

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИИ, МЕХАНИКИ И ОПТИКИ ФАКУЛЬТЕТ
ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных» Тема:
Сортировка вставками, выбором, пузырьковая.

Выполнил: Криличевский М. Е.

Номер группы: К3139

Проверил: Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание

Оглавление

<i>Содержание</i>	<i>2</i>
<i>Задание 1. Сортировка вставкой.....</i>	<i>3</i>
<i>Задание 2. Сортировка вставкой⁺.....</i>	<i>4</i>
<i>Задание 3. Сортировка вставкой по убыванию.....</i>	<i>7</i>
<i>Задание 4. Линейный поиск</i>	<i>9</i>
<i>Задание 5. Сортировка выбором.</i>	<i>11</i>
<i>Задание 6. Пузырьковая сортировка</i>	<i>14</i>
<i>Вывод</i>	<i>17</i>

Задание 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.

Решение:

```
1 import sys
2 import time
3 import resource
4 time_start = time.perf_counter()
5 def insertion_sort(a): 1 usage 2 Krimaxev *
6     for j in range(len(a)):
7         key = a[j]
8         u = j-1
9         while u >= 0 and a[u] > key:
10             a[u+1] = a[u]
11             u = u-1
12         a[u+1] = key
13     return a
14 f, f2 = open("input_n1", "r"), open("output_n1", "w")
15 l, l2 = int(f.readline()), f.readline()
16 m = [int(x) for x in l2.split()]
17 if l <= 0 or l > 10**3: print("Лимит длины превышен")
18 if l != len(m): print("Ошибка")
19 for i in range(len(m)):
20     if abs(m[i]) > 10**9: print("Ошибка")
21
22 t = insertion_sort(m)
23 s = ''
24 m1 = [str(x) for x in t]
25 for z in m1:
26     s += z
27     s += ' '
28 f2.write(s)
29
30 time_elapsed = (time.perf_counter() - time_start)
31 memMb = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
32 f.close()
33 f2.close()
34 print("ВРЕМЯ:", time_elapsed)
35 print("Память: %5.1f МБ" % (memMb))
36 print(f"Размер списка: {sys.getsizeof(m)} байт")
```

Объяснение решения: Программа открывает файл input_n1.txt, читает длину второй строки и кладет в массив m вторую строку, разбивая ее на отдельные числа в цифровом формате. Затем, с помощью алгоритма Insertion_sort, программа сортирует массив m по возрастанию. Далее массив m повторно перебирается с помощью массива m1, и каждый его элемент конвертируется обратно в строку s. После этого строка s записывается в выходной файл output_n1.txt, а с помощью импортированных библиотек time, sys и resource выводятся затраты времени и памяти, а также объем массива в байтах.

Входной файл:

1	6
2	31 41 59 26 41 58
	💡

Выходной файл:

task1.py		output_n1	×
1	26 31 41 41 58 59		
	💡		

Затраты времени и памяти:

Run	task1	×
↺	⏸	:
↑	/usr/local/bin/python3.12 /Users/max/	
↓	ВРЕМЯ: 0.00018704100511968136	
↺	Память: 9.6 МБ	
↻	Размер списка: 120 байт	
⏏	Process finished with exit code 0	
🗑		

Задание 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.

Пример.

input.txt output.txt

10 1222355691 1842375690 0123456789

В примере сортировка вставками работает следующим образом:

- Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]
- Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
- Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
- Двойка занимает второе место. [1 2 4 8]
- Тройка занимает третье место. [1 2 3 4 8]
- Семерка занимает пятое место. [1 2 3 4 7 8]
- Пятерка занимает пятое место. [1 2 3 4 5 7 8]
- Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]
- Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]
- Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

Решение:

```
1 import sys
2 import time
3 import resource
4 time_start = time.perf_counter()
5 f,f2 = open("input_n2","r"),open("output_n2","w")
6 l,l2= int(f.readline()),f.readline()
7 m = [int(x) for x in l2.split()]
8 m2 = []
9 if l!=len(m): print("Ошибка")
10 for j in range(len(m)):
11     key = m[j]
12     u = j - 1
13     while u >= 0 and m[u] > key:
14         m[u + 1] = m[u]
15         u = u - 1
16     m[u + 1] = key
17     m2.append(m.index(key))
18 s = ''
19 s1 = ''
20 m1 = [str(x) for x in m]
21 m3 = [str(x+1) for x in m2]
22 for z in m1:
23     s+=z
24     s+=' '
25 for p in m3:
26     s1+=p
27     s1+=' '
28 f2.writelines(s1+"\n")
29 f2.writelines(s)
30 f2.close()
31 f.close()
32 time_elapsed = (time.perf_counter() - time_start)
33 memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
34 print("BPEPM:",time_elapsed)
```

Объяснение решения: Программа открывает файл input_n2.txt, читает длину второй строки и кладет в массив m вторую строку, разбивая ее на отдельные числа в цифровом формате. Затем, с помощью технологии перебора, основанной на алгоритме Insertion_sort, программа сортирует массив m по возрастанию, а новые индексы элементов массива m записываются в массив m2. Далее массивы m и m2 повторно перебираются с помощью массива m1 и m3, и каждый элемент массивов конвертируется в строки s и s1. После этого строки s1 и s записываются в выходной файл

output_n2.txt, а с помощью импортированных библиотек time, sys и resource выводятся затраты времени и памяти, а также объем массива в байтах.

Входной файл:

task2.py		input_n2	output_n2
1		10	
2		1 8 4 2 3 7 5 6 9 0	
		💡	

Выходной файл:

task2.py		input_n2	output_n2
1		1 2 2 2 3 5 5 6 9 1	
2		0 1 2 3 4 5 6 7 8 9	
3		💡	
4			

Затраты времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/  
ВРЕМЯ: 0.0008504999859724194  
Память: 9.7 МБ  
Размер списка: 184 байт  
  
Process finished with exit code 0
```

Задание 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.

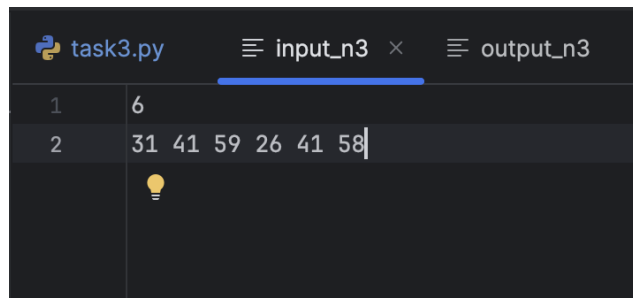
Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Решение:

```
1 import sys
2 import time
3 import resource
4 time_start = time.perf_counter()
5 def insertion_sort(a): 1 usage ± Krimaxev *
6     for j in range(len(a)):
7         key = a[j]
8         u = j-1
9         while u>=0 and a[u]<key:
10             a[u+1]=a[u]
11             u = u-1
12         a[u+1] = key
13     return a
14 f,f2 = open("input_n3","r"),open("output_n3","w")
15 l,l2= int(f.readline()),f.readline()
16 m = [int(x) for x in l2.split()]
17 if l==0 or l>10**3: print("Лимит длины превышен")
18 if l!=len(m): print("Ошибка")
19 for i in range(len(m)):
20     if abs(m[i])>10**9: print("Ошибка")
21 t = insertion_sort(m)
22 s = ''
23 m1 = [str(x) for x in m]
24 for z in m1:
25     s+=z
26     s+=' '
27 f2.writelines(s)
28 f.close()
29 f2.close()
30 time_elapsed = (time.perf_counter() - time_start)
31 memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
32 print("ВРЕМЯ:",time_elapsed)
33 print ("Память:%5.1f МБ" % (memMb))
34 print(f"Размер списка: {sys.getsizeof(m)} байт")
```

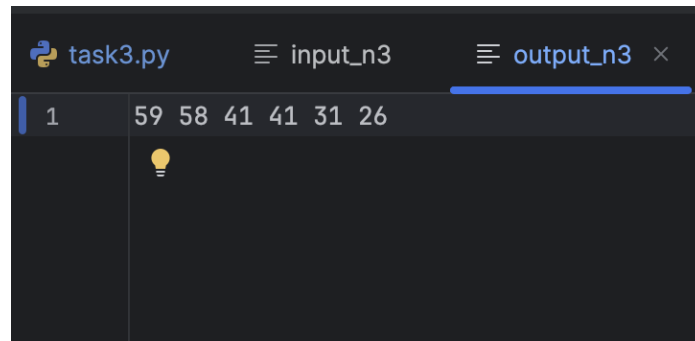
Объяснение решения: Программа открывает файл input_n3.txt, читает длину второй строки и кладет в массив m вторую строку, разбивая ее на отдельные числа в цифровом формате. Затем, с помощью обратного алгоритма Insertion_sort, программа сортирует массив m по не возрастанию. Далее массив m повторно перебирается с помощью массива m1, и каждый его элемент конвертируется обратно в строку s. После этого строка s записывается в выходной файл output_n3.txt, а с помощью импортированных библиотек time, sys и resource выводятся затраты времени и памяти, а также объем массива в байтах.

Входной файл:



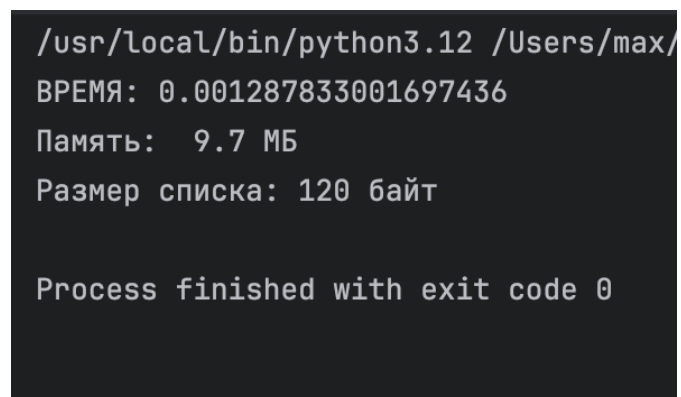
The screenshot shows a code editor with three tabs: 'task3.py', 'input_n3', and 'output_n3'. The 'input_n3' tab is active. It contains two lines of input data: line 1 has the number '6', and line 2 has the sequence '31 41 59 26 41 58'. A lightbulb icon is visible below the input data.

Выходной файл:



The screenshot shows the same code editor with the 'output_n3' tab active. It contains one line of output data: '59 58 41 41 31 26'. A lightbulb icon is visible below the output data.

Затраты времени и памяти:



The screenshot shows a terminal window with the following output:

```
/usr/local/bin/python3.12 /Users/max/  
ВРЕМЯ: 0.001287833001697436  
Память: 9.7 МБ  
Размер списка: 120 байт  
  
Process finished with exit code 0
```


Задание 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: $0 \leq n \leq 103, -103 \leq a[i], V \leq 103$

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.


Решение:

```
task4.py x input_n4 output_n4 task5.py task6.py
1 import time
2 start = time.time()
3 import resource
4 time_start = time.perf_counter()
5
6 f, f2 = open("input_n4", "r"), open("output_n4", "w")
7 l, l2 = f.readline(), int(f.readline())
8 m = [int(x) for x in l.split()]
9
10 if len(m) > 10**3: print("Лимит длины превышен")
11 for j in range(len(m)):
12     if m[j] < -10**3 or m[j] > 10**3: print("Ошибка")
13 if l2 < -10**3 or l2 > 10**3: print("Лимит значения превышен")
14 m1 = [x for x, y in enumerate(m) if y == l2]
15
16 str_count = str(len(m1))
17 s = ''
18 for k in range(len(m1)):
19     if len(m1) == 1: f2.write(str(m1[k]))
20     else:
21         s += str(m1[k])
22         s += ', '
23 if len(m1) == 0:
24     f2.write("-1")
25
26 else:
27     f2.write(str_count + '\n')
28     f2.write(s[:-2])
29 f.close()
30 f2.close()
31 time_elapsed = (time.perf_counter() - time_start)
32 memMb = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss / 1024.0 / 1024.0
33 print("ВРЕМЯ:", time_elapsed)
34 print("Память: %.5f МБ" % (memMb))
```


Объяснение решения: Программа открывает файл input_n4.txt, читает значение V и кладет в массив m вторую строку, разбивая ее на отдельные числа в цифровом формате. Затем, с помощью перебора элементов массива m в массиве $m1$ по соответствию элемента массива m значению V (в массиве $m1$ хранятся индексы элементов V). Далее массив $m1$ перебирается с проверкой соответствия следующим условиям: если длина массива $m1$ равна 1, то в выходной файл записывается индекс значения V , если нет элементов, соответствующих значению V , то в выходной файл записывается «-1», если

значение V повторяется несколько раз, то в выходной файл записывается длина массива $m1$ и все его элементы, переведенные в строку s с пробелом через запятую. После этого строка s записывается в выходной файл `output_n4.txt`, а с помощью импортированных библиотек `time` и `resource` выводятся затраты времени и памяти.

Входной файл:

1	1 7 2 7 3 7 4 7 5 7 6 7 7 7 8 7 9 7 10
2	7
	

Выходной файл:

1	10
2	1, 3, 5, 7, 9, 11, 12, 13, 15, 17
	

Затраты времени и памяти:

```
Run task4 x
/usr/local/bin/python3.12 /Users/max/
ВРЕМЯ: 0.0006544999778270721
Память: 9.5 МБ
Process finished with exit code 0
```

Задание 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.

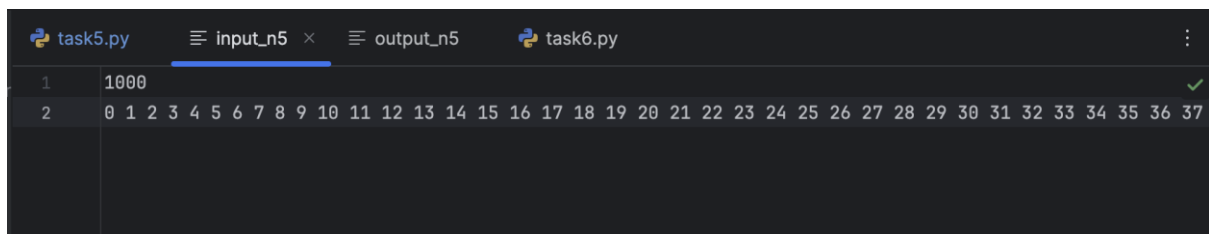
Формат входного и выходного файла и ограничения - как в задаче 1.

Решение:

```
task5.py x input_n5 output_n5 task6.py
1 import sys
2 import time
3 import resource
4 time_start = time.perf_counter()
5 def Selection_sort(a): 1 usage 1 Krimaxev
6     for i in range(0, len(a)-1):
7         mn = i
8         for j in range(i+1, len(a)):
9             if a[j] < a[mn]:
10                 mn = j
11             a[i], a[mn] = a[mn], a[i]
12     return a
13 f, f2 = open("input_n5", "r"), open("output_n5", "w")
14 l, l2 = int(f.readline()), f.readline()
15 m = [int(x) for x in l2.split()]
16
17 if l==0 or l>10**3: print("Лимит длины превышен")
18 if len(m)!=l: print("Ошибка")
19 for i in range(len(m)):
20     if abs(m[i])>10**9: print("Ошибка")
21
22 t = Selection_sort(m)
23 s = ''
24 m1 = [str(x) for x in m]
25 for z in m1:
26     s+=z
27     s+=' '
28 f2.write(s)
29 f.close()
30 f2.close()
31 time_elapsed = (time.perf_counter() - time_start)
32 memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
33 print("ВРЕМЯ:", time_elapsed)
34 print("Память:%5.1f МБ" % (memMb))
35 print(f"Размер списка: {sys.getsizeof(m)} байт")
36
```

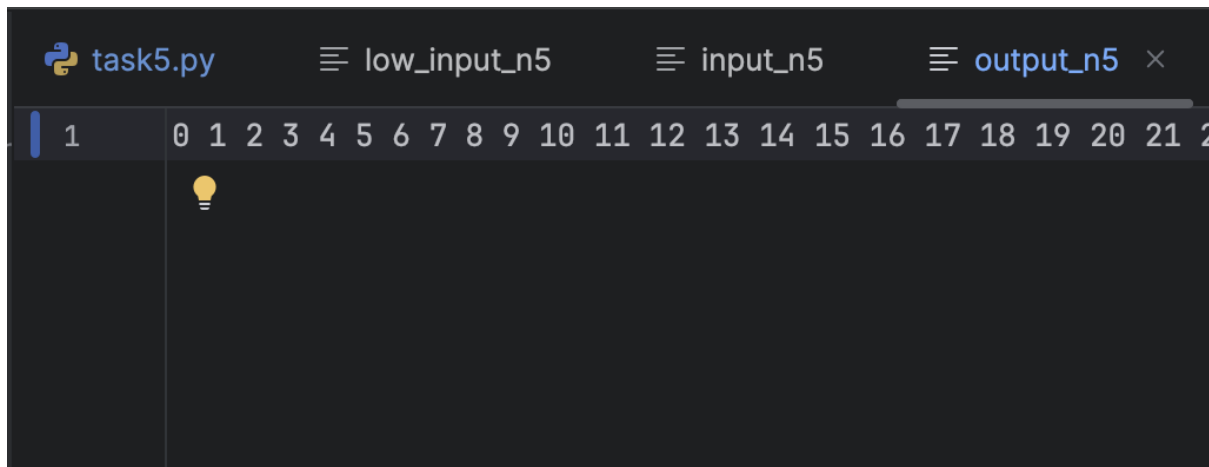
Объяснение решения: Программа открывает файл `input_n5.txt`, читает длину второй строки и кладет в массив `m` вторую строку, разбивая ее на отдельные числа в цифровом формате. Затем, с помощью алгоритма `Selection_sort`, программа сортирует массив `m` по возрастанию. Далее массив `m` повторно перебирается с помощью массива `m1`, и каждый его элемент конвертируется обратно в строку `s`. После этого строка `s` записывается в выходной файл `output_n5.txt`, а с помощью импортированных библиотек `time`, `sys` и `resource` выводятся затраты времени и памяти, а также объем массива в байтах.

Входной файл (наихудший случай):



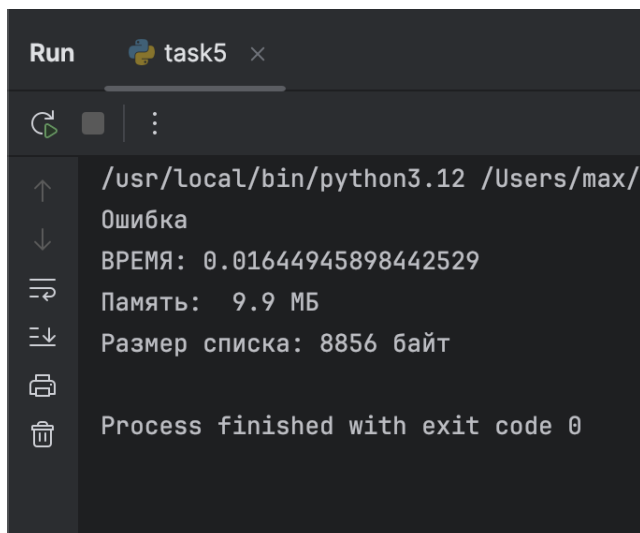
```
task5.py  input_n5  output_n5  task6.py
1 1000
2 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
```

Выходной файл (наихудший случай):



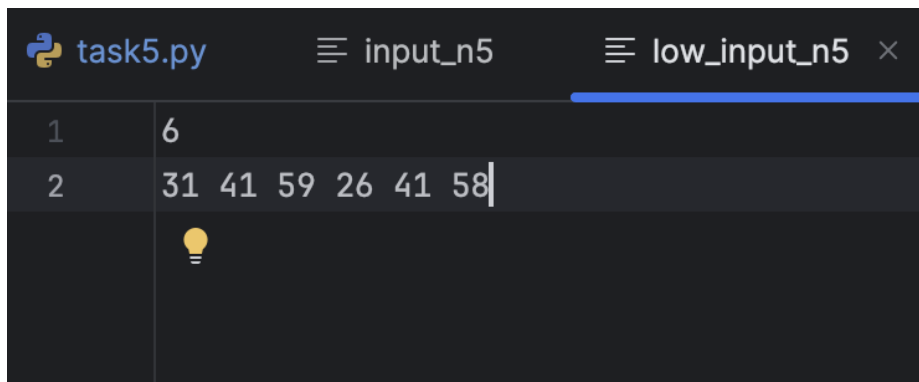
```
task5.py  low_input_n5  input_n5  output_n5
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 2
2
```

Затраты времени и памяти (наихудший случай):



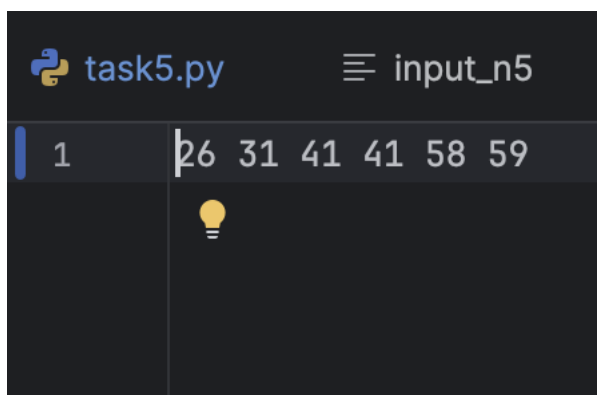
```
Run task5
/usr/local/bin/python3.12 /Users/max/...
Ошибка
ВРЕМЯ: 0.01644945898442529
Память: 9.9 МБ
Размер списка: 8856 байт
Process finished with exit code 0
```

Входной файл:



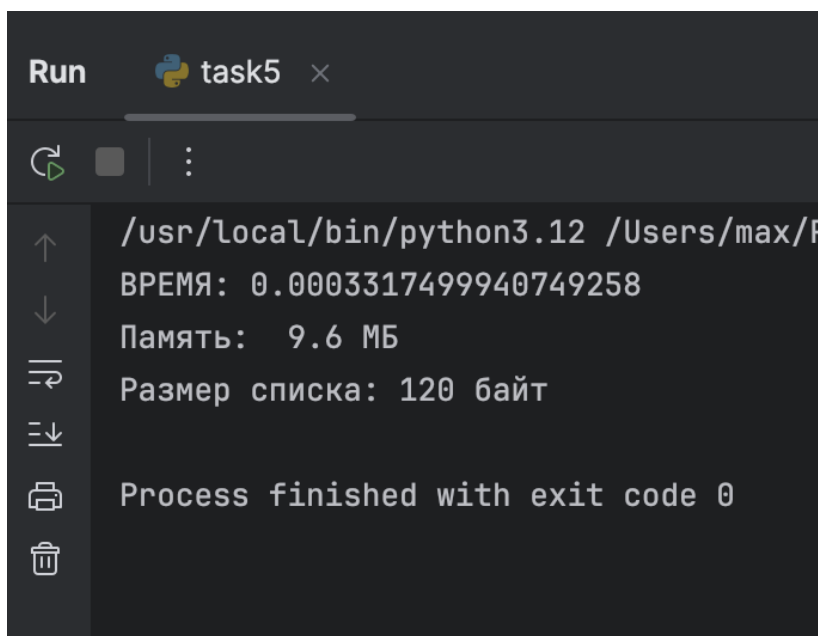
```
task5.py  input_n5  low_input_n5 x
1 6
2 31 41 59 26 41 58|
  
```

Выходной файл:



```
task5.py  input_n5
1 26 31 41 41 58 59
  
```

Затраты времени и памяти:



```
Run  task5 x
  
```

```
↑ /usr/local/bin/python3.12 /Users/max/P
↓ ВРЕМЯ: 0.0003317499940749258
⇌ Память: 9.6 МБ
⇌ Размер списка: 120 байт
⇌
⇌ Process finished with exit code 0
⇌
⇌
  
```

Задание 6. Пузырьковая сортировка

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

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, 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.

Решение:

```
task6.py x input_n6 output_n6
1 import sys
2 import time
3 import resource
4 time_start = time.perf_counter()
5
6 def Bubble_sort(a): 1 usage 1 KrimaxeV
7     for i in range(1, len(a)-1):
8         for j in range(len(a), i+1):
9             if a[j]<a[j-1]:
10                 a[j], a[j-1] = a[j-1], a[j]
11             return a
12
13
14 f, f2 = open("input_n6", "r"), open("output_n6", "w")
15 l, l2 = int(f.readline()), f.readline()
16 m = [int(x) for x in l2.split()]
17
18 if l==0 or l>10**3: print("Лимит длины превышен")
19 if len(m)!=l: print("Ошибка")
20 for i in range(len(m)):
21     if abs(m[i])>10**9: print("Ошибка")
22
23 t = Bubble_sort(m)
24 s = ''
25 m1 = [str(x) for x in m]
26 for z in m1:
27     s+=z
28     s+=' '
29 f2.write(s)
30 f.close()
31 f2.close()
32 time_elapsed = (time.perf_counter() - time_start)
33 memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
34 print("ВРЕМЯ:", time_elapsed)
35 print("Память:%5.1f МБ" % (memMb))
36 print(f"Размер списка: {sys.getsizeof(m)} байт")
```

Объяснение решения: Программа открывает файл input_n6.txt, читает длину второй строки и кладет в массив m вторую строку, разбивая ее на отдельные числа в цифровом

формате. Затем, с помощью алгоритма Bubble_sort, программа сортирует массив m. Далее массив m повторно перебирается с помощью массива m1, и каждый его элемент конвертируется обратно в строку s. После этого строка s записывается в выходной файл output_n6.txt, а с помощью импортированных библиотек time, sys и resource выводятся затраты времени и памяти, а также объем массива в байтах.

Входной файл (наихудший случай):

1	1000
2	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Выходной файл (наихудший случай):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

Затраты времени и памяти (наихудший случай):

```
Run task6 x
0.0010022500064224005
Память: 9.8 МБ
Размер списка: 8856 байт
Process finished with exit code 0
```

Входной файл:

task6.py		input_n6	×	output_n6
1	6			
2	31 41 59 26 41 58			
	💡			

Выходной файл:

task6.py		input_n6	output_n6	×
1	31 41 59 26 41 58			
	💡			

Затраты времени и памяти:

```
Run task6 ×
🔄 ⬛ ⋮
↑ /usr/local/bin/python3.12 /Users/max/Pychar
↓ ВРЕМЯ: 0.0007437499880325049
⇌ Память: 10.2 МБ
⇌ Размер списка: 120 байт
⇌
🖨 Process finished with exit code 0
🗑
```


Вывод

В данной лабораторной работе лучше познакомился и поработал с алгоритмами сортировки: Insertion_sort, Bubble_sort и Selection_sort. Также научился делать обратный алгоритм сортировки и получил опыт работы с индексами элементов в списке.