

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 21

Выполнила:
Савченко А.С.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №3. Сортировка вставкой по убыванию	5
Задача №4. Линейная сортировка	7
Задача №5. Сортировка выбором.	10
Задача №9. Сложение двоичных чисел	13
Вывод (по всей лабораторной)	16

Задачи по варианту

Задача №1. Сортировка вставкой

Текст задачи.

1 задача. 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.

Листинг кода. (именно листинг, а не скрины)

```
def InsertionSort(A):  
    for j in range(1, len(A)):  
        key = A[j]  
        i = j - 1  
        while i >= 0 and A[i] > key:  
            A[i + 1] = A[i]  
            i = i - 1  
        A[i + 1] = key  
    return A  
  
if __name__ == "__main__":  
    with open('input.txt', 'r') as file:
```

```

n = int(file.readline().strip())
array = list(map(int,
file.readline().strip().split()))

sorted_array = InsertionSort(array)

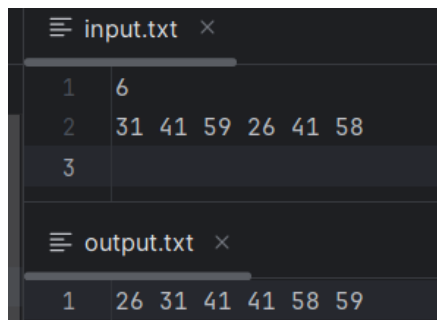
with open('output.txt', 'w') as file:
    file.write(' '.join(map(str, sorted_array)))

```

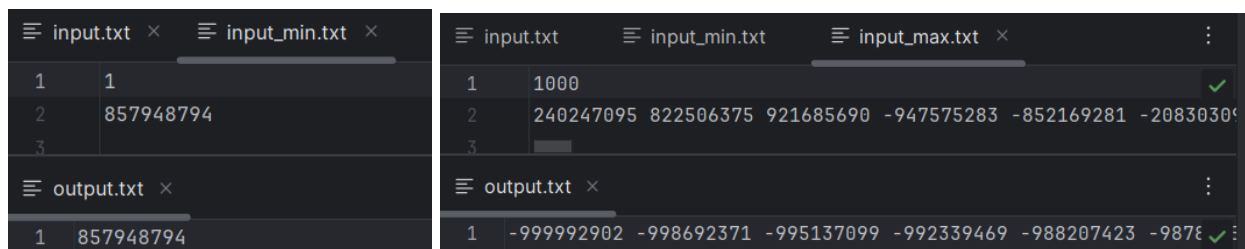
Текстовое объяснение решения:

Чтобы отсортировать массив целых чисел в неубывающем порядке, был реализован классический алгоритм сортировки вставкой (Insertion Sort). На каждой итерации цикла новый элемент вставляется в правильное место в отсортированной части массива.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов)



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.0006604 с	15.01953125 Мб

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.000531 с	14.921875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.0192105	14.87265625 Мб

Вывод по задаче:

Алгоритм выполняет свою функцию, сортирует массив целых чисел в неубывающем порядке, успешно проходит тесты и укладывается во временные ограничения.

Задача №3. Сортировка вставкой по убыванию

Текст задачи.

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?

Листинг кода. (именно листинг, а не скрины)

```
def Swap(A, i, j):
    A[i], A[j] = A[j], A[i]

def InsertionSortDescending(A):
    for j in range(1, len(A)):
        key = A[j]
        i = j - 1
        while i >= 0 and A[i] < key:
            Swap(A, i, i + 1)
```

```

        i = i - 1
        A[i + 1] = key
    return A

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
        arr = list(map(int,
file.readline().strip().split()))

    sort_array = InsertionSortDescending(arr)

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

```

Текстовое объяснение решения.

Основной алгоритм повторяет алгоритм задачи 1, но для сортировки в обратном порядке тут присутствует функция Swap, меняющая элементы местами. Так, при каждом проходе элемент встает на свое место в порядке убывания.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt		input_min.txt	
1	6		
2	31 41 59 26 41 58		
3			
output.txt			
1	59 58 41 41 31 26		

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0009091	14.96484375 Мб
Пример из задачи	0.0005882 с	14.84375 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.0318473	15.16015625 Мб

Вывод по задаче:

Добавив функцию Swap, мы изменили алгоритм сортировки вставкой (Insertion Sort) на убывающий порядок. Алгоритм выполняет свою функцию, сортирует массив целых чисел в невозрастающем порядке, успешно проходит тесты и укладывается во временные ограничения.

Задача №4.Линейная сортировка

Текст задачи.

4 задача. 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: $0 \leq n \leq 10^3$, $-10^3 \leq a_i, V \leq 10^3$
- **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. (именно листинг, а не скрины)

```
def linear_search(A, V):
    indices = [i for i, x in enumerate(A) if x == V]
    if indices:
        return len(indices), indices
    else:
        return -1

if __name__ == "__main__":
    with open('input.txt', 'r') as file:
        A = list(map(int,
file.readline().strip().split()))
        V = int(file.readline().strip())

    result = linear_search(A, V)

    with open('output.txt', 'w') as file:
        if result == -1:
            file.write(str(result))
```



```

else:
    file.write(f"{result[0]}\n")
    file.write(','.join(map(str, result[1])) +
'\n')

```

Текстовое объяснение решения.

Считываем массив и искомое число, создаем массив индексов, в цикле алгоритм последовательно проверяет каждый элемент массива пока не найдет совпадение или не дойдет до конца, возвращаем кол-во вхождений и индекс, если искомое число не было найдено возвращ -1.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

input.txt	
1	5 1 2 3 4 5
2	3
3	
output.txt	
1	1
2	3

input.txt		input_min.txt		input_max.txt	
1	Акула Аллигатор Анаконда Антилопа Барсук Белка Бизон 90 ^				
2	Свинья				
3					
output.txt					
1	1				
2	77				

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов)

input.txt		input_min.txt	
1	-1000		
2	-1000		
3			
output.txt			
1	1		
2	0		
3			

```

input.txt x  input_min.txt  input_max.txt x
1  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ✓
2  199
3
output.txt x
1  1 ✓
2  198

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.004689 с	14.9453125 Мб
Пример из задачи	0.000686 с	14.7578125 Мб
Пример из задачи	0.0005681 с	14.83203125 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.0012044 с	15.11328125 Мб

Вывод по задаче:

Алгоритм успешно ищет число в массиве. Кроме того возвращается кол-во вхождений искомого числа в массиве, в этом случае выводятся все позиции вхождений.

Добавила указание на кодировку utf-8 и Свинья найдена:) Несмотря на то, что это не самый эффективный способ.

Задача №5. Сортировка выбором.

Текст задачи.

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.

Листинг кода. (именно листинг, а не скрины)

```
def selection_sort(A):
    n = len(A)
    for i in range(n - 1):
        min_index = i
        for j in range(i + 1, n):
            if A[j] < A[min_index]:
                min_index = j
        A[i], A[min_index] = A[min_index], A[i]
    return A

if __name__ == "__main__":
    with open('input.txt', 'r', encoding='utf-8') as file:
        n = int(file.readline().strip())
        array = list(map(int, file.readline().strip().split()))

        sorted_array = selection_sort(array)

        with open('output.txt', 'w', encoding='utf-8') as file:
            file.write(' '.join(map(str, sorted_array)))
```

Текстовое объяснение решения.

Как работает алгоритм:

Найди наименьший элемент в списке.

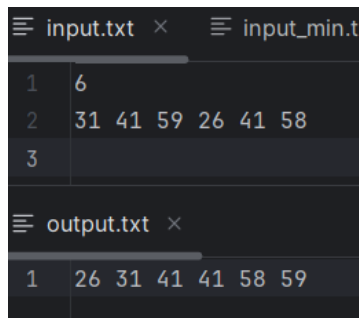
Поставь его в начало списка, а первый элемент перемести туда, где раньше стоял наименьший.

Теперь ищи следующий наименьший элемент, но не учитывая уже отсортированные элементы.

Поставь этот элемент на второе место, а второй элемент перемести на освободившееся место.

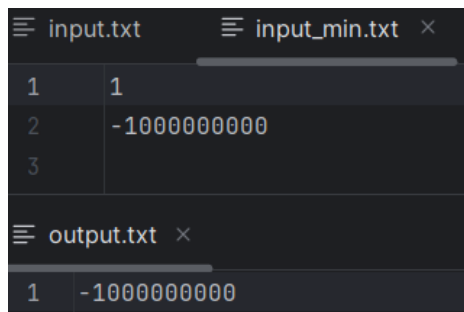
Повторяй эти шаги, пока не дойдешь до конца списка.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)

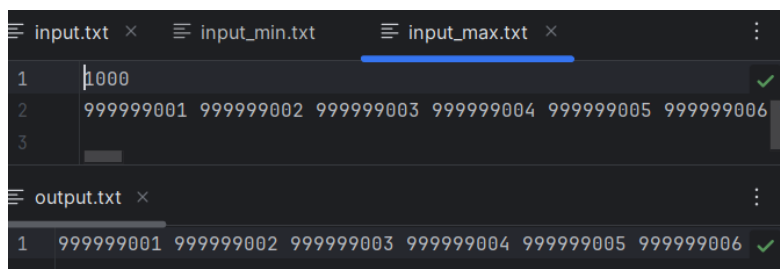


```
input.txt x input_min.t
1 6
2 31 41 59 26 41 58
3
output.txt x
1 26 31 41 41 58 59
```

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов)



```
input.txt input_min.txt x
1 1
2 -1000000000
3
output.txt x
1 -1000000000
```



```
input.txt x input_min.txt x input_max.txt x
1 1000 ✓
2 999999001 999999002 999999003 999999004 999999005 999999006 ✓
3
output.txt x
1 999999001 999999002 999999003 999999004 999999005 999999006 ✓
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0006508 с	14.98828125 Мб
Пример из задачи	0.0006382 с	14.84765625 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.022542 с	15.24609375 Мб

Вывод по задаче:

Алгоритм успешно сортирует используя сортировку выбором и проходит все тесты, укладываясь во временные ограничения.

При сравнении двух алгоритмов(Сортировка вставкой и Сортировка выбором) не трудно понять, что обе имеют одинаковую временную сложность $O(n^2)$ в наихудшем случае, но при небольшом количестве элементов сортировка вставкой может быть быстрее, так как она выполняет меньше операций.

Задача №9. Сложение двоичных чисел

Текст задачи.

9 задача. 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 выполнения вашего алгоритма.

Листинг кода. (именно листинг, а не скрины)

```
def binary_addition(A, B):
    n = len(A)
    C = [0] * (n + 1)
    curr = 0

    for i in range(n - 1, -1, -1):
        total = A[i] + B[i] + curr
        C[i + 1] = total % 2
        curr = total // 2

    C[0] = curr
    return C


def read_input_file():
    with open('input.txt', 'r') as file:
        line = file.readline().strip()
    A_str, B_str = line.split()
    A = list(map(int, A_str))
    B = list(map(int, B_str))
    return A, B

def write_output_file(C):
    with open('output.txt', 'w') as file:
        file.write(''.join(map(str, C)) + '\n')

if __name__ == "__main__":
    A, B = read_input_file()
    array = binary_addition(A, B)
    write_output_file(array)
```

Текстовое объяснение решения.

Для удобства читаем числа как списки, создаем элементный массив C длиной $(n+1)$. В цикле идем с конца массивов, складывая соответствующие биты и учитываем перенос(`car`). Записываем результат сложения каждого бита в соответствующую позицию массива результата и обновляем перенос. Результат работы кода на примерах из текста задачи: (скрины input output файлов)

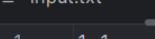


```

input.txt x
1 1011 1101
2

```

Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов)



The screenshot shows a Jupyter Notebook with two cells. The first cell, titled 'input.txt', contains the code `min(1, 1)`. The second cell, titled 'output.txt', displays the result of the execution, which is `1`.

[illegible]

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005575 с	14.9140625 Мб

Пример из задачи	0.0008198 с	14.73046875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.0010352 с	15.02734375 Мб

Вывод по задаче:

Получившийся алгоритм успешно складывает двоичные числа используя идею массива и хранения переноса. Тесты из примера, а также на максимальных и минимальных значениях пройдены успешно за время и память гораздо меньшие чем отведенные.

Вывод (по всей лабораторной)

Благодаря выполнению этой базовой лабораторной работы я вспомнила о простых видах сортировок и их асимптотику. Все алгоритмы корректно работают для входных данных и укладываются в ограничения времени и памяти.