?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 21

Выполнила:
Савченко А.С.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета **2**
Задачи по варианту **3**
Задача №1. Сортировка слиянием 3
Задача №3. Число инверсий 8
Задача №4.Бинарный поиск 12
Задача №5. Представитель большинства 15
Задача №8. Умножение многочленов 18
Вывод (по всей лабораторной) **22**

Задачи по варианту

Задача №1. Сортировка слиянием

1 задача. Сортировка слиянием

1. Используя *псевдокод* процедур `Merge` и `Merge-sort` из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры:

   - **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.

2. 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. Сравните, например, с сортировкой вставкой на этих же данных.

3. 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.

Текст задачи:

1 задача. Сортировка слиянием

1. Используя *псевдокод* процедур `Merge` и `Merge-sort` из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры:

   - **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.

2. 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. Сравните, например, с сортировкой вставкой на этих же данных.

3. 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.

Листинг кода (именно листинг, а не скрины):

```
#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.
def merge(A, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L = [0] * n1
    R = [0] * n2
```

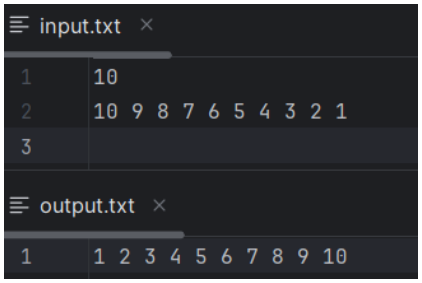
```
    for i in range(n1):
        L[i] = A[p + i]
    for j in range(n2):
        R[j] = A[q + 1 + j]

    i = j = 0
    k = p

    # copy элем-ы из массивов L , R обратно в массив A
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            j += 1
        k += 1

    # copy оставшиеся эл массива L
    while i < n1:
        A[k] = L[i]
        i += 1
        k += 1

    # copy элементы массива R, если они есть
    while j < n2:
        A[k] = R[j]
        j += 1
        k += 1


def merge_sort(A, p, r):
    if p < r:
        q = (p + r) // 2
        merge_sort(A, p, q)
```

```
        merge_sort(A, q + 1, r)
        merge(A, p, q, r)
    return A


def main():
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
                                A     =     list(map(int,
file.readline().strip().split()))

    merge_sort(A, 0, n - 1)

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


if __name__ == "__main__":
    main()
```

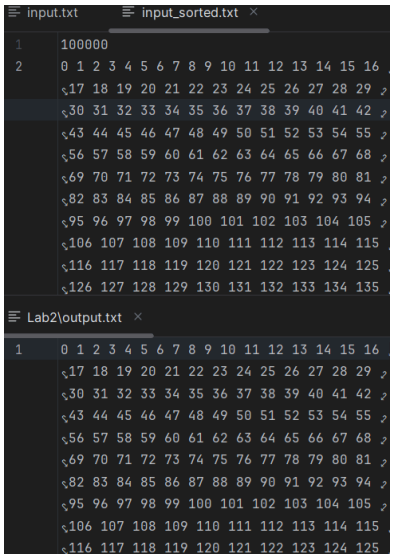
Текстовое объяснение решения:

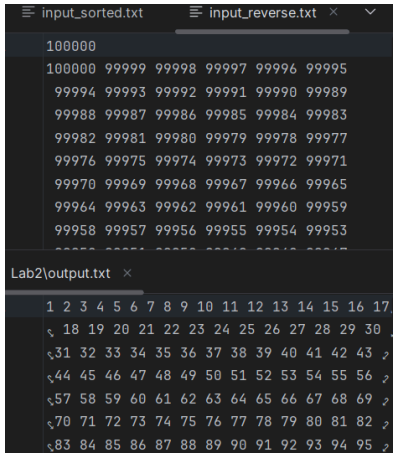
Для алгоритма будем пользоваться принципом Разделяй и Властвуй. А именно рекурсивно делить массив на 2 половины и потом сливать их снова в один отсортированный массив. Элементы двух половин массива левой и правой последовательно сравниваются, а меньший копируется в исходный отсортированный массив.

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
Пример из задачи	0.0005311 с	15.17578125 Мб
1000 чисел обратно отсортированный массив	0.0030683 с	15.109375 Мб
10**4 обратно отсортированный	0.0264194 с	15.87890625 Мб
10**5 обратно отсортированный	0.3455843 с	16.9375 Мб
1000 сортированный	0.0031792 с	15.28125 Мб
10**4 сортированный	0.0274096 с	15.88671875 Мб
10**5 сортированный	: 0.3157147 с	16.88671875 Мб

Вывод по задаче:
Исходя из проведенных тестов можно сделать выводы:

Для "маленьких" 10**3 и "средних" 10**4 массивов разница во времени выполнения и потреблении памяти между обратно отсортированным и уже отсортированным массивами незначительна.

Для "больших" массивов (10**5) время выполнения и потребление памяти обратно отсортированного массива немного больше, чем для уже отсортированного, но разница все еще незначительна.

Результаты подтверждают стабильность работы алгоритма как отсортированного массива, так и для обратно отсортированного. Сложность остается O(n*log(n)).

Задача №3. Число инверсий

Текст задачи:

3 задача. 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 сортированном массиве число инверсий равно 0, а в массиве, сортированном 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 ($1 \le n \le 10^5$) — 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.

Листинг кода (именно листинг, а не скрины):

```
def inv_count(A, l, q, r):
    n1 = q - l + 1
    n2 = r - q
```

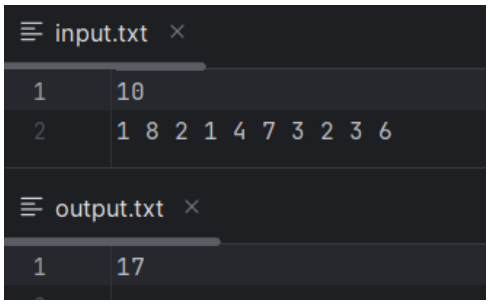
```
    L = [0] * n1
    R = [0] * n2

    for i in range(n1):
        L[i] = A[l + i]
    for j in range(n2):
        R[j] = A[q + 1 + j]

    i = j = 0
    k = l
    icounter = 0

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            icounter += (n1 - i)
            j += 1
        k += 1

    while i < n1:
        A[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        A[k] = R[j]
        j += 1
        k += 1

    return icounter
```

```
def merge_sort(A, AA, l, r):
    icounter = 0
    if l < r:
        q = (l + r) // 2
        icounter += merge_sort(A, AA, l, q)
        icounter += merge_sort(A, AA, q + 1, r)
        icounter += inv_count(A, l, q, r)
    return icounter


def main():
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
                                A    =    list(map(int,
file.readline().strip().split()))

    AA = [0] * n
    icounter = merge_sort(A, AA, 0, n - 1)

    with open('output.txt', 'w') as file:
        file.write(str(icounter) + '\n')


if __name__ == "__main__":
    main()
```

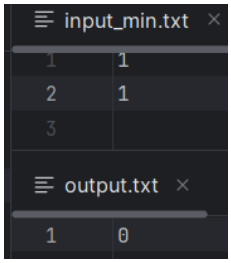
Текстовое объяснение решения:

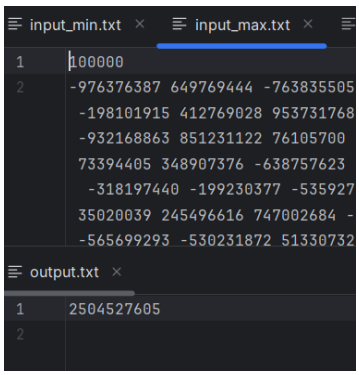
Алгоритм основан на сортировке слиянием. Рекурсивно делим массив на половины, сортируем каждую половину и сливаем в отсортированный. Во время слияния подсчитываем инверсии, когда элем из правой половины массива вставляется перед элементом из левой половины.

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 входных данных из текста задачи	0.0006778 с	15.109375 Мб
Пример из задачи	0.000777 с	15.1796875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.4190819 с	18.11328125 Мб

Вывод по задаче:

Алгоритм для подсчета инверсий основывается на сортировке слиянием, он справляется с кол-вом числе 10**5, тк его сложность O(nlogn)

Задача №4.Бинарный поиск

Текст задачи:

4 задача. 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 $a_0 < a_1 < ... < a_{n-1}$ 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 $b_0, ...b_{k-1}$, $1 \leq b_j \leq 10^9$ 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:

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

В этом примере есть возрастающая последовательность из $a_0 = 1, a_1 = 5, a_2 = 8, a_3 = 12$ и $a_4 = 13$ длиной в $n = 5$ и пять чисел для поиска: 8 1 23 1 11. Видно, что $a_2 = 8$ и $a_0 = 1$, но чисел 23 и 11 нет в последовательности a, поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

Листинг кода (именно листинг, а не скрины):

```
def binary_search(arr, x):
    l, r = 0, len(arr) - 1
    while l <= r:
        mid = (l + r) // 2
```

```
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            l = mid + 1
        else:
            r = mid - 1
    return -1


def main():
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
                                arr    =    list(map(int,
file.readline().strip().split()))
        k = int(file.readline().strip())
                                k_n    =    list(map(int,
file.readline().strip().split()))

    results = []
    for kk in k_n:
        result = binary_search(arr, kk)
        results.append(result)

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


if __name__ == "__main__":
    main()
```

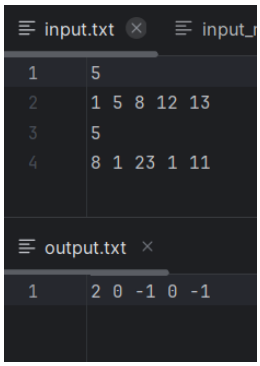
Текстовое объяснение решения:

Определяем начальные границы поиска в массиве, затем пока левая граница не превысит правую

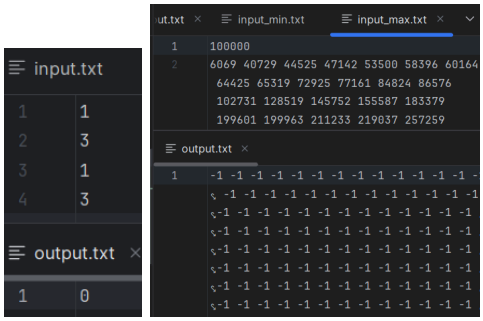
Проверяем середину, если элемент на этой позиции = искомому, возвращаем его индекс, если меньше - продолжаем поиск в правой половине, если больше

-в левой, и так пока не найдём искомый, если найти не получается возвращаем -1.

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 входных данных из текста задачи	0.0005473 с	15.02734375 Мб
Пример из задачи	0.0005793 с	14.796875 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.3419957 с	20.328125 Мб

Вывод по задаче:

Алгоритм бинарного поиска эффективен для большого массива чисел тк его сложность O(logn). Память и время используются эффективно.

Задача №5. Представитель большинства

Текст задачи:

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов $a_1, a_2, ...a_n$, и нужно проверить, содержит ли она элемент, который появляется больше, чем $n/2$ раз. Наивный метод это сделать:

```
Majority(A):
for i from 1 to n:
        current_element = a[i]
        count = 0
        for j from 1 to n:
                if a[j] = current_element:
                        count = count+1
        if count > n/2:
                return a[i]
return "нет элемента большинства"
```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй"для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время $O(n \log n)$.

- **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 положительных целых чисел, по модулю не превосходящих 10^9, $0 \leq a_i \leq 10^9$.
- **Формат выходного файла (output.txt).** Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример 1:

input.txt	output.txt
5 2 3 9 2 2	1

Число "2"встречается больше 5/2 раз.

- Пример 2:

input.txt	output.txt
4 1 2 3 4	0

Нет элемента, встречающегося больше $n/2$ раз.

Листинг кода (именно листинг, а не скрины):

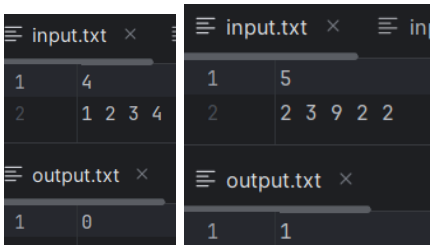
```
def count(A, l, r, x):
    return sum(1 for i in range(l, r + 1) if A[i] == x)


def maj(A, l, r):
    if l == r:
        return A[l]

    mid = (l + r) // 2
    left_maj = maj(A, l, mid)
    right_maj = maj(A, mid + 1, r)

    if left_maj == right_maj:
        return left_maj

    l_count = count(A, l, r, left_maj)
    r_count = count(A, l, r, right_maj)

    return left_maj if l_count > r_count else right_maj


def is_majority(A, elem):
    count = sum(1 for x in A if x == elem)
    return count > len(A) // 2


def main():
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
                                A     =     list(map(int,
file.readline().strip().split()))

    result = 1 if is_majority(A, maj(A, 0, n - 1)) else
0
```

```
    with open('output.txt', 'w') as file:
        file.write(str(result) + '\n')

if __name__ == "__main__":
    main()
```

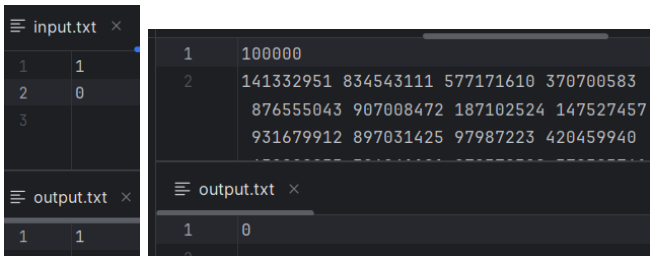
Текстовое объяснение решения:

Считаем кол-во элементов x в массиве A в границах l r. Рекурсивно разделяем массив на 2 части и ищем “Представитель большинства”, возвращаем его. В функции is_maj проверяем действительно ли элемент встречается более n/2 раз, в зависимости от результата печатаем в выходной файл 1 или 0.

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 входных данных из текста задачи	0.0005849 с	14.98046875 Мб
Пример из задачи	0.0008767 с	14.78515625 Мб

Верхняя граница диапазона значений входных данных из текста задачи	0.3264099 с	17.98046875 Мб

Вывод по задаче:

Алгоритм ищет элемент который встречается в массиве более n/2 раз. В этом алгоритме тоже используется принцип разделяй и властвуй) Что позволяет эффективно обрабатывать достаточно большие массивы тк сложность O(nlogn)

Задача №8. Умножение многочленов

Текст задачи:

8 задача. Умножение многочленов

Выдающийся немецкий математик Карл Фридрих Гаусс (1777—1855) заметил, что хотя формула для произведения двух комплексных чисел $(a + bi)(c + di) = ac - bd + (bc + ad)i$ содержит *четыре* умножения вещественных чисел, можно обойтись и *тремя*: вычислим ac, bd и $(a + b)(c + d)$ и воспользуемся тем, что $bc + ad = (a + b)(c + d) - ac - bd$.

Задача. Даны 2 многочлена порядка $n - 1$: $a_{n-1}x^{n-1} + a_{n-2}x^{n-1} + ... + a_1x + a_0$ и $b_{n-1}x^{n-1} + b_{n-2}x^{n-1} + ... + b_1x + b_0$. Нужно получить произведение:

$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + ... + c_1x + c_0$, где:

$$\begin{aligned} c_{2n-2} &= a_{n-1}b_{n-1} \\ c_{2n-3} &= a_{n-1}b_{n-2} + a_{n-2}b_{n-1} \\ ... &\quad ... \\ c_2 &= a_2b_0 + a_1b_1 + a_0b_2 \\ c_1 &= a_1b_0 + a_0b_1 \\ c_0 &= a_0b_0 \end{aligned}$$

Пример. Входные данные: $n = 3,\ A = (3, 2, 5),\ B = (5, 1, 2)$

$$\begin{aligned} A(x) &= 3x^2 + 2x + 5 \\ B(x) &= 5x^2 + x + 2 \\ A(x)B(x) &= 15x^4 + 13x^3 + 33x^2 + 9x + 10 \end{aligned}$$

Ответ: $C = (15, 13, 33, 9, 10)$.

- **Формат входного файла (input.txt).** В первой строке число n - порядок многочленов A и B. Во второй строке коэффициенты многочлена A через пробел. В третьей строке коэффициенты многочлена B через пробел.
- **Формат выходного файла (output.txt).** Ответ - одна строка, коэффициенты многочлена $C(x) = A(x)B(x)$ через пробел.
- Нужно использовать метод "Разделяй и властвуй". Подсказка: любой многочлен $A(x)$ можно разделить на 2 части, например, $A(x) = 4x^3 + 3x^2 + 2x + 1$ разделим на $A_1 = 4x+3$ и $A_2 = 2x+1$. И многочлен $B(x) = x^3+2x^2+3x+4$ разделим на 2 части: $B_1 = x + 2$, $B_2 = 3x + 4$. Тогда произведение $C = A(x) * B(x) = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + A_2B_2$ - требуется 4 произведения (проверьте правильность данной формулы). Можно использовать формулу Гаусса и обойтись всего тремя произведениями.

Листинг кода (именно листинг, а не скрины):

```
def plussing_p(A, B):
    return [a + b for a, b in zip(A, B)]


def subtract_p(A, B):
    return [a - b for a, b in zip(A, B)]


def karatsuba(A, B):
    n = len(A)

    if n == 1:
        return [A[0] * B[0]]

    m = n // 2

    A0, A1 = A[:m], A[m:]
    B0, B1 = B[:m], B[m:]
```

```
    t0, t2 = karatsuba(A0, B0), karatsuba(A1, B1)
    t1 = karatsuba(plussing_p(A0, A1), plussing_p(B0,
B1))
    t1 = subtract_p(subtract_p(t1, t0), t2)

    result = [0] * (2 * n)
    for i in range(len(t0)):
        result[i] += t0[i]
    for i in range(len(t1)):
        result[i + m] += t1[i]
    for i in range(len(t2)):
        result[i + 2 * m] += t2[i]

    return result


def main():
    with open('input.txt', 'r') as file:
        n = int(file.readline().strip())
                                A    =    list(map(int,
file.readline().strip().split()))
                                B    =    list(map(int,
file.readline().strip().split()))

    # доп-е нулями
    while len(A) < len(B):
        A.append(0)
    while len(B) < len(A):
        B.append(0)
    while len(A) & (len(A) - 1) != 0:
        A.append(0)
        B.append(0)

    C = karatsuba(A, B)
```

```
    # удаляем лишние нулей для вывода
    while len(C) > 1 and C[-1] == 0:
        C.pop()

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


if __name__ == "__main__":
    main()
```

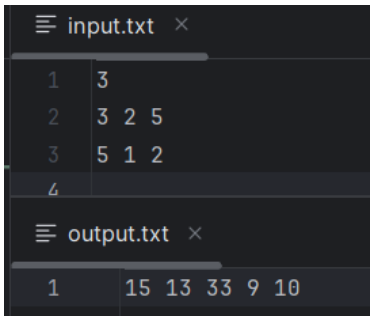
Текстовое объяснение решения:

На ввод примимаем многочленны в виде коэфициентов, при неоходимости добовляем нули. В лучшем случае если для многочлена из коэф. =1 просто возвращаем результат умножения, в остальных случаях далим многочлены A и B на 2 половины A_i и B_i, выделяем нижнюю и верхнюю части. Рекурсивно считаем произведения во вспомогательных переменных t. Формируем итоговый список коэффициентов.

*plussing_p и subtract_p вычисляют попарную сумму и разность коэф-в

В конце для более красивого вывода убираем лишние нили.

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 входных данных из текста задачи	0.0006511 с	14.953125 Мб
Пример из задачи	0.0006956 с	15.0 Мб
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче:

Мне прям понравилось, сам алгоритм интересный, посмотрела видео Андрея Станкевича, чтобы понять формулу Карацубы, рекомендую)
Алгоритм Карацубы позволяет умножить два многочлена или числа длины n за время порядка O(n^log_2(3)) ~ O(n^1.6).

Вывод (по всей лабораторной)

Лабораторная направлена на изучение принципа Разделяй и Властвуй, лежащую в основе алгоритма merge sort. Выполняя задания я несколько раз отсортировала массивы, нашла представителя большинства и узнала интересный алгоритм для перемножения многочленов. Все написанные алгоритмы работают со сложностью O(nlogn) (и последний O(n^log_2(3))), что доказывает их эффективность.