

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
Вариант 7

Выполнила:
Заботкина М.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №2. Сортировка слиянием+	6
Задача №9. Метод Штрассена для умножения матриц	9
Дополнительные задачи	14
Задача №3. Число инверсий	14
Задача №4. Бинарный поиск	17
Задача №7. Поиск максимального подмассива за линейное время	20
Вывод	24

Задачи по варианту

Задача №1. Сортировка слиянием

Перепишите процедуру *Merge* так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива *L* или *R* скопированы обратно в массив *A*, после чего в этот массив копируются элементы, оставшиеся в непустом массиве. или перепишите процедуру *Merge* (и, соответственно, *Merge-sort*) так, чтобы в ней не использовались значения границ и середины - *p*, *r* и *q*.

Формат входного файла (*input.txt*). В первой строке входного файла содержится число *n* ($1 \leq n \leq 2 \cdot 10^4$) — 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.

Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, 10^4 , 10^5 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний.

```
def merge(left, right):
    res = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            res.append(left[i])
            i += 1
        else:
            res.append(right[j])
            j += 1

    while i < len(left):
        res.append(left[i])
        i += 1

    while j < len(right):
        res.append(right[j])
        j += 1

    return res

def merge_sort(a):
    if len(a) <= 1:
        return a

    mid = len(a) // 2
    left = merge_sort(a[:mid])
```

```

        right = merge_sort(a[mid:])
        return merge(left, right)

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline())
        a = list(map(int, f.readline().split()))

        if not 1 <= n <= 2 * 10**4:
            with open(output_f, 'w') as f:
                f.write('Число не входит в диапазон')
            return

        for element in a:
            if abs(element) > 10**9:
                with open(output_f, 'w') as f:
                    f.write('Число превосходит допустимое значение')
                return

        sorted_a = merge_sort(a)
        with open(output_f, 'w') as f:
            f.write(' '.join(map(str, sorted_a)))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) Начинаем с написания функции `merge()`, которая отвечает за объединение двух списков в один. Для этого создаём переменную `res`. Для левого и правого списка будем использовать `i` и `j` для обозначения индексов. Цикл будет продолжаться до тех пор, пока оба списка непустые. Если элемент левого списка меньше или равен элементу правого списка, то он добавляется в общий список `res`, а индекс увеличивается на единицу, в ином случае происходит наоборот. Следующие два цикла предназначены для добавления в общий список оставшихся элементов, если они остались в списках `left` и `right`.

2) Функция `merge_sort()` отвечает за разделения массива на две части и дальнейшее применение предыдущей функции. Если список пуст или в нём только один элемент, то вернётся этот же список. Находим центр списка и делим массив на две части. Так как функция рекурсивна, то деление списков пополам будет до тех пор, пока не останется в каждом по одному элементу. Затем, начиная с базового случая, начинается процесс слияния. Два подмассива объединяются с помощью функции `merge()` в один отсортированный массив.

3) Функция `check_and_write()`, в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью `with open` файл будет открыт для чтения и по умолчанию закроется после считывания значений: `n` – количество элементов в массиве, `a` – сам массив, где каждый элемент из строкового вида перейдёт к целочисленному. Если числа не

пройдут проверку, то появится в файле output соответствующая запись и программа прекратит работу. Вызываем функцию `merge_sort()` и записываем в выходной файл.

4) Чтобы функция `check_and_write()` начала работать, нужно её вызвать. Передаем ей нужные файлы.

Результат работы кода на примерах из текста задачи:

```

input.txt
1 1000
2 -694654849 -948748459 190742659 343496038 -712758452 17906854 -6

output.txt
1 -999196476 -998805559 -998275568 -995646935 -993415686 -992681941

input.txt
1 10000
2 174216663 -510031807 -121142096
   190794646 969964007 45482541
   -528385698 -290755844 -979540196

output.txt
1 -999999034 -999826272 -999493865
   -999241996 -999068003 -998836053
   -998540118 -998379954 -998277525
   -997983821 -997239152 -997229422
  
```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt
1 20000
2 -41010739 180947201 -35892636 -379569207
   -756163001 -767480482 922703259 -33234068
   -817031149 140685725 -807247532 634420480
   990028331 557883721 555245790 21565245

output.txt
1 -999992384 -999890065 -999874274 -999803534 -999712817 -999510889 -999377589
   -999361808 -999357024 -999167379 -999126357 -999111400 -999111254 -999063252
   -999004016 -998833100 -998690469 -998404295 -998146262 -998139747 -998072217
   -998062649 -998009066 -997981358 -997835929 -997758098 -997730563 -997552843
   -997390190 -997100816 -997100809 -997100663 -997085456 -997052650 -997014443

output.txt
1 31753565
2 31753565
  
```

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.009844199987128377	0.01715373992919922
Средние значения	0.02070979995187372	0.0680389404296875
Верхняя граница диапазона значений входных данных из	0.7625499999849126	2.349956512451172

Вывод по задаче: программа работает быстро даже при больших значениях и использует минимум памяти, без использования граничных значений код выглядит даже более понятно.

Задача №2. Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием. Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

- *Формат входного файла (input.txt).* В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .

- *Формат выходного файла (output.txt).* Выходной файл состоит из нескольких строк.

- В последней строке выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.

- Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: I_f , I_l , V_f , V_l , где I_f — индекс начала области слияния, I_l — индекс конца области слияния, V_f — значение первого элемента области слияния, V_l — значение последнего элемента области слияния.

- Все индексы начинаются с единицы (то есть, $1 \leq I_f \leq I_l \leq n$).

Индексы области слияния должны описывать положение области слияния в исходном массиве! Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.

- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб

```
def merge(a, l, r, l_start, r_end, f):
    res = []
    i = j = 0
    while i < len(l) and j < len(r):
        if l[i] <= r[j]:
```

```

        res.append(l[i])
        i += 1
    else:
        res.append(r[j])
        j += 1

    while i < len(l):
        res.append(l[i])
        i += 1

    while j < len(r):
        res.append(r[j])
        j += 1

    a[l_start:r_end + 1] = res
    f.write(f"{l_start + 1} {r_end + 1} {a[l_start]} {a[r_end]}\n")

def merge_sort(a, l, r, f):
    if l < r:
        mid = (l + r) // 2
        merge_sort(a, l, mid, f)
        merge_sort(a, mid + 1, r, f)
        merge(a, a[l:mid + 1], a[mid + 1:r + 1], l, r, f)
    return a

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline())
        a = list(map(int, f.readline().split()))

    if not 1 <= n <= 10**5:
        with open(output_f, 'w') as f:
            f.write('Число не входит в диапазон')
        return

    for element in a:
        if abs(element) > 10**9:
            with open(output_f, 'w') as f:
                f.write('Число превосходит допустимое значение')
            return

    with open(output_f, 'w') as f:
        merge_sort(a, 0, n - 1, f)
        f.write(' '.join(map(str, a)))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) Снова всё начинается с функции `merge()`. В этот раз для вывода граничных элементов чуть изменим код и будем использовать дополнительные значения: `l_start` и `r_end`, и `a` – исходный массив. Цикл будет продолжаться до тех пор, пока оба списка непустые. Если элемент левого списка меньше или равен элементу правого списка, то он добавляется в общий список `res`, а индекс увеличивается на единицу, в ином случае происходит наоборот. Следующие два цикла предназначены для добавления в общий список оставшихся элементов, если они остались в списках `l` и `r`. Исходный массив `a` обновляется: часть массива от `l_start` до

r_end (то есть границы слияния) заменяется на отсортированный результат из res. Функция записывает в файл индексы и значения первого и последнего элементов в слиянии.

2) Функция merge_sort() отвечает за разделения массива на две части и дальнейшее применение предыдущей функции. Так как здесь используются индексы, то проверяем, что левый индекс меньше правого, иначе массив имеет только один элемент или массив пустой, и его разделить нельзя. Ищем центральный индекс для разделения по нему массива. Затем каждый массив делится ещё на два массива и так до момента, пока изначальный массив не будет состоять из отдельных элементов. Затем используется предыдущая функция для объединения подмассивов в единый массив.

3) Функция check_and_write(), в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью with open файл будет открыт для чтения и по умолчанию закроется после считывания значений: n – количество элементов в массиве, a – сам массив, где каждый элемент из строкового вида перейдёт к целочисленному. Если числа не пройдут проверку, то появится в файле output соответствующая запись и программа прекратит работу. Вызываем функцию merge_sort() и записываем в выходной файл.

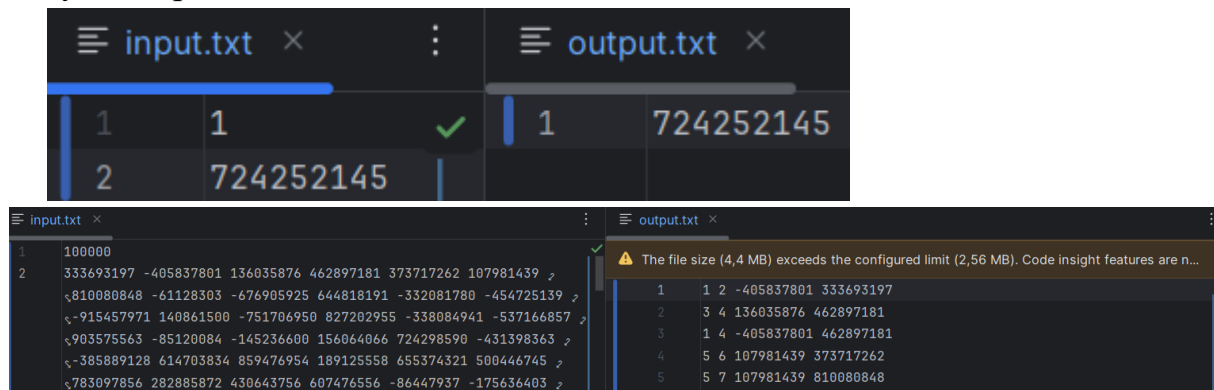
4) Чтобы функция check_and_write() начала работать, нужно её вызвать. Передаём ей нужные файлы.

Результат работы кода на примерах из текста задачи:

input.txt	output.txt
1 4	1 1 2 7 9
2 9 7 5 8	2 3 4 5 8
	3 1 4 5 9
	4 5 7 8 9

input.txt	output.txt
1 10	1 1 2 1 8
2 1 8 2 1 4 7 3 2 3 6	2 1 3 1 8
	3 4 5 1 4
	4 1 5 1 8
	5 6 7 3 7
	6 6 8 2 7
	7 9 10 3 6
	8 6 10 2 7
	9 1 10 1 8
	10 1 1 2 2 3 3 4 6 7 8

Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи	0.010401399922557175	0.017157554626464844
Пример из задачи	0.0011301001068204641	0.01718616485595703
Верхняя граница диапазона значений входных данных из текста задачи	8.18310439994093	10.772686958312988

Вывод по задаче: из-за большого количества записей в файл при больших значениях время выполнения программы увеличивается, но при этом используемая память остаётся в пределах допустимого.

Задача №9. Метод Штрассена для умножения матриц

Цель. Применить метод Штрассена для умножения матриц и сравнить его с простым методом.

- *Формат входа. Стандартный ввод или input.txt. Первая строка - размер квадратных матриц n для умножения. Следующие строки соответственно сами значения матриц A и B .*

- *Формат выхода. Стандартный вывод или output.txt. Матрица $C = A \cdot B$.*

Простой метод:

```
def matrix_mult(n, X, Y):
    Z = [[0] * n for i in range(n)]
    for i in range(n):
```

```

        for j in range(n):
            for k in range(n):
                Z[i][j] += X[i][k] * Y[k][j]

    return Z

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline())

        A = []
        for i in range(n):
            A.append(list(map(int, f.readline().split())))

        B = []
        for i in range(n):
            B.append(list(map(int, f.readline().split())))

    C = matrix_mult(n, A, B)
    with open(output_f, 'w') as f:
        for r in C:
            f.write(' '.join(map(str, r)) + '\n')

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) Простой метод заключается в стандартном перемножении матриц. Здесь создается пустая матрица Z размером $n \times n$, заполненная нулями. Каждый элемент результирующей матрицы $Z[i][j]$ вычисляется как сумма произведений элементов строки i матрицы X и столбца j матрицы Y . Это соответствует правилу умножения матриц.

Метод Штрассена:

```

def add_matrix(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A))] for i in range(len(A))]

def sub_matrix(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A))] for i in range(len(A))]

def strassen(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]

    mid = n // 2

    A11, A12, A21, A22 = [r[:mid] for r in A[:mid]], [r[mid:] for r in A[:mid]], [r[:mid] for r in A[mid:]], [r[mid:] for r in A[mid:]]
    B11, B12, B21, B22 = [r[:mid] for r in B[:mid]], [r[mid:] for r in B[:mid]], [r[:mid] for r in B[mid:]], [r[mid:] for r in B[mid:]]

    P1 = strassen(A11, sub_matrix(B12, B22))
    P2 = strassen(add_matrix(A11, A12), B22)
    P3 = strassen(add_matrix(A21, A22), B11)
    P4 = strassen(A22, sub_matrix(B21, B11))
    P5 = strassen(add_matrix(A11, A22), add_matrix(B11, B22))

```

```

P6 = strassen(sub_matrix(A12, A22), add_matrix(B21, B22))
P7 = strassen(sub_matrix(A11, A21), add_matrix(B11, B12))

C11 = add_matrix(sub_matrix(add_matrix(P5, P4), P2), P6)
C12 = add_matrix(P1, P2)
C21 = add_matrix(P3, P4)
C22 = sub_matrix(sub_matrix(add_matrix(P1, P5), P3), P7)

C = [[0] * n for i in range(n)]
for i in range(mid):
    C[i][:mid], C[i][mid:], C[i + mid][:mid], C[i + mid][mid:] =
C11[i], C12[i], C21[i], C22[i]

return C

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline())

        A = []
        for i in range(n):
            A.append(list(map(int, f.readline().split())))

        B = []
        for i in range(n):
            B.append(list(map(int, f.readline().split())))

        C = strassen(A, B)
        with open(output_f, 'w') as f:
            for r in C:
                f.write(' '.join(map(str, r)))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) Для начала напомним функции для сложения и для вычитания элементов матриц. В дальнейшем это пригодится для реализации метода Штрассена. Переменная i – количество строк, j – количество столбцов (то есть количество элементов в одной строке), элементы с одинаковыми индексами по строке и столбцу складываются (или вычитаются, если функция `sub_matrix()`) и эта матрицу функция возвращает.

2) В функции `Strassen()` реализован сам алгоритм. На вход функция получает n – размерность матрицы (так как она квадратная по условию, то количество строк и столбцов совпадает). Если размер матрицы равен единице, то перемножаются нулевые элементы. В ином случае ищем середину строк и столбцов, поделив n нацело. Затем делим обе матрицы на 4 подматрицы. $P1$ - $P7$ – это промежуточные умножения матриц по методу Штрассена. После вычисления промежуточных матриц, формируем подматрицы результирующей матрицы C . Создаём матрицу C , заполненную нулями и заменяем эти нули с помощью цикла на вычисленные ранее подматрицы.

3) Функция `check_and_write()`, в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью `with open` файл будет открыт для чтения и по умолчанию закроется после считывания значений: `n` – размер матриц, `A` и `B` – две матрицы. Если числа не пройдут проверку, то появится в файле `output` соответствующая запись и программа прекратит работу. Вызываем функцию `strassen()` и записываем в выходной файл.

4) Чтобы функция `check_and_write()` начала работать, нужно её вызвать. Передав ей нужные файлы.

Результат работы кода на примерах (простой метод):

input.txt	output.txt
1 4	1 7005 11089 13890 8162
2 44 84 13 83	2 7303 9093 15841 10920
3 95 71 86 54	3 3841 3563 11005 6973
4 69 60 75 1	4 8044 12830 17717 10645
5 83 92 6 98	5
6 6 4 67 47	
7 25 37 92 29	
8 25 13 11 26	
9 52 92 37 40	
10	

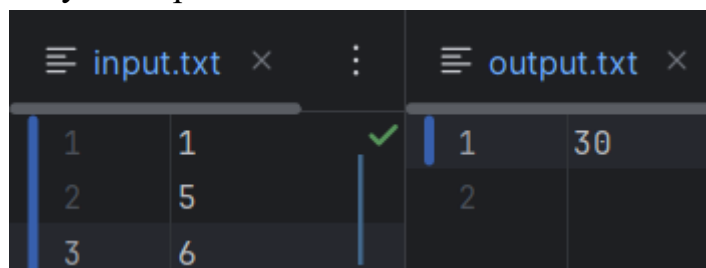
Результат работы кода на минимальных значениях (простой метод):

input.txt	output.txt
1 1	1 962
2 26	2
3 37	

Результат работы кода на примерах (метод Штрассена):

input.txt	output.txt
1 4	1 7005 11089 13890 8162
2 44 84 13 83	2 7303 9093 15841 10920
3 95 71 86 54	3 3841 3563 11005 6973
4 69 60 75 1	4 8044 12830 17717 10645
5 83 92 6 98	5
6 6 4 67 47	
7 25 37 92 29	
8 25 13 11 26	
9 52 92 37 40	

Результат работы кода на минимальных значениях (метод Штрассена):



Простой метод

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.00045930000487715006	0.017274856567382812
Пример	0.005189000046811998	0.017026901245117188

Метод Штрассена

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0005732000572606921	0.017274856567382812
Пример	0.002982800011523068	0.018169403076171875

Вывод по задаче: на минимальных значениях разницы по времени и памяти не наблюдается, но чем больше будет матрица, тем более выгодным становится метод Штрассена, хотя и будет он использовать больше памяти.

Дополнительные задачи

Задача №3. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$. Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной.

Например, в отсортированном массиве число инверсий равно 0, а в массиве, отсортированном наоборот - каждые два элемента будут составлять инверсию (всего $n(n-1)/2$).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем. Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

- Формат входного файла (*input.txt*). В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- Формат выходного файла (*output.txt*). В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

```
def merge(l, r):
    res = []
    i = j = inversions = 0

    while i < len(l) and j < len(r):
        if l[i] <= r[j]:
            res.append(l[i])
            i += 1
        else:
            res.append(r[j])
            inversions += len(l) - i
            j += 1

    res.extend(l[i:])
    res.extend(r[j:])

    return res, inversions

def merge_sort(a):
    if len(a) <= 1:
        return a, 0

    mid = len(a) // 2
    l, l_inversions = merge_sort(a[:mid])
    r, r_inversions = merge_sort(a[mid:])
    merged, split_inv = merge(l, r)
```

```

        return merged, l_inversions + r_inversions + split_inv

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline())
        a = list(map(int, f.readline().split()))

    if not 1 <= n <= 10 ** 5:
        with open(output_f, 'w') as f:
            f.write('Число не входит в диапазон')
        return

    for element in a:
        if abs(element) > 10 ** 9:
            with open(output_f, 'w') as f:
                f.write('Число превосходит допустимое значение')
            return

    merged, inversions = merge_sort(a)

    with open(output_f, 'w') as f:
        f.write(str(inversions))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) Функция `merge()` сливает два отсортированных списка и считает инверсии. Инверсия возникает, когда элемент из правой части меньше элемента из левой. Когда такое происходит, количество инверсий увеличивается на число оставшихся элементов в левой части. После слияния функция возвращает отсортированный список и число инверсий.

2) `Merge_sort`: если массив содержит 1 или 0 элементов, он уже отсортирован, и инверсий нет. Массив делится на две части: левая и правая. Затем каждая из них рекурсивно делится и подсчитываются инверсии в обеих частях. После того, как обе части массива отсортированы, их нужно слить воедино с помощью функции `merge`. Во время слияния подсчитываются инверсии, возникающие между элементами из левой и правой части.

3) Функция `check_and_write()`, в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью `with open` файл будет открыт для чтения и по умолчанию закроется после считывания значений: `n` – количество элементов в массиве, `a` – сам массив, где каждый элемент из строкового вида перейдёт к целочисленному. Если числа не пройдут проверку, то появится в файле `output` соответствующая запись и программа прекратит работу. Вызываем функцию `merge_sort()` и записываем в выходной файл число инверсий.

4) Чтобы функция `check_and_write()` начала работать, нужно её вызвать, передав ей нужные файлы.

Результат работы кода на примерах из текста задачи:

```

input.txt x
1 10
2 1 8 2 1 4 7 3 2 3 6

output.txt x
1 17

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt x
1 100000
2 -669842202 -297784176 -213490012 -804060939 -702459427 -773191526
  -712494261 964232440 -181933563 -427585931 874334577 593865171
  -547565722 836492606 333843430 106904518 711940008 -990369426
  -475547745 -183482810 -177001208 -476211416 -837238619 -182954111
  -462182187 747303872 -264446310 66003285 -171544522 -10788230

output.txt x
1 2508505951

input.txt x
1 1
2 45

output.txt x
1 0

```

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0008059000829234719	0.01717662811279297
Пример из задачи	0.0006343999411910772	0.017210006713867188
Верхняя граница диапазона значений входных данных из текста задачи	5.866182099911384	9.777156829833984

Вывод по задаче: алгоритм на основе сортировки слиянием эффективно справляется с подсчетом инверсий даже на больших объемах данных, оставаясь в пределах допустимых ограничений по времени и памяти.

Задача №4. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < \dots < a_{n-1}$ из n различных положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k , $1 \leq k \leq 10^5$ и k положительных целых чисел b_0, \dots, b_{k-1} , $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k - 1$ вывести индекс $0 \leq j \leq n - 1$, такой что $a_i = b_j$ или -1 , если такого числа в массиве нет.
- **Ограничение по времени.** 2сек.
- **Ограничение по памяти.** 256 мб.
- **Пример:**

```
def binary_search(a, target):
    l = 0
    r = len(a)

    while l <= r:
        mid = (l + r) // 2
        if a[mid] == target:
            return mid
        elif a[mid] < target:
            l = mid + 1
        else:
            r = mid - 1

    return -1

def check_and_write(input_f, output_f):
    with open(input_f, 'r') as f:
        n = int(f.readline())
        a = list(map(int, f.readline().split()))
        k = int(f.readline())
        b = list(map(int, f.readline().split()))

    if not 1 <= n <= 10 ** 5:
        with open(output_f, 'w') as f:
            f.write('Число не входит в диапазон')
        return

    for element in a:
        if not (1 <= element <= 10 ** 9):
            with open(output_f, 'w') as f:
                f.write('Число превосходит допустимое значение')
            return
```

```

if not 1 <= k <= 10 ** 5:
    with open(output_f, 'w') as f:
        f.write('Число не входит в диапазон')
    return

for element in b:
    if not (1 <= element <= 10 ** 9):
        with open(output_f, 'w') as f:
            f.write('Число в массиве b превосходит допустимое
значение')
        return

res = [binary_search(a, x) for x in b]
with open(output_f, 'w') as f:
    f.write(' '.join(map(str, res)))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) l — начало массива, r — конец массива. Этот цикл будет выполняться до тех пор, пока левая граница l не станет больше правой границы r . Как только $l > r$, это означает, что элемент не был найден, и цикл завершится. На каждой итерации индекс середины массива вычисляется как целая часть от деления суммы индексов l и r на два. Это позволяет делить массив пополам для поиска. Если элемент, находящийся в середине массива $a[mid]$, равен искомому значению $target$, функция возвращает индекс этого элемента mid . Если элемент в середине не равен $target$, алгоритм решает, в какой половине массива продолжить поиск. Если центральный элемент меньше нужного, то мы сдвигаем левую границу на одну позицию вправо от середины, чтобы продолжить поиск в правой части. Иначе искомый элемент находится в левой части массива. Правая граница сдвигается на одну позицию влево от середины.

2) Функция `check_and_write()`, в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью `with open` файл будет открыт для чтения и по умолчанию закроется после считывания значений: n — количество элементов в массиве a , a — массив длиной n , k — количество элементов в массиве b , b — массив длиной k . Если числа не пройдут проверку, то появится в файле `output` соответствующая запись и программа прекратит работу. Для каждого элемента x из массива b выполняется бинарный поиск в массиве a с помощью функции `binary_search`. Результаты поиска сохраняются в список `res`.

3) Чтобы функция `check_and_write()` начала работать, нужно её вызвать. Передав ей нужные файлы.

Результат работы кода на примерах:

```

input.txt x
1 5
2 1 5 8 12 13
3 5
4 8 1 23 1 11
-
output.txt x
1 2 0 -1 0 -1

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt x
1 100000
2 872218606 789439174 19736174 47810613 501858140 693464619
  358337766 112069193 141653409 674840682 731866537 85174945
  18800707 287651619 475428631 572520383 806515015 882315185
  398197549 633379731 124695847 289818713 38883934 210403452
  570010000 888701100 370101051 151150512 307701010 101007000
-
output.txt x
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-
input.txt x
1 1
2 5
3 1
4 4
-
output.txt x
1 -1

```

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0008338999468833208	0.017099380493164062
Пример из задачи	0.0010381999891251326	0.017099380493164062
Верхняя граница диапазона значений входных данных из текста задачи	5.544071300071664	13.554971694946289

Вывод по задаче: алгоритм показывает высокую эффективность, особенно на малых и средних входных данных, оставаясь в пределах

допустимых ограничений по времени и памяти. При максимальных значениях работа замедляется.

Задача №7. Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива $A[1..j]$, распространите ответ на поиск максимального подмассива, заканчивающегося индексом $j + 1$, воспользовавшись следующим наблюдением: максимальный подмассив массива $A[1..j + 1]$ представляет собой либо максимальный подмассив массива $A[1..j]$, либо подмассив $A[i..j + 1]$ для некоторого $1 \leq i \leq j + 1$. Определите максимальный подмассив вида $A[i..j + 1]$ за константное время, зная максимальный подмассив, заканчивающийся индексом j .

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание случайного массива чисел, аналогично задаче №1 (в этом случае формат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично задаче №6.

```
def max_sub(a):
    max_sum = curr_sum = a[0]
    start = end = 0

    for i in range(1, len(a)):
        if curr_sum + a[i] < a[i]:
            curr_sum = a[i]
            start = i
        else:
            curr_sum += a[i]

        if curr_sum > max_sum:
            max_sum = curr_sum
            end = i

    return a[start:end + 1]

def check_and_write(input_f, output_f):
    with open(input_f) as f:
        n = int(f.readline().strip())
        a = list(map(int, f.readline().strip().split()))

    if not 1 <= n <= 2 * 10**4:
        with open(output_f, 'w') as f:
            f.write('Число не входит в диапазон')
        return

    for element in a:
        if abs(element) > 10**9:
```

```

        with open(output_f, 'w') as f:
            f.write('Число превосходит допустимое значение')
        return

    maxi = max_sub(a)
    with open(output_f, 'w') as f:
        f.write(' '.join(map(str, maxi)))

check_and_write('../txtf/input.txt', '../txtf/output.txt')

```

1) `max_sum` и `curr_sum` оба инициализируются первым элементом массива `a[0]`. `max_sum` — хранит максимальную сумму, которую удалось найти на текущий момент. `curr_sum` — сумма текущего подмассива. `start` и `end` — индексы начала и конца подмассива, который имеет максимальную сумму на текущий момент. Изначально они оба указывают на первый элемент массива. Цикл проходит по всем элементам массива, начиная с индекса 1, чтобы вычислять текущие суммы и искать подмассив с наибольшей суммой. Если сумма текущего подмассива плюс элемент меньше самого элемента `a[i]`, выгоднее начать новый подмассив с этого элемента. В этом случае, текущая сумма `curr_sum` обновляется на значение текущего элемента `a[i]`. Индекс начала подмассива обновляется на текущий индекс `i`. В противном случае, к текущей сумме добавляется элемент `a[i]`, и подмассив продолжается. Если текущая сумма `curr_sum` больше, чем максимальная сумма `max_sum`, найденная до этого момента, обновляются

2) Функция `check_and_write()`, в отличие от предыдущих лабораторных работ, теперь объединяет в себе чтение файла, проверку значений на допустимость и запись ответа в выходной файл. С помощью `with open` файл будет открыт для чтения и по умолчанию закроется после считывания значений: `n` — количество элементов в массиве, `a` — сам массив, где каждый элемент из строкового вида перейдёт к целочисленному. Если числа не пройдут проверку, то появится в файле `output` соответствующая запись и программа прекратит работу. Вызываем функцию `max_sub()` и записываем в выходной файл.

3) Чтобы функция `check_and_write()` начала работать, нужно её вызвать. Передав ей нужные файлы.

Результат работы кода на примерах из текста задачи:

input.txt ×

1

1000

2

-359188684 -54746819 288224968 3918399 939550795 61231888
557739376 7654671 55720105 -237238619 -224582532 -288199721
-250549438 714962172 -70694375 -168500333 825982198 -25410500
100302270 154874991 -13369057 416818546 -975253210 -765223571
-319365817 689495096 14907209 -272672749 120470380 334609233
-950036063 1279956 -810161462 932323002 409169 278050998

output.txt ×

1

262072640 331046134 799618795 518902427 -152620459 45930622 ↗
↖-624181895 -936949117 756001200 574154232 -183620093 -613120343 ↗
↖-128071427 -495305581 916274042 346630448 -330164570 838604797 ↗
↖-455359100 862850154 -508784812 664997623 764816296 753653123 ↗
↖769088209 789543459 -759427525 73820552 -387269905 220698183 ↗
↖735951969 -527003877 590117024 782189558 -941480811 324603575 ↗

input.txt ×

1

10000

2

-422141435 -432295929 -632441511 812721114 2724125 701904582
828755111 -372543974 -795605163 959847024 667041360 -420822876
611528165 -854460608 666036222 497892142 -932474056 921221981
305275448 323140985 -260455622 -541706657 -940903699 -214741953
19002522 -24367798 -319256479 343044052 -829815909 -322944589
-274005740 -835995134 -48204722 -451975349 -474554838 608724868

output.txt ×

1

833894710 920398940 240130012 -421006102 759408229 201590107 ↗
↖-629609734 596878992 811868273 -442748491 280603784 -187236537 ↗
↖189215495 578225990 818938008 660735691 -409723218 282496637 ↗
↖936883431 194553717 674206380 -346109901 122947754 833385470 ↗
↖517337689 -61867868 677552482 367337308 -778284573 898675507 ↗
↖529332420 -765359515 277145510 241841624 687064068 -103079673 ↗

Результат работы кода на максимальных и минимальных значениях:

input.txt ×

1

20000

2

-899266030 -656889818 -474091575 -955242594 643185159 732124652
429959524 -781358713 -341993970 -320742657 175866787 -162752629
219996583 897734534 5640644 356211792 -342592290 -254165355
-821488722 -103290159 -738515582 251214184 -108599948 625785346
-843548282 480646919 -817919168 -305056474 668112706 -115053239
342550581 684917862 -999973131 243384704 -438891637 837720031
786071245 -740390289 310394087 -741999335 862153676 411924448

output.txt ×

1

668112706 -115053239 342550581 684917862 -999973131 243384704 ↗
↖-438891637 837720031 786071245 -740390289 310394087 -741999335 ↗
↖862153676 411924448 -814166445 174180908 985248292 963062881 ↗
↖864728029 276429413 241805208 728441018 -419388763 -690516366 ↗
↖-895228997 190986976 -544115800 112077510 420968829 -673920448 ↗

input.txt ×

1

1

2

6

output.txt ×

1

6

	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из	0.0023459000512957573	0.017127037048339844

текста задачи		
Средние значения	0.01177119999192655	0.11244773864746094
Верхняя граница диапазона значений входных данных из текста задачи	0.16359390004072338	2.111726760864258

Вывод по задаче: алгоритм выполняется за линейное время и потребляет умеренное количество памяти, что делает его эффективным для решения задачи на больших объемах данных.

Вывод

Алгоритм сортировки слиянием демонстрирует стабильную производительность даже на больших массивах данных. Основное преимущество метода декомпозиции — это его способность разбивать сложные задачи на меньшие подзадачи, которые легко решаются, а затем объединяются в конечное решение.