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 6

Выполнил:
Данилова А.В.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. 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 ($1 \le n \le 2 \cdot 104$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109 .

• 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, 104 , 105 чисел порядка 109 , 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

```
def merge(arr, l, m, r):

    n1 = m - l + 1
    n2 = r - m

    L = [0] * (n1)
    R = [0] * (n2)

    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
```

```
    i = 0
    j = 0
    k = l

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

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

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


def mergeSort(arr, l, r):
    if l < r:

        m = l + (r - l) // 2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)

    return arr
```

Функция merge(arr, l, m, r):

Эта функция объединяет два отсортированных подмассива arr[l...m] и arr[m+1...r] в один отсортированный массив. Она работает путем сравнения элементов из двух подмассивов и размещения наименьшего элемента в результирующий массив. Процесс продолжается до тех пор, пока не будут обработаны все элементы обоих подмассивов.

Функция mergeSort(arr, l, r):

Это рекурсивная функция, которая сортирует массив arr с индексами от l до r. Если l < r, значит, массив содержит более одного элемента, и он должен быть разделен на два подмассива. Она вызывается рекурсивно для каждого подмассива (mergeSort(arr, l, m) и mergeSort(arr, m+1, r)), сортируя их независимо. После сортировки подмассивов merge(arr, l, m, r) объединяет их в один отсортированный массив. Если l >= r, значит, подмассив содержит один или ни одного элемента, и он уже отсортирован.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.20740319998003542 секунд	0.00787353515625 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.216592000098899 секунд	0.76318359375 MB

Задача №2. Сортировка слиянием+

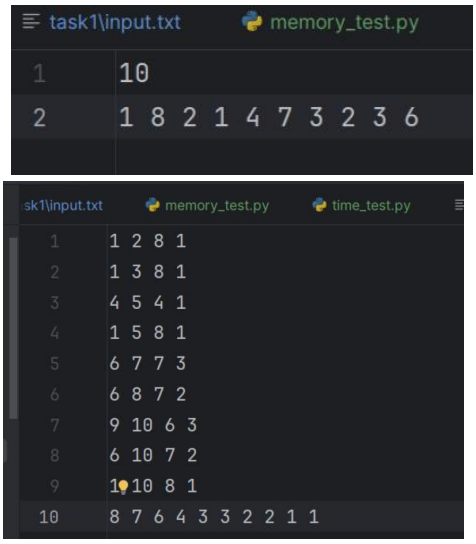
Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием. Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить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

```
def merge(arr, left, m, right):
    n1 = m - left + 1
    n2 = right - m

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

    for i in range(0, n1):
        L[i] = arr[left + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    i = 0
    j = 0
    k = left

    while i < n1 and j < n2:
        if L[i] >= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

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

    while j < n2:
```

```
        arr[k] = R[j]
        j += 1
        k += 1


    with open('output.txt', 'a') as answer:
        answer.write(str(left + 1) + ' ' + str(right + 1) + ' ' +
str(arr[left]) + ' ' + str(arr[right]) + '\n')


def mergeSort(arr, left, right):
    if left < right:
        m = (left + right) // 2

        mergeSort(arr, left, m)
        mergeSort(arr, m + 1, right)
        merge(arr, left, m, right)

        return arr


result = ''
file = open('input.txt', 'r')
n = int(file.readline())
data = [int(elem) for elem in file.readline().split() if abs(int(elem)) <=
10e9]
file.close()
with open('output.txt', 'w') as ans:
    if not (1 <= n <= 10e3 and n == len(data)):
        ans.write('---Incorrect data---')
    else:
        result = ' '.join(map(str, mergeSort(data, 0, n - 1)))
with open('output.txt', 'a') as ans:
    ans.write(result)
```

Поворачивается знак в коде из первой задачи при сравнении элементов из двух подмассивов для последующего размещения наименьшего элемента в результирующий массив и выводятся индексы граничных элементов, места слияния.

Результат работы кода на примерах из текста задачи:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	5.550007335841656e-05 секунд	0.0002193450927734375 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.26120850001461804 секунд	0.76318359375 MB

Вывод по задаче 1, 2: В этих задачах был изучен и реализован метод сортировки слиянием.

Задача №8. Умножение многочленов

Даны 2 многочлена порядка n − 1: an−1x n−1 + an−2x n−1 + ... + a1x + a0 и bn−1x n−1 + bn−2x n−1 + ... + b1x + b0. Нужно получить произведение: 7 c2n−2x 2n−2 + c2n−3x 2n−3 + ... + c1x + c0, где: c2n−2 = an−1bn−1 c2n−3 = an−1bn−2 + an−2bn−1 ... ... c2 = a2b0 + a1b1 + a0b2 c1 = a1b0 + …

```
from math import *


def binary(lenarr, array):
    # добавление нулей для: n = степень двойки
    return [0] * (2 ** ceil(log(lenarr, 2)) - lenarr) + array


def mult2(a, b, n, al, bl):
    r = [0] * (2 * n - 1)
    if n == 1:
        r[0] = a[al] * b[bl]
        return r

    r[0: n - 1] = mult2(a, b, n // 2, al, bl)
    r[n: 2 * n - 1] = mult2(a, b, n // 2, (al + n // 2), (bl + n // 2))

    de = mult2(a, b, n // 2, al, (bl + (n // 2)))
    ed = mult2(a, b, n // 2, (al + (n // 2)), bl)

    k = 0
    for i in range(n // 2, n + n // 2 - 1):
        r[i] += de[k] + ed[k]
        k += 1
        if k >= len(de):
            break
```

```
    return r


def polynomial_mult(n, a, b):
    real_n = n
    a = binary(n, a)
    b = binary(n, b)
    n = len(a)
    al, bl = 0, 0
    answer = mult2(a, b, n, al, bl)

    return answer[len(answer) - 2 * real_n + 1:]



file = open('input.txt', 'r')
poly_ord = int(file.readline())
array_a = [int(elem) for elem in file.readline().split()]
array_b = [int(elem) for elem in file.readline().split()]
file.close()
with open('output.txt', 'w') as ans:
    ans.write('C = ' + '(' + ','.join(map(str, polynomial_mult(poly_ord,
array_a, array_b))) + ')')
```

Функция binary(lenarr, array):

Дополняет входной массив нулями до длины, равной ближайшей степени двойки от длины массива. Используется для подготовки массивов полиномов к обработке алгоритмом FFT.

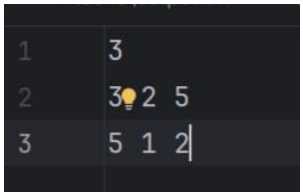
Функция mult2(a, b, n, al, bl):

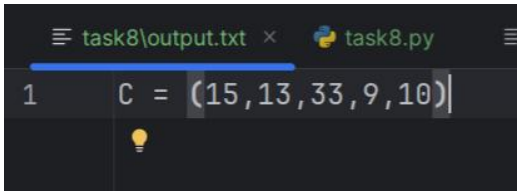
Рекурсивно умножает два полинома, представленных массивами a и b, начиная с индексов al и bl соответственно. Использует алгоритм "разделяй и властвуй" для разбиения полиномов на части и рекурсивного умножения. Выполняет сложение результатов умножения частей полиномов с использованием метода "разделяй и властвуй.

Функция polynomial_mult(n, a, b):

Принимает два полинома a и b степени n и возвращает полином, являющийся их произведением. Дополняет массивы полиномов a и b нулями до ближайшей степени двойки, используя функцию binary. Вызывает функцию mult2 для умножения дополненных массивов. Возвращает результат умножения, отбрасывая ведущие нули.

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи	0.00012079998850822449 секунд	0.00215911865234375 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.42735789995640516 секунд	0.2857818603515625 MB

Вывод по задаче: Узнала интересный факт про разложение полинома и убедилась, что алгоритм «Разделяй и властвуй» эффективно справляется с решением задач, которые можно разделить на меньшие подзадачи.

Дополнительные задачи

Задача №3. Число инверсий.

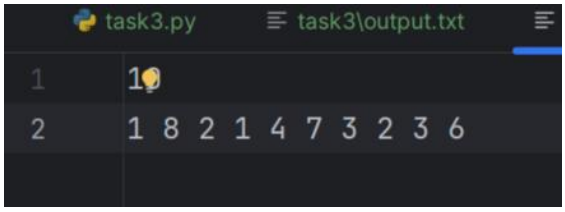
Инверсией в последовательности чисел A называется такая ситуация, когда i < j, а Ai > Aj . Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной. Например, в сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего n(n − 1)/2). Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем

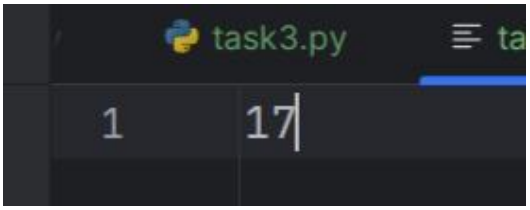
```
def merge(arr, left, m, r):

    global local_var
    n1 = m - left + 1
    n2 = r - m

    L = [0] * n1
    R = [0] * n2
    for i in range(0, n1):
        L[i] = arr[left + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    i = 0
    j = 0
    k = left
    while i < n1 and j < n2:
        if L[i] > R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            local_var += i
```

```
            # print(i, arr)
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        local_var += i
        # print(i, arr)
        j += 1
        k += 1


def mergeSort(arr, left, right):
    if left < right:
        m = (left + right) // 2

        mergeSort(arr, left, m)
        mergeSort(arr, m + 1, right)
        merge(arr, left, m, right)

    return arr


def inversion_number(lenarr, arr):
    global local_var
    local_var = 0
    mergeSort(arr, 0, lenarr - 1)

    return local_var
```

Так же как и раньше реализуется сортировка слиянием, однако во время слияния внутри функции merge подсчитывается число инверсий между элементами из двух подмассивов. Если L[i] > R[j], значит, элементы arr[left + i] и arr[m + 1 + j] образуют инверсию. В этом случае индекс i увеличивается, так как элемент L[i] уже помещен в результирующий массив, а число инверсий увеличивается на количество элементов, уже помещенных из левого подмассива: local_var += i (только левого чтобы не было повторов элементов при подсчёте).

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
Нижняя граница	3.8800062611699104e-	0.0002193450927734375

диапазона значений входных данных из текста задачи	05 секунд	MB
Верхняя граница диапазона значений входных данных из текста задачи	0.30639699986204505 секунд	0.763275146484375 MB

Вывод по задаче: Немного дополнив сортировку слиянием, можно решить разные интересные задачи. Вспомнила про область видимости переменных на языке python.

Задача №4 Бинарный поиск.

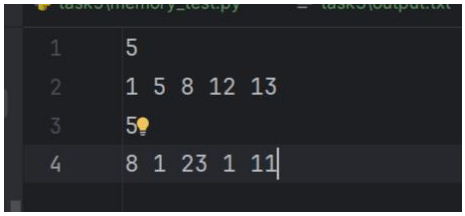
В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска

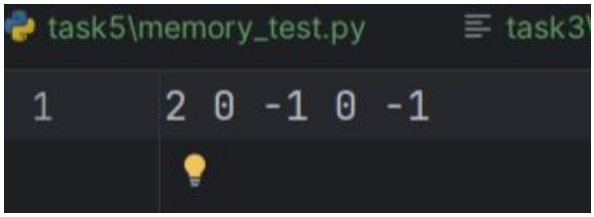
```
def binary_search(len1: int, search_in: list, len2: int, the_search: list):

    answer = [-1] * len2
    for i in range(len2):
        low = 0
        high = len1 - 1

        while low <= high:

            mid = (high + low) // 2

            if search_in[mid] < the_search[i]:
                low = mid + 1
            elif search_in[mid] > the_search[i]:
                high = mid - 1
            else:
                answer[i] = mid
                break

    return answer
```

Создается массив answer размером len2, который будет содержать индексы найденных элементов или -1, если элемент не найден. Для каждого элемента the_search[i] выполняется бинарный поиск в массиве search_in. Бинарный поиск работает следующим образом: находится средний индекс mid. После происходит сравнение search_in[mid] с элементом the_search[i]. Если search_in[mid] меньше the_search[i], то low присваивается значение mid + 1, так как искомый элемент находится правее. Если search_in[mid] больше the_search[i], то high присваивается

mid - 1, так как искомый элемент находится левее. Иначе элемент найден, его индекс записывается в массив.

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.20740319998003542 секунд	0.00787353515625 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.216592000098899 секунд	0.76318359375 MB

Вывод по задаче: Был изучен и реализован алгоритм бинарного поиска в массиве.

Задача №5. Представитель большинства.

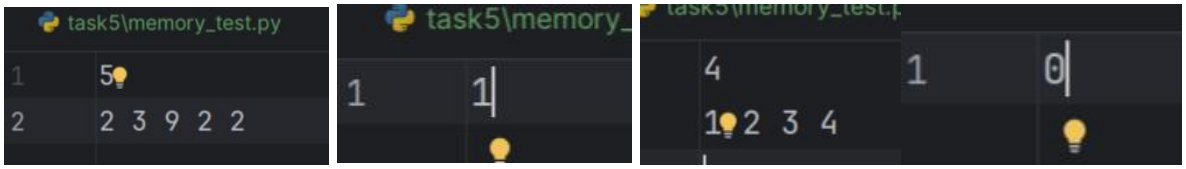
Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a1, a2, ...an, и нужно проверить, содержит ли она элемент, который появляется больше, чем n/2 раз.

```
from lab_2.task1.task1 import mergeSort


def majority(n, array):

    mergeSort(array, 0, n - 1)
    mid = (0 + n) // 2
    first_entry = array.index(array[mid])
    last_entry = n - array[::-1].index(array[mid]) - 1
    if last_entry - first_entry >= mid:
        return 1
    else:
        return 0
```

Полученный массив сортируется слиянием. Я не стала переписывать код, поэтому импортировала функцию из первой задачи. Предположим,

такой элемент существует, тогда в нашем массиве(уже отсортированном!) подряд стоят одинаковые элементы, количество которых превышает или равно n // 2 + 1. Из этого следует, что средний элемент всегда равен искомому элементу. Теперь, узнав значение, можем найти первое и последнее вхождение элемента в массиве. Если их разность больше половины длины массива, то представитель большинства существует.

Результат работы кода на примерах из текста задачи:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	3.1800009310245514e-05 секунд	0.7632408142089844 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.3448820114135742 секунд	0.7632408142089844 MB

Вывод о задаче: Сортировка слиянием довольно быстрая, поэтому удобно, использовать её, для решение задач, в которых необходимо отсортировать список.

Вывод:

В процессе выполнения Лабораторной работы №2 были реализованы сортировка слиянием и бинарный поиск, проверены умение работать с файлами, создание тестов минимальных, максимальных значений и оценка времени работы, затрат памяти при созданных рандомно или очевидно наиболее подходящих(например для 1 задачи) данных.