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.
Продвинутый уровень.
Вариант 22

Выполнил:
Трусова Светлана Викторовна
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.

```
import time
import tracemalloc

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

with open('input1') as f:
   n = int(f.readline())
   massive = [int(a) for a in f.readline().split()]


def insertionsort(data):
   for i in range(1, n):
       key = data[i]
       j = i - 1
```

```
        while j >= 0 and key < data[j]:
            data[j+1] = data[j]
            j -= 1
        data[j+1] = key
    return data



with open('output1', 'w') as f:
    for element in insertionsort(massive):
        f.write(str(element))
        f.write(' ')


print(round(tracemalloc.get_traced_memory()[1]/(2**20), 5))
print(time.perf_counter() - t_start)
```

Описание работы кода:

1. Открываем файл ‘input1’, который содержит вводные данные.
2. Считываем первую строку файла содержащую число n.
3. Создаем переменную massive, которая будет списком и, пробегаясь по каждому элементу второй строки файла, записываем значение элементов в числовом виде.
4. Создаем функцию insertionsort, на вход которой поступает массив, который необходимо отсортировать методом вставок.
    a. Создаем цикл for, который начинается с 1(индекс второго элемента списка), отвечающий за индекс числа из неотсортированной части.
    b. Записываем в переменную key число, которое мы перемещаем по списку в “на его место”, а j будет индексом элемента, перед стоящего перед элементом key.
    c. Цикл while нам нужен для перемещения переменной key по списку, до того момента, пока она не встанет туда, где предыдущий элемент меньше key, а следующий больше.

d. После того как мы поставили на место число, мы можем переходить к следующему элементу из неотсортированной части.

5. После полной сортировки списка нам необходимо записать его в файл. Для этого открываем файл ‘output1’ в режиме ‘write’ и проходимся по элементам получившегося списка, записывая каждое с строчном представлении в выходной файл через пробел.

Тест	Время выполнение, сек	Затрата памяти, Mbytes
6 31 41 59 26 41 58	0.0022288998588919 64	0.03608
10 23 68 49 304 203 22 5905 2390 3920 1000	0.0035386001691222 19	0.03608

Вывод по заданию №1:

В ходе выполнения задания №1 я познакомилась с реализацией сортировки вставками на Python, повторила способы работы с файлами.

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.

```
import time
import tracemalloc


t_start = time.perf_counter()
```

```
tracemalloc.start()
def insertion_sort(n, data):
    indexes = [1]
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if data[i] < data[j]:
                data[i], data[j] = data[j], data[i]
                i, j = j, i
        indexes.append(i+1)
    return data, indexes



with open('input2', 'r') as file:
    n = int(file.readline())
            massive    =    [int(a)    for    a    in
file.readline().split()]
file.close()


result_data,   result_indexes   =   insertion_sort(n,
massive)


if 1 <= n <= 10**3:
    with open('output2', 'w') as file:
                        file.write('    '.join(map(str,
result_indexes)))
        file.write('\n')
        file.write(' '.join(map(str, result_data)))
    file.close()
else:
```

```
        print('Введенные данные не соответствуют
условию')


print(round(tracemalloc.get_traced_memory()[1]/(2**2
0), 5))
print(time.perf_counter() - t_start)
```

Описание работы кода:

1. Создаем функцию insertionsort, на вход которой поступает массив, который необходимо отсортировать методом вставок.
    a. На вход она принимает два параметра: n - число элементов массива и numbers - массив элементов. Далее производим сортировку вставкой данного массива: попарно сравниваем элементы и в случае, если правый меньше левого, меняем их местами. При этом создаем список indexes, в который будем записывать новые индексы нужных нам элементов. Изначально в этом списке только один элемент - единица. После выполнения этого алгоритма функция возвращает отсортированный список, а также список индексов.
2. Открываем файл ‘input2’, который содержит вводные данные.
3. Считываем первую строку файла содержащую число n.
4. Создаем переменную massive, которая будет списком и, пробегаясь по каждому элементу второй строки файла, записываем значение элементов в числовом виде.
6. После полной сортировки списка нам необходимо записать его в файл. Для этого открываем файл ‘output2’ в режиме ‘write’ и проходимся по элементам получившегося списка, записывая каждое с строчном представлении в выходной файл через пробел/

Тест	Время выполнения, сек	Затраченная память, Mbytes
10 1 8 4 2 3 7 5 6 9 0	0.0023834002204239 37	0.03609
10 23 68 49 304 203 22 5905 2390 3920 1000	0.0021362998522818 09	0.03609

Вывод по задаче №2:
В ходе решения данной задачи мы научились осуществлять сортировку вставкой массива целых чисел, выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях и разработали алгоритм по сохранению данных о новых индексах элементов при их обработке.

Задача №3. Сортировка вставкой по убыванию.

3 задача. 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.

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

```
import time
import tracemalloc


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


with open('input3') as f:
   n = int(f.readline())
   massive = [int(a) for a in f.readline().split()]



def insertionsort(data):
   for i in range(1, n+1):
       key = data[i]
       j = i - 1
```

```
        while j >= 0 and key > data[j]:
            data[j+1], data[j] = data[j], data[j+1]
            j -= 1
        data[j+1] = key
    return data



with open('output3', 'w') as f:
    for element in insertionsort(massive):
        f.write(str(element))
        f.write(' ')


print(round(tracemalloc.get_traced_memory()[1]/(2**20), 5))
print(time.perf_counter() - t_start)
```

Описание работы кода:

1. Открываем файл “input3”
2. Считываем первую строку файла содержащую число n.
3. Создаем переменную massive, которая будет списком и, пробегаясь по каждому элементу второй строки файла, записываем значение элементов в числовом виде.
4. Создаем функцию insertionsort, на вход которой поступает массив, который необходимо отсортировать методом вставок.
    a. Создаем цикл for, который начинается с 1(индекс второго элемента списка), отвечающий за индекс числа из неотсортированной части.
    b. Записываем в переменную key число, которое мы перемещаем по списку в “на его место”, а j будет индексом элемента, перед стоящего перед элементом key.
    c. Цикл while нам нужен для перемещения переменной key по списку, до того момента, пока она не встанет туда, где

предыдущий элемент больше key, а следующий меньше. (поменяли знак “<” на знак “>”)

d. После того как мы поставили на место число, мы можем переходить к следующему элементу из неотсортированной части.

4. После полной сортировки списка нам необходимо записать его в файл. Для этого открываем файл ‘output3’ в режиме ‘write’ и проходимся по элементам получившегося списка, записывая каждое с строчном представлении в выходной файл через пробел.

Тест	Время выполнение, сек	Затраченная память, Mbytes
6 31 41 59 26 41 58	0.0022288998588919 64	0.03593
10 23 68 49 304 203 22 5905 2390 3920 1000	0.0035386001691222 19	0.03593

Вывод по задаче №3:

В ходе выполнения задания №1 я познакомилась с реализацией сортировки вставками на Python, повторила способы работы с файлами.

Задача №5. Сортировка выбором.

5 задача. 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.

```
import time
import tracemalloc

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

with open('input5') as f:
   n = int(f.readline())
   massive = [int(a) for a in f.readline().split()]



def selectionsort(data):
   for i in range(n - 1):
       minim = i
       for j in range(i+1, n):
           if data[minim] > data[j]:
               minim = j
       data[minim], data[i] = data[i], data[minim]
   return data
```

```
with open('output5', 'w') as f:
    for element in selectionsort(massive):
        f.write(str(element))
        f.write(' ')


print(round(tracemalloc.get_traced_memory()[1]/(2**20), 5))
print(time.perf_counter() - t_start)
```

Описание работы кода:

1. Считываем первую строку файла содержащую число n.
2. Создаем переменную massive, которая будет списком и, пробегаясь по каждому элементу второй строки файла, записываем значение элементов в числовом виде.
3. Создаем функцию selectionsort, на вход которой поступает массив, который необходимо отсортировать методом выбора.
    e. Создаем цикл for, с отвечающий за индекс числа, на место которого надо поставить следующий наименьший элемент.
    f. Цикл for с переменной j отвечает за то, чтобы найти наименьший элемент списка из правой части.
    g. Проверяем нашли ли мы меньшее число =, чем число, которое было найдено до этого, и если оно меньше, то меняем индекс наименьшего числа на текущий j.
    h. После полного прохождения по всем элементам списка, меняем местами текущий (i-ый) элемент списка с найденым наименьшим элементом правее от него.
    i. После того как мы поставили на место число, мы можем переходить к следующему элементу из правой части.

4. После полной сортировки списка нам необходимо записать его в файл. Для этого открываем файл ‘output5’ в режиме ‘write’ и проходимся по элементам получившегося списка, записывая каждое с строчном представлении в выходной файл через пробел.

Тест	Время выполнение, сек	Затраченная память, Mbytes
6 31 41 59 26 41 58	0.0022288998588919 64	0.03593
10 23 68 49 304 203 22 5905 2390 3920 1000	0.0035386001691222 19	0.03593

Вывод по задаче №5:
В этой задаче мы узнали о новом методе сортировке selection sort, а затем успешно реализовали его. Такой подход обеспечивает стабильную работу для сортировки массива за фиксированное количество операций, что делает его подходящим для небольших наборов данных.

Задача №6. Пузырьковая сортировка.

6 задача. Пузырьковая сортировка

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 $A'[1] \leq A'[2] \leq ... \leq A'[n]$, 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.

```
import time
import tracemalloc


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

```
with open('input6') as f:
   n = int(f.readline())
   massive = [int(a) for a in f.readline().split()]


def bubblesort(n, data):
     for i in range(n - 1): #не делаем последнюю
проходку, потому что останется последний элемент
       for j in range(n):
           if data[j] > data[j+1]:
                       data[j], data[j+1] = data[j+1],
data[j]


   return data


with open('output6', 'w') as f:
   for element in bubblesort(n, massive):
       f.write(str(element))


       f.write(' ')


print(round(tracemalloc.get_traced_memory()[1]/(2**20
), 5))
print(time.perf_counter() - t_start)
```

Описание работы кода:

1. Открываем файл ‘input6’
2. Считываем первую строку файла содержащую число n.
3. Создаем переменную massive, которая будет списком и, пробегаясь по каждому элементу второй строки файла, записываем значение элементов в числовом виде.

4. Создаем функцию bubblesort, на вход которой поступает число n и массив, который необходимо отсортировать методом пузырьковой сортировки.
a. Создаем цикл for, с переменной i, отвечающей за количество прохождений по списку.
b. Цикл for с переменной j отвечает за то, чтобы сравнить следующий после элемента элемент списка.
c. Сравниваем j и j+1 элементы, если j > j+1, меняем их местами.
d. На последнем прохождении большие числа всплывают, как пузырики, поэтому в начале останется самый маленький элемент списка.

5. После полной сортировки списка нам необходимо записать его в файл. Для этого открываем файл ‘output6’ в режиме ‘write’ и проходимся по элементам получившегося списка, записывая каждое с строчном представлении в выходной файл через пробел.

Вывод по задаче №6:

В ходе выполнения задачи №6 был изучен метод пузырьковой сортировки, работа с файлами. Такой вид сортировки подх

Тест	Время выполнение, сек	Затраченная память, Mbytes
6 31 41 59 26 41 58	0.0022288998588919 64	0.03593
10 23 68 49 304 203 22 5905 2390 3920 1000	0.0035386001691222 19	0.03593

одит для небольшого массива данных.

Задача №7. Знакомство с жителями спортлэнда

7 задача. 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.

```
import time
import tracemalloc

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

with open('input7') as f:
    n = int(f.readline())
            massive1    =    [float(a)    for    a    in
f.readline().split()]
    massive = []
    cnt = 0
    for i in massive1:
        cnt += 1
```

```
        massive.append([i, cnt])
massive = sorted(massive)

with open('output7', 'w') as f:
    f.write(str(massive[0][1]))
    f.write(' ')
    f.write(str(massive[n // 2][1]))
    f.write(' ')
    f.write(str(massive[-1][1]))

print(round(tracemalloc.get_traced_memory()[1]/(2**2
0), 5))
print(time.perf_counter() - t_start)
```

Описание хода работы:

1. Открываем файл “input7”
2. создаем список с индексами жителей в списке и записываем в массив подмассивы, содержащие достаток жителя и его индекс.
3. Сортируем массив по доходу жителей
4. Записываем в открытый файл записываем индексы самого бедного, со среднем достатком и самого богатого жителя Сортлэнла.

Тест	Время выполнение, сек	Затраченная память, Mbytes
5 10.00 8.70 0.01 5.00 3.00	0.0052412999793887 14	0.03593

Вывод по задаче №7:

Иногда проще будет использовать готовую функцию sorted сортировки к спискам.

Вывод

Таким образом, в ходе выполнения лабораторной работы были рассмотрены и реализованы различные сортировки для массивов, а также повторены способы определения времени работы кода и количество памяти потраченное при выполнении кода.