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

Отчет по лабораторной работе №2 по курсу «Алгоритмы и структуры данных»

Тема: Сортировка слиянием. Метод декомпозиции

Выполнила:
Сусликова В.Д.
Группа:
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Сортировка слиянием

Листинг кода:

```
def sort_merge(nums):
    if len(nums) > 1:
        middle = len(nums) // 2
        left = nums[:middle]
        right = nums[middle:]
        sort_merge(left)
        sort_merge(right)
        p = r = q = 0
        while (p < len(left)) and (r < len(right)):
            if left[p] < right[r]:
                nums[q] = left[p]
                p += 1
            else:
                nums[q] = right[r]
                r += 1
            q += 1
        while p < len(left):
            nums[q] = left[p]
            p += 1
            q += 1
        while r < len(right):
            nums[q] = right[r]
            r += 1
            q += 1
    return nums
output_f = open("C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task1/txtf/output.txt", 'w')
input_f = open("C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task1/txtf/input.txt")
num_len = int(input_f.readline())
file = input_f.readline()
if 1 <= num_len <= 2 * 10 ** 4:
    nums = list(map(int, list(file.split(' '))))
    if all([abs(x) <= 10 ** 9 for x in nums]):
        output_f.write(' '.join(map(str, sort_merge(nums))))
    else:
        print('Введите подходящие числа')
else:
```

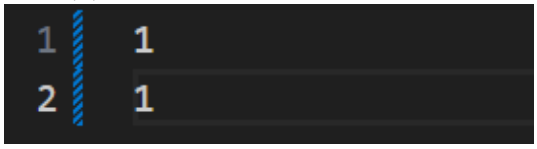
```
    print('Неверное количество введенных чисел')
output_f.close()
input_f.close()
```

Открываем входные и выходные файлы input и output (в данном случае с полным путем, чтобы избежать проблем с структурой папок). Далее вводим функцию, реализующую алгоритм сортировки слиянием: Делим массив напополам по среднему элементу. Так продолжаем пока длина каждой части массива не станет равна 1. После этого осуществляем слияние частей массива в отсортированном порядке. После кодовой реализации алгоритма мы считываем данные из input файла, делаем проверки на правильность количества введенных чисел и вхождение этих чисел в указанный в задании диапазон значений, после чего считанные данные передаем в написанную ранее функцию, получаем возвращенное ей значение и записываем его в output файл, после чего закрываем все открытые ранее файлы.

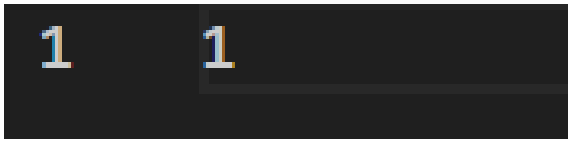
Пример ввода и вывода:

Минимальные значения:

Ввод:



Вывод:



Пример:

Ввод:

Вывод:

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница диапазона значений входных данных	0.000585556	0.0002135999966
Верхняя граница диапазона значений входных данных из текста задачи	0.1178998947	0.26745719998

Пример	0.000778198	0.0005880999

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, познакомились с алгоритмом сортировки слиянием и принципом декомпозиции и смогли оценить его время выполнения и затраты памяти.

Задача №3. Число инверсий

Листинг кода:

```
def search(mass, copy_mass, l, mid, r):
    i = k = l
    j = mid + 1
    count_inf = 0

    while i <= mid and j <= r:
        if mass[i] <= mass[j]:
            copy_mass[k] = mass[i]
            i += 1
        else:
            copy_mass[k] = mass[j]
            j += 1
            count_inf += (mid - i + 1)
        k += 1

    while i <= mid:
        copy_mass[k] = mass[i]
        k += 1
        i += 1

    for i in range(l, r + 1):
        mass[i] = copy_mass[i]
    return count_inf


def search_inversions(mass, copy_mass, l, r):
    if r <= l:
        return 0

    mid = (l + r) // 2
    count_inf = 0
```

```
    count_inf += search_inversions(mass, copy_mass, l, mid)
    count_inf += search_inversions(mass, copy_mass, mid + 1, r)
    count_inf += search(mass, copy_mass, l, mid, r)

    return count_inf


with open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task3/txtf/output.txt', 'w') as f:
    file = open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task3/txtf/input.txt')
    n = int(file.readline())
    nums = list(map(int, file.readline().split()))
    nums_copy = nums.copy()
    f.write(str(search_inversions(nums, nums_copy, 0, n - 1)))
```

Открываем вводные и выводные файлы input и output и реализуем функцию, выполняющую алгоритм продвинутой сортировки вставками: попутно с проведением самой сортировки ведем дополнительный массив с данными о совершенных в процессе исполнения алгоритма перестановках, занося туда новый индексы переставляемых элементов массива. После кодовой реализации алгоритма мы считываем данные из input файла, делаем проверки на правильность количества введенных чисел и вхождение этих чисел в указанный в задании диапазон значений, после чего считанные данные передаем в написанную ранее функцию, получаем возвращенное ей значение и записываем его в output файл, после чего закрываем все открытые ранее файлы.

Пример ввода и вывода:

Пример:

Ввод:

```
1   10
2   1 8 4 2 3 7 5 6 9 0
```

Вывод:

```
1   1 2 2 2 3 5 5 6 9 1
2   0 1 2 3 4 5 6 7 8 9
```

Минимальные значения:

Ввод:

```
1   1
2   1
```

Вывод:

```
1   1
2   1
```

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница диапазона значений входных данных из текста задачи	0.0005893707	0.0004886999
Верхняя граница диапазона значений входных данных из текста задачи	0.142508506	0.202545600
Пример	0.00109291076	0.0005232999

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, познакомились с алгоритмом продвинутой сортировки вставками и смогли оценить его время выполнения и затраты памяти, а так же благодаря дополнению исходного алгоритма смогли детально изучить работу сортировок вставками, отслеживая каждый совершаемый ход.

Задача №4. Бинарный поиск

Листинг кода:

```
file = open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task4/txtf/input.txt')
n, mass = int(file.readline()), list(map(int,
file.readline().split()))
k, mass_find = int(file.readline()), list(map(int,
file.readline().split()))


def bin_searching(mass, what_find):
    l = 0
    r = len(mass) - 1
    while l <= r:
        mid = (l + r) // 2
        if what_find == mass[mid]:
            return mid
        elif what_find < mass[mid]:
            r = mid - 1
```

```
        else:
            l = mid + 1
    return -1


def start_search(mass, mass_find):
    return [bin_searching(mass, mass_find[i]) for i in
range(len(mass_find))]


output_f = open("C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task1/txtf/output.txt", 'w')
output_f.write(' '.join(map(str, start_search(mass, mass_find))))
```

Открываем вводные и выводные файлы input и output и реализуем функцию, выполняющую алгоритм обратной сортировки вставками (сортировки по убыванию), который идентичен исходному алгоритму сортировки вставками с единственным изменением: key (сравниваемая переменная) должна быть в данном случае больше, а не меньше предыдущей, чтобы выполнилось условие их перестановки. После кодовой реализации алгоритма мы считываем данные из input файла, делаем проверки на правильность количества введенных чисел и вхождение этих чисел в указанный в задании диапазон значений, после чего считанные данные передаем в написанную ранее функцию, получаем возвращенное ей значение и записываем его в output файл, после чего закрываем все открытые ранее файлы.

Пример ввода и вывода:

Пример:

Ввод:

```
1   6
2   31 41 59 26 41 58
```

Вывод:

```
1   59 58 41 41 31 26
```

Минимальные значения:

Ввод:

```
1   1
2   1
```

Вывод:

```
1   1
```

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница диапазона значений входных данных из текста задачи	0.0005855560	0.000137700
Верхняя граница диапазона значений входных данных из текста задачи	0.120044708	0.319034200
Пример	0.000778198	0.00028169999

Ответ на дополнительный вопрос: Да, алгоритм сортировки вставками можно переписать с помощью рекурсии, однако он не имеет заметных позитивных отличий от оригинального алгоритма, затрачивает примерно столько же времени и значительно больше памяти засчет рекурсии соответственно

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, познакомились с алгоритмом обратной сортировки вставками (смогли соответствующим заданию образом переделать исходный алгоритм сортировки вставками) и смогли оценить его время выполнения и затраты памяти, так же ответили на дополнительный вопрос задачи – смогли представить себе реализацию алгоритма сортировки вставками с помощью рекурсии и примерно оценить ее отличия в затратах времени и памяти от оригинала.

Задача №5 .Представитель большинства

Листинг кода:

```
def majority_finder(a):
    dic = {}
    for i in a:
        dic[i] = dic.get(i, 0) + 1

    for k, v in dic.items():
        if v > len(a) // 2:
            return 1
```

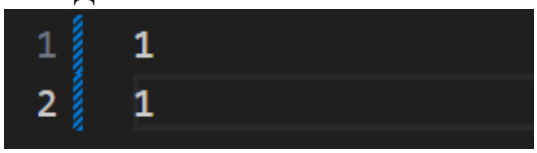
```
    return 0


with open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task5/txtf/output.txt', 'w') as f:
    file = open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task5/txtf/input.txt')
    n = int(file.readline())
    list_input = list(map(int, file.readline().split()))
    f.write(str(majority_finder(list_input)))
```

Открываем входные и выходные файлы input и output (в данном случае с полным путем, чтобы избежать проблем с структурой папок). Далее вводим функцию, реализующую алгоритм Bubble - sort: заводим циклы основной и вложенный, которые будут менять местами соседние переменные с перебираемыми в циклах индексами при условии, что число с меньшим индексом больше числа с большим. После кодовой реализации алгоритма мы считываем данные из input файла, делаем проверки на правильность количества введенных чисел и вхождение этих чисел в указанный в задании диапазон значений, после чего считанные данные передаем в написанную ранее функцию, получаем возвращенное ей значение и записываем его в output файл, после чего закрываем все открытые ранее файлы.

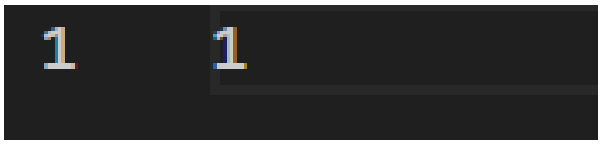
Пример ввода и вывода:

Минимальные значения:

Ввод:

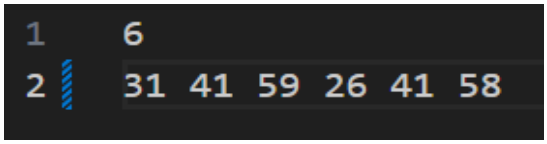


Вывод:

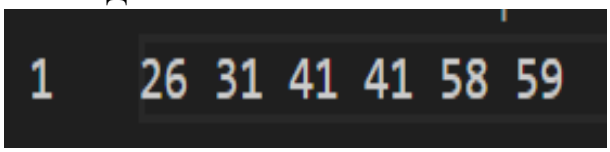


Пример:

Ввод:



Вывод:



	Затраты памяти (Мб)	Время выполнения (с)

Нижняя граница диапазона значений входных данных из текста задачи	0.00058555603	0.000218799978
Верхняя граница диапазона значений входных данных из текста задачи	0.1178522109	0.651409499
Пример	0.000778198	0.00027789999

Сравнение с сортировкой вставками:

Сортировка вставками исполняется значительно быстрее, что делает ее гораздо более эффективной в плане затрат времени

Вставки / Пузырьковая	Время выполнения (с)	
Средние значения	0.293300999	0.65270710
Наихудший случай	0.390035600	0.76158029

(В силу особенностей реализации теста на верхнюю границу диапазона значений результаты оценки времени из них мы можем принять за оценку средних случаев.)

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, познакомились с алгоритмом обратной сортировки вставками (смогли соответствующим заданию образом переделать исходный алгоритм сортировки вставками) и смогли оценить его время выполнения и затраты памяти и сравнить полученные данные с сортировкой вставками и выяснить их относительную эффективность.

Задача №7. Поиск максимального подмассива за линейное время

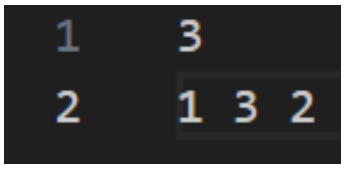
Листинг кода:

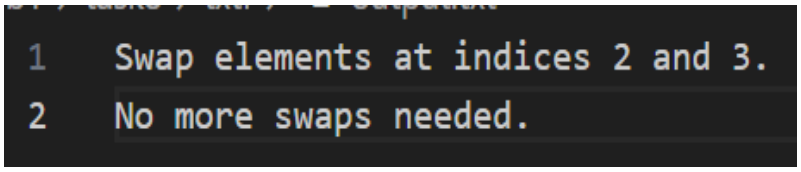
```
def find_max(a):
    max_sum, summ = 0, 0
    for i in range(len(a)):
        if summ == 0:
            start_indx = i
        summ += a[i]
        if max_sum < summ:
            max_sum = summ
            end_indx = i
        if summ < 0:
            summ = 0
    return a[start_indx:end_indx + 1]


with open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task7/txtf/output.txt', 'w') as file:
    f_input = list(map(int, open('C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task7/txtf/input.txt').readline().split()))
    a = find_max(f_input)
    file.write(' '.join(map(str, a)))
```

Реализуем алгоритм swap-сортировки: ведем массив с индексами совершаемых перестановок, задаем два цикла основной и вложенный, в которых с помощью перебора индексов сравниваемых чисел отслеживаем перестановки. Перестановки происходят по следующему принципу: мы выбираем i-тое число, задаем число перед ним, как условно минимальное и сравниваем его с оставшимися числами массива. Если условно минимальное число больше одного из последующих, то значение условно минимальной переменной переназначается на новое наименьшее число, после чего исходное i-тое число и выбранное вышеописанным образом наименьшее меняются местами, попутно в дополнительный массив записываются индексы меняемых между собой переменных. Так происходит до полной сортировки массива. После этого открываем входные и выходные файлы input и output и построчечно записываем туда из массива полученные значения меняемых индексов в соответствии с условием задания. Затем закрываем входной и выходной файлы.

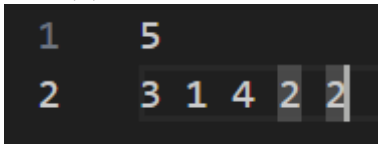
Пример ввода и вывода:

Минимальные значения:
Ввод: Вывод:

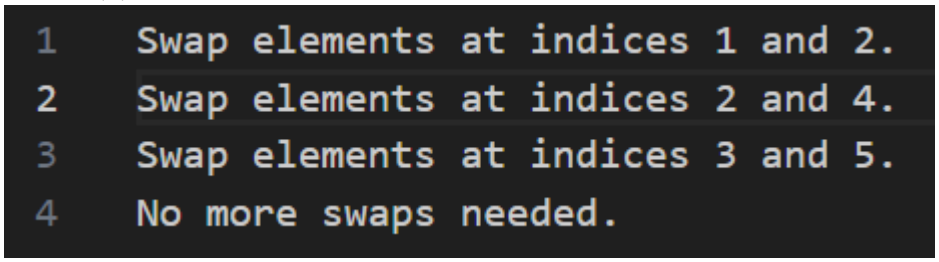




Пример:
Ввод:



Вывод:



	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница диапазона значений входных данных из текста задачи	0.00086021423	0.0002781999
Верхняя граница диапазона значений входных данных из текста задачи	0.872369766	8.116906799
Пример	0.0010166168	0.0004249000

(Как мы можем заметить, на верхней границе входных значений код исполняется очень медленно и не сравним по эффективности с другими алгоритмами.)

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, поняли, что можно создавать собственные, пусть и гораздо менее эффективные алгоритмы сортировки, познакомились с алгоритмом swap-сортировки и смогли оценить его время выполнения и затраты памяти.

Задача №9. Метод Штрассена для умножения матриц
Листинг кода:

```
def read(num_len, file_A, file_B):
    a = []
    b = []
    d = []
    for x in file_A:
        d.append(int(x))
        if len(d) == num_len:
            a.append(d)
            d = []
    for x in file_B:
        d.append(int(x))
        if len(d) == num_len:
            b.append(d)
            d = []
    return [a, b]

def divide(num_len, file_A, file_B):
    g = read(num_len, file_A, file_B)
    new_a= []
    for a in g[0]:
        if len(a) > 2:
            d = []
            for x in a:
                d.append(x)
                if len(d) == 2:
                    new_a.append(d)
                    d = []
        else:
            continue
    if new_a == []:
        new_a = g[0].copy()

    new_b= []
    for b in g[1]:
        if len(b) > 2:
            d = []
            for x in b:
                d.append(x)
                if len(d) == 2:
                    new_b.append(d)
                    d = []
```

```
        else:
            continue
    if new_b == []:
        new_b = g[1].copy()

    return [new_a, new_b]

def block_merge(num_len, file_A, file_B):
    g = divide(num_len, file_A, file_B)
    a = g[0]
    new_a = []
    row = []
    x = 0
    while x != len(a):
        row.append([a[x], a[x + num_len // 2]])
        if len(row) == num_len // 2:
            new_a.append(row)
            row = []
            x += num_len // 2
        x += 1

    b = g[1]
    new_b = []
    row = []
    x = 0
    while x != len(b):
        row.append([b[x], b[x + num_len // 2]])
        if len(row) == num_len // 2:
            new_b.append(row)
            row = []
            x += num_len // 2
        x += 1
    return [new_a, new_b]

def strassen(file_A, file_B):
    mat_a = file_A
    mat_b = file_B
    p1 = mat_a[0][0] * (mat_b[0][1] - mat_b[1][1])
    p2 = (mat_a[0][0] + mat_a[0][1]) * mat_b[1][1]
    p3 = (mat_a[1][0] + mat_a[1][1]) * mat_b[0][0]
    p4 = mat_a[1][1] * (mat_b[1][0] - mat_b[0][0])
    p5 = (mat_a[0][0] + mat_a[1][1]) * (mat_b[0][0] + mat_b[1][1])
    p6 = (mat_a[0][1] - mat_a[1][1]) * (mat_b[1][0] + mat_b[1][1])
    p7 = (mat_a[0][0] - mat_a[1][0]) * (mat_b[0][0] + mat_b[0][1])
```

```
    c = []
    c.append([p5 + p4 - p2 + p6, p1 + p2])
    c.append([p3 + p4, p1 + p5 - p3 - p7])
    return c

def strassen_apply(num_len, file_A, file_B):
    g = block_merge(num_len, file_A, file_B)
    a = g[0]
    b = g[1]
    final = []
    for x in range(len(a)):
        row_a = a[x]
        row_b = b[x]
        for y in range(len(row_a)):
            st_a = row_a[y]
            st_b = row_b[y]
            c = strassen(st_a, st_b)
            final.append(c)
    return(final)

output_f = open("C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task9/txtf/output.txt", 'w')
input_f = open("C:/Users/ВЕРОНИКА/Desktop/показ/lab-
aisd/lab2/task9/txtf/input.txt")
num_len = int(input_f.readline())
g = input_f.readline()
h = input_f.readline()
file_A = list(map(int, list(g.split(' '))))
file_B = list(map(int, list(h.split(' '))))

output_f.write(' '.join(map(str, strassen_apply(num_len, file_A,
file_B))))

output_f.close()
input_f.close()
```

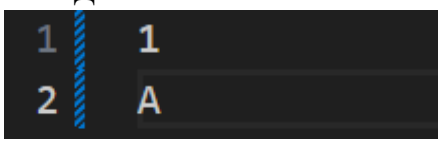
Объявляем функцию, создающую палиндром из строки – возвращаем минимальный по алфавиту элемент строки если никакой из элементов в ней не повторяется, иначе считаем повторяемые и неповторяемые элементы и ведем их учет, попутно проверяя, подходящие ли знаки находятся в строке – иначе выводим соответствующую ошибку. После этого объединяем знаки в строку в зависимости от количества повторений символов всеми возможными способами, вычисляем наиболее длинную из итоговых строк, сохраняем и возвращаем ее значение как итог функции. Открываем входные и выходные файлы input и output, считываем оттуда необходимые для вызова

функции значения, вызываем функцию и заносим возвращенное ей значение в файл, после чего закрываем входной и выходной файлы.

Пример ввода и вывода:

Минимальные значения:

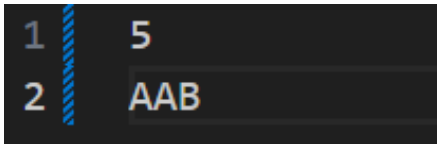
Ввод:



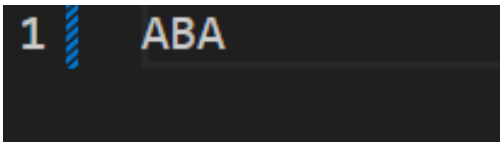
Вывод:

Пример 1:

Ввод:

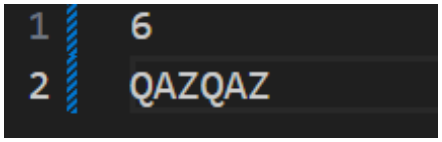


Вывод:

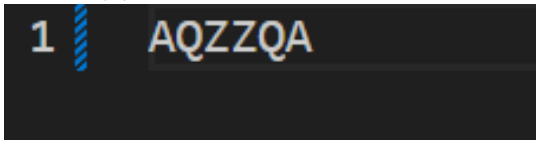


Пример 2:

Ввод:

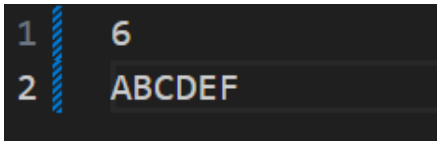


Вывод:



Пример 3:

Ввод:



Вывод:

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница диапазона значений входных данных из текста задачи	0.00019909	0.000243900
Верхняя граница диапазона значений входных данных из текста задачи	10.847418785	1.32427740
Пример 3	0.0005779266	0.000740051
Пример 2	0.0007944107	0.00020120

Пример 1	0.000795364	0.0001746

Вывод по задаче: Мы научились объявлять функции и использовать их с передачей им различных переменных, поняли, что можно создавать собственные, пусть и гораздо менее эффективные алгоритмы сортировки и смогли использовать их для решения практических задач вроде составления палиндромов, оценили время работы таких алгоритмов и их затраты памяти.

Вывод: Мы рассмотрели несколько различных алгоритмов сортировки, научились представлять их в коде на языке Python, выполнили несколько практических заданий на основе сортировок и вспомнили работу с объявлением функций и библиотекой random. Ко всем выполненным кодам вычислили время и затраты памяти на исполнение алгоритма, что позволяет нам сравнить данные и вычислить наиболее эффективный алгоритм.