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

Выполнила:

Жмачинская Д.С.

К3141

Проверила:

Ромакина О.М.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета

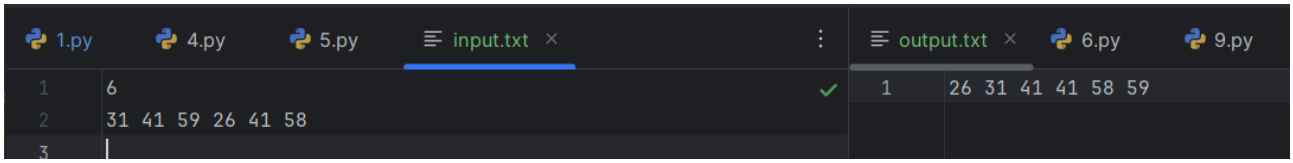
Задача №1. Сортировка вставкой

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

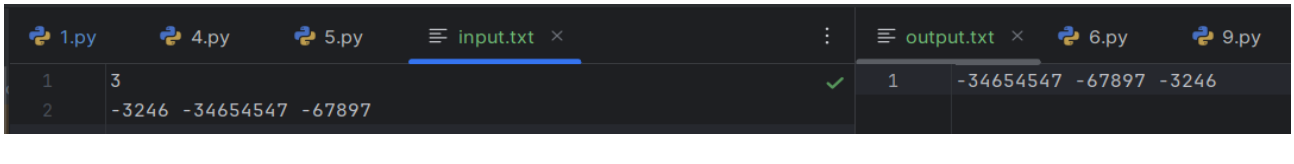
```
def insertion_sort(n, lst):
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j] < lst[j - 1]:
                lst[j], lst[j - 1] = lst[j - 1], lst[j]
            else:
                break
    return lst
import time, tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f = open("input.txt")
n = int(f.readline())
mas = [int(el) for el in f.readline().split()]
f.close()
str_lst = list(map(str, insertion_sort(n, mas)))
res = " ".join(str_lst)
w = open("output.txt", 'w')
w.write(res)
w.close()
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
tracemalloc.stop()
```

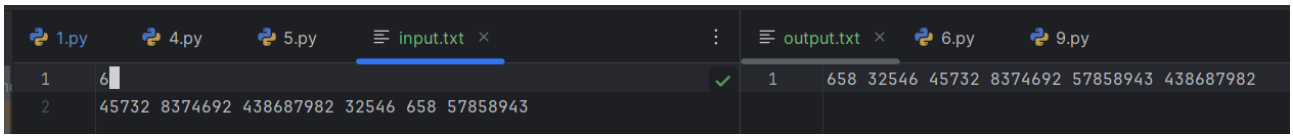
Текстовое объяснение решения.
Получаем значение n и mas из файла, передаем эти данные в функцию insertion_sort. В функции производим сортировку вставками: проходимся по списку от i элемента влево, в случае если элемент слева больше, чем i, то меняем местами, если нет, значит элемент встал на свое место, выходим из вложенного цикла. После выполнения функции выводим ответ.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.002744500001426786	0.01726055145263672
Пример из задачи	0.002919199992902577	0.017251014709472656
Верхняя граница диапазона значений входных данных из текста задачи	0.002271599951200187	0.01729869842529297

Задача №4. Линейный поиск

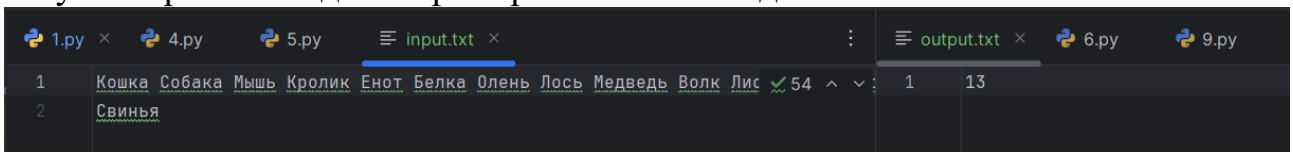
Текст задачи.
Рассмотрим задачу поиска. Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V. Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую. Дополнительно: попробуйте найти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

```
def linear_search(mas, v):
    ind = []
    for i in range(len(mas)):
        if mas[i] == v:
            ind.append(i)
    if len(ind) > 0:
        return ind
    return [-1]
import time, tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f = open("input.txt")
mas = [el for el in f.readline().split()]
v = f.readline()
f.close()
str_lst = list(map(str, linear_search(mas, v)))
res = ", ".join(str_lst)
w = open("output.txt", 'w')
w.write(res)
w.close()
```

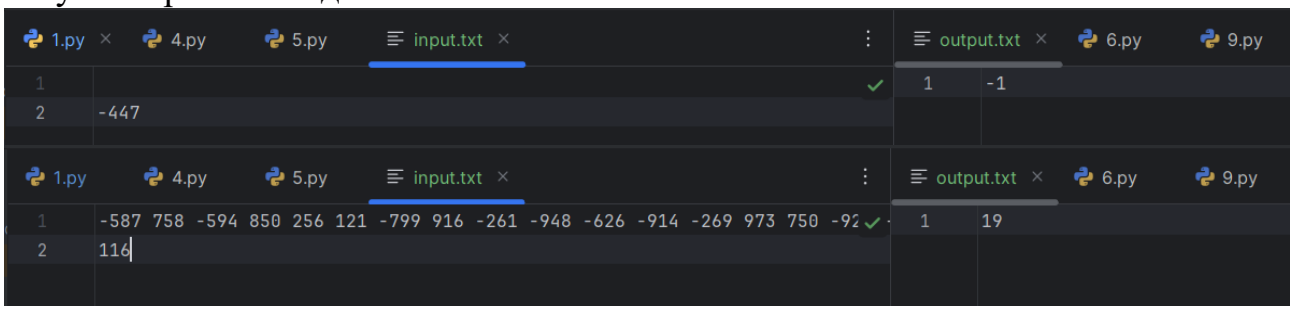
```
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
tracemalloc.stop()
```

Текстовое объяснение решения.
Получаем данные из файла. Функция linear_search: перебираем все элементы списка, в случае если элемент совпадает с значением поиска – записываем индекс в массив. В конце возвращаем список индексов (если длина списка равна 0, то возвращаем список [-1]). Выводим все в файл.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0006035000005795155	0.01712989807128906
Пример из задачи	0.002090900000439433	0.023641586303710938
Верхняя граница диапазона значений входных данных из текста задачи	0.0018307000000277185	0.08409786224365234

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.

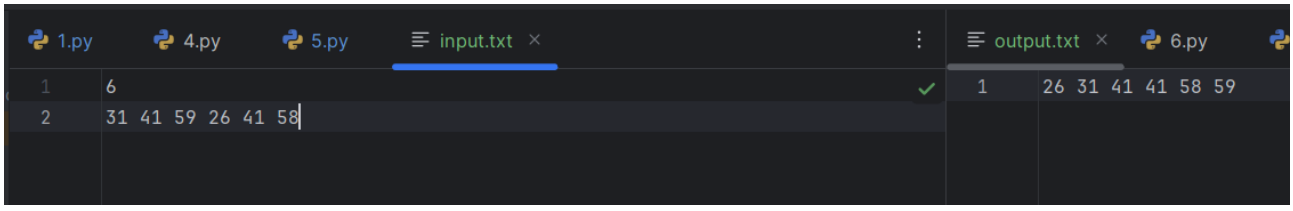
Текст задачи.

```
import time, tracemalloc
def selection_sort(n, lst):
    for i in range(n-1):
        mn = lst[i]
        temp = i
        for j in range(i+1, n):
            if mn > lst[j]:
                mn = j
                temp = j
    if temp != i:
        lst[i], lst[temp] = lst[temp], lst[i]
    return lst
tracemalloc.start()
t_start = time.perf_counter()
f = open("input.txt")
n = int(f.readline())
mas = [int(el) for el in f.readline().split()]
f.close()
str_lst = list(map(str, selection_sort(n, mas)))
res = " ".join(str_lst)
w = open("output.txt", 'w')
w.write(res)
w.close()
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
tracemalloc.stop()
```

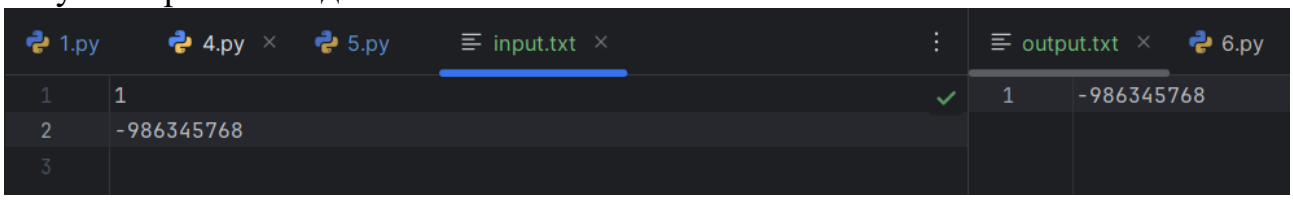
Текстовое объяснение решения.

Функция selection_sort: ищет в массиве минимальный элемент (начиная от i+1) потом этот мин. элемент меняет с текущем (левым).

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0030870999908074737	0.016962051391601562
Пример из задачи	0.0009128000237978995	0.017273902893066406
Верхняя граница диапазона значений входных данных из текста задачи	0.0005562000442296267	0.017395973205566406

Задача №6. Пузырьковая сортировка

Текст задачи.

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректоности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq \ldots \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A.

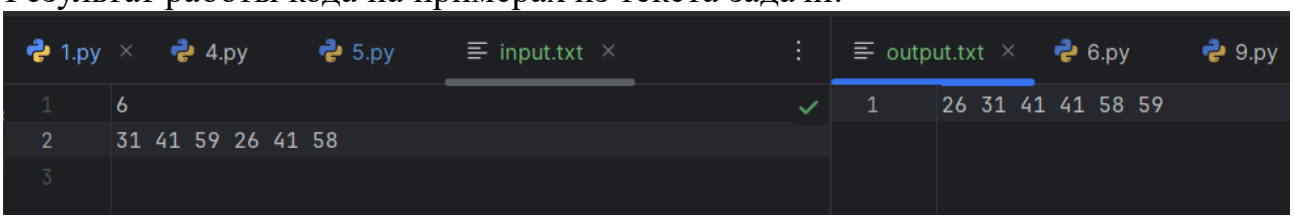
Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

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

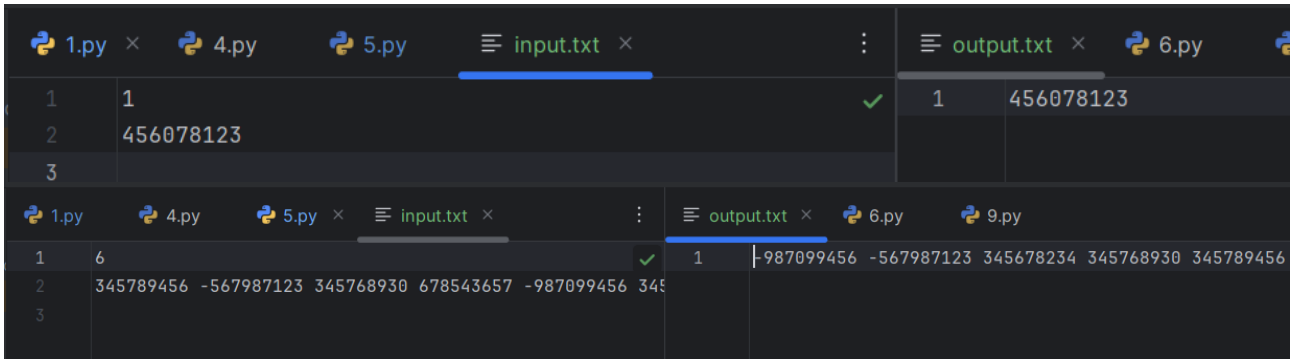
```python
import time, tracemalloc
def bubble_sort(n, lst):
    for i in range(n-1):
        exc = 0
        for j in range(n - i - 1):
            if lst[j] > lst[j+1]:
                exc += 1
                lst[j], lst[j+1] = lst[j+1], lst[j]
        if exc == 0:
            break
    return lst
def proof(lst):
    for i in range(len(lst)-1):
        if not(int(lst[i]) <= int(lst[i+1])):
            return False
        return True
tracemalloc.start()
t_start = time.perf_counter()
f = open("input.txt")
n = int(f.readline())
mas = [int(el) for el in f.readline().split()]
f.close()
str_lst = list(map(str, bubble_sort(n, mas)))
res = " ".join(str_lst)
w = open("output.txt", 'w')
w.write(res)
w.close()
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
print(proof(str_lst))
tracemalloc.stop()
```

Текстовое объяснение решения.
Функция bubble_sort: сравниваем два соседних элемента, в случае если элемент слева больше, чем элемента справа –> меняем их местами. Переменная exc служит для оптимизации кода. Если после вложенного цикла переменная exc равна нулю, следовательно, перестановок за данную итерацию не было -> массив отсортирован Функция proof линейно перебирает отсортированный массив и проверяет условие, что элемент слева меньше или равен элементу справа. Если все в порядке (пары прошли проверку) возвращаем True, иначе False.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0005039999959990382	0.017235755920410156
Пример из задачи	0.001854300033301115	0.017273902893066406
Верхняя граница диапазона значений входных данных из текста задачи	0.0005289999535307288	0.016962051391601562

Задача №9. Сложение двоичных чисел

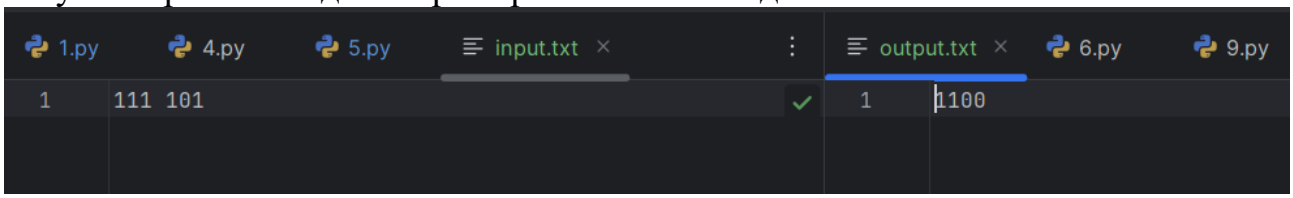
Текст задачи.
Рассмотрим задачу сложения двух n-битовых двоичных целых чисел, хранящихся в n-элементных массивах A и B. Сумму этих двух чисел необходимо занести в двоичной форме в (n + 1)-элементный массив C. Напишите скрипт для сложения этих двух чисел.

```
def addition_bin(a, b):
    n = len(a)
    temp = 0
    res = ''
    for i in range(n-1, -1, -1):
        if int(a[i]) + int(b[i]) == 2:
            if temp == 0:
                res = '0' + res
            else:
                res = '1' + res
            temp = 1
        elif int(a[i]) + int(b[i]) == 1:
```

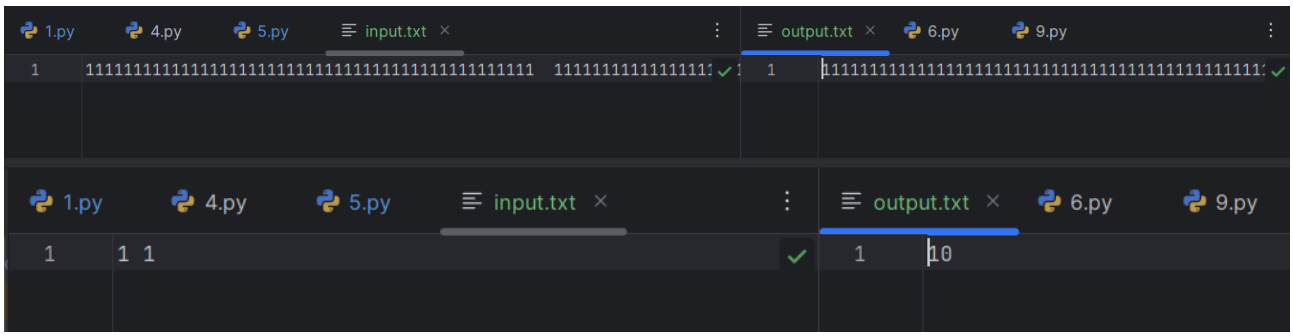
```
            if temp == 1:
                res = '0' + res
                temp = 1
            else:
                res = '1' + res
        else:
            if temp == 1:
                res = '1' + res
                temp = 0
            else:
                res = '0' + res
    if temp == 1:
        res = "1" + res
    return res
import time, tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f = open("input.txt")
a, b = map(str, f.readline().split())
res = addition_bin(a, b)
w = open("output.txt", 'w')
w.write(res)
w.close()
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
tracemalloc.stop()
```

Текстовое объяснение решения. Функция addition_bin: перебираем строки с конца в начало. Переменная temp – переменная “в уме”. Рассматриваем все случаи и записываем результат в зависимости от текущего значения переменной temp.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0007873000577092171	0.017928123474121094
Пример из задачи	0.0023950999984663213	0.018052101135253906
Верхняя граница диапазона значений входных данных из текста задачи	0.002390399982687086	0.01816558837890625

Вывод: Научилась пользоваться такими алгоритмами как: сортировка вставкой, линейный поиск. Сравнила алгоритмы сортировок: вставкой, выбором, пузырьковой.