
        if a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]

with open('output.txt', 'w') as file:
    file.write(' '.join(map(str, a)))

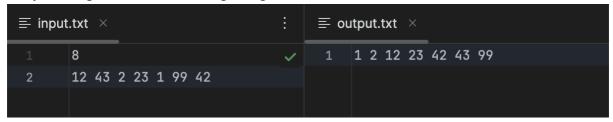
print(time.perf_counter() - start)
print(tracemalloc.get_traced_memory()[0] / 1024, 'MB')
```

Текстовое объяснение решения.

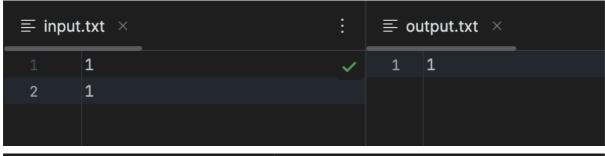
В основе метода сортировки лежат многократные перестановки, которые я реализовываю через два вложенных цикла. Если первый элемент больше

второго, то мы меняем их местами. Готовый массив записываю в выходной файл.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	≣ input	t.txt ×	÷	≡ οι	ıtput.txt	×					:
1	1	1000	~	1	122823	168782	178883	237352	327436	390004	453249
	2	99953448 99922764 99642682 9963940	3 99								

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0002589579962659627	1.6748046875 MB
Пример из задачи	0.0005174580146558583	1.7333984375 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.5633124170126393	37.6396484375 MB

Вывод по задаче: данная сортировка работает и дает на выходе корректный результат. Время в наихудшим случае, то есть когда массив отсортирован в обратную сторону, равно 0.5633124170126393(для максимальной верхней границы, данной в задаче). Алгоритм из номера 1 выполняют данную сортировку за 0.23828616700484417, что на 0,32502625 меньше, чем в данной задаче.

Задача №7. Знакомство с жителями Сортлэнда

Текст задачи.

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет п, где п может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, облада-ющим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним до-статком, если при сортировке жителей по сумме денежных сбережений он оказы-вается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до п. Информация о размере денежных накоплений жителей хранится в массиве М таким образом, что сумма денежных накоплений жителя, обладающего

идентификационным номером і, содержится в ячейке М і . 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 п (3 ≤ n < 9999, п нечетно). Вторая строка содержит описание массива М, состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива М раз-личны, а их значения имеют точность не более двух знаков после запятой и не превышают 10°.
- Формат выходного файла (output.txt). В выходной файл выведите три целых положительных числа, разделенных пробелами идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

```
import time
import tracemalloc

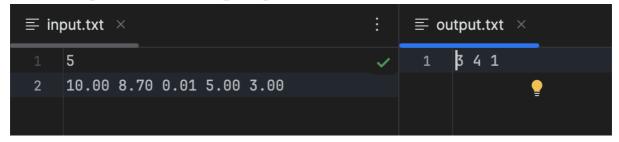
start = time.perf_counter()
tracemalloc.start()
```

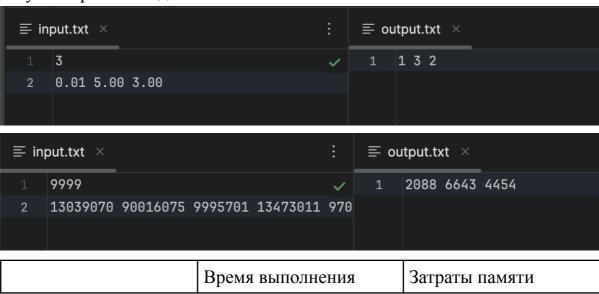
```
file = open('input.txt')
n = int(file.readline()) # 3 <= n <= 9999
a = file.readline().split()
m = sorted([(float(a[i]), i + 1) for i in range(len(a))])
res = [m[0][1], m[len(m) // 2 ][1], m[-1][1]]
with open('output.txt', 'w') as file:
    file.write(' '.join(map(str, res)))

print(time.perf_counter() - start)
print(tracemalloc.get_traced_memory()[0] / 1024, 'MB')</pre>
```

Для начала я сортирую массив жителей по их достатку, одновременно записываю достаток и идентификационный номер. Потом просто записываю в результирующий массив номер первого, среднего и последнего жителя. Записываю данные в выходной файл.

Результат работы кода на примерах из текста задачи:





Нижняя граница диапазона значений входных данных из текста задачи	0.0004199169925414026	1.9248046875 MB
Пример из задачи	0.0005991669895593077	2.2080078125 MB

Вывод по задаче: программа работает достаточно быстро и выводит корректный ответ.

Задача №8. Секретарь Своп

Текст задачи.

Дан массив, состоящий из п 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.

```
import time
import tracemalloc

start = time.perf_counter()
tracemalloc.start()

file = open('input.txt')
n = int(file.readline())
a = [int(i) for i in file.readline().split()]

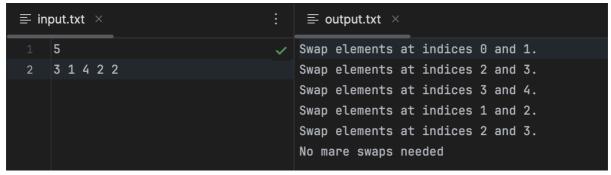
with open('output.txt', 'w') as file:
    for i in range(len(a) - 1):
        for j in range(len(a) - i - 1):
```

```
if a[j] > a[j + 1]:
    a[j], a[j + 1] = a[j + 1], a[j]
    file.write(f'Swap elements at indices {j} and {j + 1}.\n')
    file.write('No mare swaps needed')

print(time.perf_counter() - start)
print(tracemalloc.get_traced_memory()[0] / 1024, 'MB')
```

Получаю данные, сортирую их и сразу записываю в результирующий файл.

Результат работы кода на примерах из текста задачи:



≡ in	put.txt ×			:	≣ 0	utput.txt ×						
1	5000			~	Swap	elements	at	indices	591	and	592.	
2	19050507 911	.35477 46	321112	201235	Swap	elements	at	indices	592	and	593.	
					Swap	elements	at	indices	593	and	594.	
					Swap	elements	at	indices	594	and	595.	
					Swap	elements	at	indices	597	and	598.	
					Swap	elements	at	indices	598	and	599.	
					Swap	elements	at	indices	600	and	601.	
					Swap	elements	at	indices	601	and	602.	
					Swap	elements	at	indices	602	and	603.	
					Swap	elements	at	indices	603	and	604.	
					Swap	elements	at	indices	604	and	605.	
					Swap	elements	at	indices	605	and	606.	
					Swap	elements	at	indices	606	and	607.	
					Swap	elements	at	indices	607	and	608.	
					Swap	elements	at	indices	806	and	609.	
					Swap	elements	at	indices	609	and	610.	
					Swap	elements	at	indices	610	and	611.	
					Swap	elements	at	indices	611	and	612.	
					Swap	elements	at	indices	612	and	613.	
					Swap	elements	at	indices	613	and	614.	
					Swap	elements	at	indices	614	and	615.	
					Swap	elements	at	indices	616	and	617.	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006767910090275109	1.6357421875 MB
Пример из задачи	0.0020877910137642175	1.6669921875 MB
Верхняя граница диапазона значений входных данных из текста задачи		179.2255859375 MB

Вывод по задаче: алгоритм работает

Задача №9. Сложение двоичных чисел

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.

- Формат входного файла (input.txt). В одной строке содержится два пбитовых двоичных числа, записанные через пробел ($1 \le n < 103$)
- Формат выходного файла (output.txt). Одна строка двоичное число, которое является суммой двух чисел из входного файла.
- Оцените асимптотическое время выполнение вашего алгоритма.

Листинг кода.

```
import time
import tracemalloc

start = time.perf_counter()
tracemalloc.start()

file = open('input.txt')
a, b = file.readline().split()

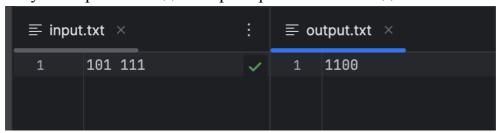
with open('output.txt', 'w') as file:
    file.write(bin(int(a, 2) + int(b, 2))[2:])

print(time.perf_counter() - start)
print(tracemalloc.get_traced_memory()[0] / 1024)
```

Текстовое объяснение решения.

Перевожу оба числа в 10 систему счисления с помощью int, выполняю операцию сложения и с помощью bin выполняю обратный перевод и делаю срез первых двух элементов.

Результат работы кода на примерах из текста задачи:



_ ≣ inpu	ıt.txt ×	:	≡ oι	ıtput.txt ×	
1	1 0	~	1	1	



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000963500002399087	1.5966796875 MB
Пример из задачи	0.00045037499512545764	1.6826171875 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.0003672919992823154	3.7177734375 MB

Вывод по задаче: работает достаточно быстро для верхней границы в том числе.

Задача №10. Палиндром

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.

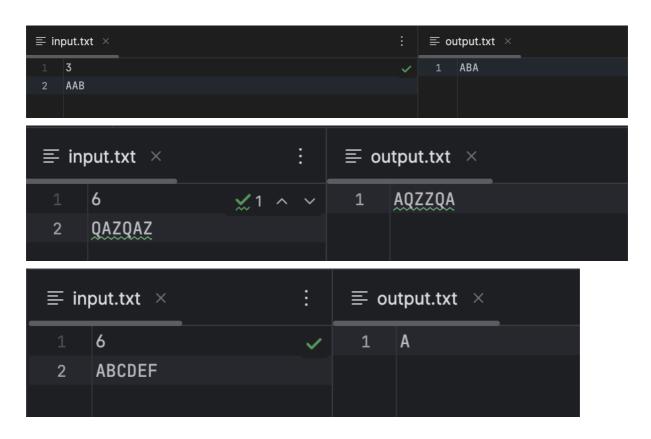
Требуется из данных букв по указанным правилам составить палиндром наиболь-шей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

```
import random
import time
import tracemalloc
start = time.perf counter()
tracemalloc.start()
file = open('input.txt')
n = int(file.readline())
list let = list(file.readline())
dict = \{\}
res = "
for ch in list let:
  if ch not in dict.keys():
     dict[ch] = 1
  else:
     dict[ch] += 1
     if dict[ch] == 2:
       res += ch
        dict[ch] = 0
res = ".join(sorted(res))
list mid = []
for ch, count in dict.items():
  if count == 1:
     list mid.append(ch)
with open('output.txt', 'w') as file:
  if list mid:
     file.write(res + str(sorted(list mid)[0]) + res[::-1])
  else:
     file.write(res + res[::-1])
print(time.perf_counter() - start)
print(tracemalloc.get traced memory()[0] / 1024, 'MB')
```

Для начала получаю строку из файла и сразу "разбиваю" ее в массив, таким образом получаю список со всеми буквами. Затем прохожу по этому

списку и в случает, если буква еще ни разу не встречалась, я записываю ее в словарь в качестве ключа, а в качестве значения указываю 1. Если буква уже встречалась ранее, то я уведичиваю ее счетчик на 1 и сразу проверяю, если значения равно 2, то у меня есть пара букв(которые можно поставить с разных сторон), записываю эту букву в результат и обнуляю счетчик. Таким образом у меня получается строка res со всеми парными буквами. Так как мне необходимо первое слово, то сортирую и возвращаю обратно в str формат. Затем я прохожу по ключам и значениям в словаре и записываю буквы, которые встречаются ровно один раз, чтобы добавить ее в середину(при наличии). В выходной файл записываю строку + среднюю букву + перевернутую строку, таким образом получаю палиндром.

Результат работы кода на примерах из текста задачи:



Результат на минимальном значении

≡ input.txt × :		≡ output.txt ×			
1	1	~	1	S	
2	S				

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.002662999992026016	2.2919921875 MB
Пример из задачи	0.0017458749935030937	2.3076171875 MB
Пример из задачи	0.0013538329803850502	2.3349609375 MB
Пример из задачи	0.0012444160238374025	2.5576171875 MB

Вывод по задаче: интересная задача, в целом работает

Вывод

В лабораторной работе я вспомнила некоторые алгоритмы сортировки, убедилась в их корректности, так же открыла для себя несколько новых алгоритмов сортировки. Попрактиковала свои навыки.