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 15

Выполнил:
Скворцов Д.А.
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

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 ($1 \le n \le 10^3$) — 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.

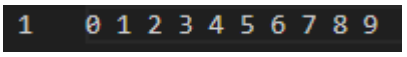
Листинг кода.

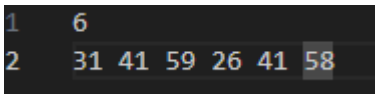
```
def insertion_sort(n, lst):
    if n <= 1: return

    for i in range(1, n):
        key_elem = lst[i]
        j = i - 1

        while j >= 0 and key_elem < lst[j]:
            lst[j+1] = lst[j]
            j -= 1
        lst[j+1] = key_elem
    return

def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
        n = int(inp.readline())
        lst = [int(x) for x in inp.readline().split()]
        insertion_sort(n, lst)
        print(*lst, file=out, end='')
```

Алгоритм сортировки вставками состоит из двух циклов: один отсчитывает ключевой элемент (*key_elem*), с которым сравнивается элементы отсортированной части; второй - меняет элементы местами, пока

не находит место для вставки ключевого элемента в отсортированную часть.

Далее функция *main* идентична для всех задач с разницей только в формате ввода и вывода.





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0003339000977575779 секунд	419 байт
Пример из задачи	0.0002631000243127346 секунд	475 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.6293573000002652 секунд	385 байт

Вывод по задаче:

Алгоритм реализован корректно: скорость не оптимальна (квадратична и быстро растёт), затраты по памяти постоянны.

Задача №4. Линейный поиск

Рассмотрим задачу поиска.

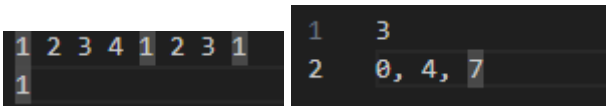
- **Формат входного файла.** Последовательность из n чисел A = a1, a2, . . . , an в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \le n \le 10^3$, $-10^3 \le a_i$, $V \le 10^3$.
- **Формат выходного файла.** Одно число - индекс i, такой, что V = A[i], или значение −1, если V в отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V .

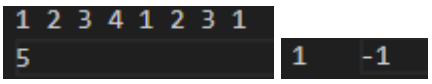
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

Листинг кода.

```
def linear_search(lst, v):
    return [i for i, a in enumerate(lst) if a == v]


def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
        lst = [int(x) for x in inp.readline().split()]
        v = int(inp.readline())
        res = linear_search(lst, v)
        if res:
            print(len(res), file=out)
            print(*res, file=out, sep=', ', end='')
        else:
            print(-1, file=out, end='')
```

Функция *linear_search()* возвращает список индексов всех элементов, равных заданному, иначе возвращает пустой список. Сам ответ обрабатывается и записывается в *output.txt* в зависимости от наполненности списка.





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00029550003819167614 секунд	424 байта
Верхняя граница диапазона значений входных данных из	0.0008562000002712011 секунд	395 байт

текста задачи		

Вывод по задаче:
Алгоритм работает корректно: время выполнения возрастает линейно в зависимости от длины входного массива, затрат памяти не происходит.

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.

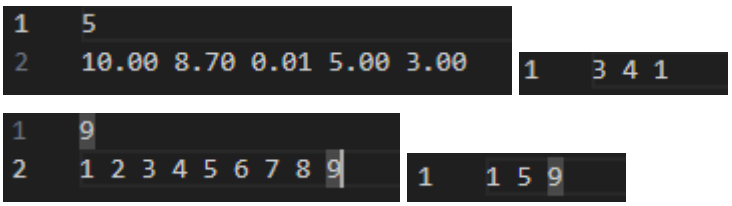
- **Формат входного файла (input.txt).** Первая строка входного файла содержит число жителей n ($3 \le n \le 9999$, n нечетно). Вторая строка содержит описание массива M, состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения 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.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода.

```
def insertion_sort(n, inp_lst):
    lst = [(i, x) for i, x in enumerate(inp_lst)]
    if n <= 1: return lst
```

```
    for i in range(1, n):
        key_elem = lst[i]
        j = i - 1
        while j >= 0 and key_elem[1] < lst[j][1]:
            lst[j+1] = lst[j]
            j -= 1
        lst[j+1] = key_elem
    return lst

def find_info(n, lst):
    s_lst = insertion_sort(n, lst)
    maxi, mini, mid = s_lst[-1][0], s_lst[0][0], s_lst[n//2][0]
    return mini+1, mid+1, maxi+1

def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
        n = int(inp.readline())
        inp_lst = [float(x) for x in inp.readline().split()]
        print(*find_info(n, inp_lst), file=out, end='')
```

Сортируем массив функцией *insertion_sort()* с добавлением в пару к элементам их индексов. Далее находим в отсортированном массиве при помощи длины индексы максимального, медианного и минимального элементов.



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00022280006669461727 секунд	424 байта
Верхняя граница диапазона значений входных данных из текста задачи	0.7349825999699533 секунд	84808 байт

Вывод по задаче:
Алгоритм работает корректно: искомые номера элементов находятся, но время выполнения зависит от скорости используемой сортировки, в данном случае - квадратично.

Дополнительные задачи

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.

Листинг кода.

```
def reverse_insertion_sort(n, lst):
    if n <= 1: return

    for i in range(n-2, -1, -1):
        key_elem = lst[i]
        j = i + 1

        while j <= n-1 and key_elem < lst[j]:
            lst[j-1] = lst[j] # swap
            j += 1
        lst[j-1] = key_elem
    return

def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
        n = int(inp.readline())
        lst = [int(x) for x in inp.readline().split()]
        reverse_insertion_sort(n, lst)
        print(*lst, file=out, end='')
```

Переделываем функцию *insertion_sort()* в работающую в обратную сторону: сортированная часть сохраняется в конце массива, а ключевой элемент берётся с последнего индекса и до нулевого.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.0003111001569777727 секунд	419 байт

входных данных из текста задачи		
Верхняя граница диапазона значений входных данных из текста задачи	1.6360555000137538 секунд	385 байт

Вывод по задаче:
Так же, как и в случае обычной сортировки вставками, время работы программы возрастает квадратично, затраты по памяти константны.

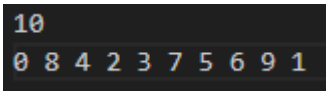
Задача №6. Пузырьковая сортировка

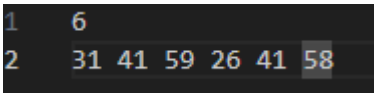
Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректоности процедуры вам необходимо доказать, что она завершается и что A′ [1] ≤ A′ [2] ≤ ... ≤ A′ [n], где A′ - выход процедуры Bubble_Sort, а n - длина массива A. Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой. Формат входного и выходного файла и ограничения - как в задаче 1.

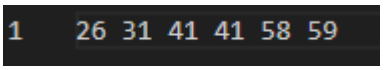
Листинг кода.

```
def bubble_sort(n, lst):
    if n <= 1: return

    for i in range(1, n):
        for j in range(n-1, i-1, -1):
            if lst[j] < lst[j-1]:
                lst[j], lst[j-1] = lst[j-1], lst[j]
    return

def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
        n = int(inp.readline())
        lst = [int(x) for x in inp.readline().split()]
        bubble_sort(n, lst)
        print(*lst, file=out, end='')
```

Используя псевдокод из условия напишем сортировку в функции *bubble_sort()*. Операцию *swap* выполним при помощи вырожденных кортежей.







	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0003299999516457319 секунд	419 байт
Верхняя граница диапазона значений входных данных из текста задачи	1.946696399943903 секунд	383 байта

Вывод по задаче:
При помощи проверяющей системы проверено и доказано, что сортировка завершает свою работу и выводит корректный результат. Сортировка пузырьком менее эффективна, чем вставками из-за постоянного использования операции *swap*.

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 ($1 \leq n \leq 10^3$)
- **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.

```
def bin_sum(a, b):
    ret = []
    carry = 0
    for i in range(len(a)-1, -1, -1):
        if a[i] == 1 and b[i] == 1:
```

```
            ret.append(1 if carry == 1 else 0)
            carry = 1
        elif a[i] == 0 and b[i] == 0:
            ret.append(1 if carry == 1 else 0)
            carry = 0
        else:
            ret.append(0 if carry == 1 else 1)
    if carry == 1:
        ret.append(1)
    return ret[::-1]


def main():
    with open('input.txt', 'r') as inp, open('output.txt', 'w') as out:
                    a,  b  =  [[int(x)  for  x  in  st]  for  st  in
inp.readline().rstrip().split()]
        res = bin_sum(a, b)
        print(*res, file=out, sep='', end='')
```

Программа проходится циклом по каждому разряду чисел, добавляя в результат и перенос 0 либо 1 по правилам сумматора. Если при завершении цикла совершается перенос разряда, то в начало нашего ответа необходимо добавить единицу.

```
1   1000001 1111111
```

```
1   11000000
```

```
1   10 10
```

```
1   100
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00035879993811249733 секунд	424 байта
Верхняя граница диапазона значений входных данных из текста задачи	0.0010881000198423862 секунд	16808 байт

Вывод по задаче:

Задача выполняется за линейное время, так как требует единоразового прохода по всей строке длиной n.

Вывод

Сортировки, работающие за квадратичное время (вставками, выбором, пузырьковая) можно использовать в задачах, не требующих обработки больших данных. Их плюсом является простота реализации, но скорость промышленной программы, пользующейся подобными сортировками будет неудовлетворительной.