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 9

Выполнила:

Коновалова Кира Романовна

К3139

Проверил:

Афанасьев А.В

Санкт-Петербург

2024 г.

# Содержание отчета

Оглавление

[Содержание отчета 2](#_Toc182384616)

[Задачи по варианту 3](#_Toc182384617)

[Задача №1. Сортировка слиянием 3](#_Toc182384618)

[Задача №3. Число инверсий 4](#_Toc182384619)

[Задача №4. Бинарный поиск 5](#_Toc182384620)

[Задача №5. Представитель большинства 6](#_Toc182384621)

[Задача №7. Поиск максимального подмассива за линейное время 7](#_Toc182384622)

[Задача №8. Умножение многочленов 8](#_Toc182384623)

# Задачи по варианту

## Задача №1. Сортировка слиянием

Текст задачи:

напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры

Листинг кода:

*'''Сортировка слиянием'''*from lab2.utils import open\_file, write\_file  
  
def merge\_sort(array):  
 if len(array) > 1:  
 mid = len(array) // 2  
 left = array[:mid]  
 right = array[mid:]  
 merge\_sort(left)  
 merge\_sort(right)  
 i = j = k = 0  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 array[k] = left[i]  
 i += 1  
 else:  
 array[k] = right[j]  
 j += 1  
 k += 1  
 while i < len(left):  
 array[k] = left[i]  
 i += 1  
 k += 1  
 while j < len(right):  
 array[k] = right[j]  
 j += 1  
 k += 1  
 return array  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 n\_str, m = open\_file("../txtf/input.txt")  
 n = int(n\_str[0])  
 ans = merge\_sort(m)  
 write\_file(ans,"../txtf/output.txt" )

Текстовое объяснение задачи:

Для решения первой задачи я реализовала рекрсивный алгоритм сортировки вставками. Где я делю исходный список пополам (left и right) и продолжаю делить каждый из этих двух образовавшихся списков до тех пор пока не получу список длиной 1. А дальше сливаю списки в один, сравнивая элемента между собой, так что на выводе выводиться отсортированный список.

Тесты:

import unittest  
from lab2.task1.src.merge\_sort import merge\_sort  
  
class TestMergeSort(unittest.TestCase):  
 def test1(self):  
 self.assertEqual(merge\_sort([12, 11, 13, 5, 6, 7]), [5, 6, 7, 11, 12, 13])  
 def test2(self):  
 self.assertEqual(merge\_sort([1, 2, 3, 4, 5]), [1, 2, 3, 4, 5])  
 def test3(self):  
 self.assertEqual(merge\_sort([8, 7, 6, 5, 4, 3, 2, 1]), [1, 2, 3, 4, 5, 6, 7, 8])  
 def test4(self):  
 self.assertEqual(merge\_sort([3]), [3])  
 def test\_5(self):  
 self.assertEqual(merge\_sort([]), [])  
 def test6(self):  
 array = [1000000000, 999999999, 999999998]  
 self.assertEqual(merge\_sort(array), [999999998, 999999999, 1000000000])  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

Вывод по задаче:

Алгоритм показал себя эффективно для всех протестированных массивов.

## Задача №3. Число инверсий

Текст задачи:

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Листинг кода:

*'''Число инверсий'''*from lab2.utils import open\_file, write\_file  
  
def inv\_count(array, copy\_array, left, middle, right):  
 i = left  
 j = middle + 1  
 k = left  
 cnt = 0  
 while i <= middle and j <= right:  
 if array[i] <= array[j]:  
 copy\_array[k] = array[i]  
 i += 1  
 else:  
 copy\_array[k] = array[j]  
 cnt += (middle - i + 1)  
 j += 1  
 k += 1  
 while i <= middle:  
 copy\_array[k] = array[i]  
 i += 1  
 k += 1  
 while j <= right:  
 copy\_array[k] = array[j]  
 j += 1  
 k += 1  
 for i in range(left, right + 1):  
 array[i] = copy\_array[i]  
 return cnt  
def inversions\_numbers(array, copy\_array, left, right):  
 if len(copy\_array) < len(array):  
 raise ValueError("The Aarray is too small to sort.")  
 cnt = 0  
 if left < right:  
 middle = (left + right) // 2  
 cnt += inversions\_numbers(array, copy\_array, left, middle)  
 cnt += inversions\_numbers(array, copy\_array, middle + 1, right)  
 cnt += inv\_count(array, copy\_array, left, middle, right)  
 return cnt  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 n\_str, massive = open\_file("../txtf/input.txt")  
 n = int(n\_str[0])  
 massive\_copy = massive.copy()  
 ans = inversions\_numbers(massive, massive\_copy, 0, n-1)  
 write\_file(ans, "../txtf/output.txt")

Текстовое объяснение задачи:

Данный алгоритм это модифицированный для подсчета инфверсий алгоритм сортировки слиянием. Первая функция inv\_count. Вторая функция такой же алгоритм сортировки слиянием как из задачи 1, который записывает число инверсий если они есть.

Тесты:

import unittest  
from lab2.task3.src.inversions\_numbers import inversions\_numbers  
  
  
  
class TestInversionsNumbers(unittest.TestCase):  
  
 def test\_inversions(self):  
 array = [1, 8, 2, 1, 4, 7, 3, 2, 3, 6]  
 self.assertEqual(inversions\_numbers(array, [0] \* len(array), 0, len(array) -1 ), 17)  
 self.assertEqual(inversions\_numbers([1, 2, 3, 4, 5], [1, 2, 3, 4, 5], 0, 4), 0)  
 self.assertEqual(inversions\_numbers([5, 4, 3, 2, 1], [5, 4, 3, 2, 1], 0, len([5, 4, 3, 2, 1])-1), 10)  
  
 def test\_empty\_array(self):  
 self.assertEqual(inversions\_numbers([], [], 0, len([])-1 ), 0)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

## Задача №4. Бинарный поиск

Текст задачи:

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.

Листинг кода:

*'''Бинарный поиск'''*from lab2.utils import open\_file, write\_file  
def binary\_search(arr, target):  
 left, right = 0, len(arr) - 1  
 while left <= right:  
 mid = (left + right) // 2  
 if arr[mid] == target:  
 return mid  
 elif arr[mid] < target:  
 left = mid + 1  
 else:  
 right = mid - 1  
 return -1  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 data\_n, data\_k = open\_file("../txtf/input.txt")  
 n = int(data\_n[0])  
 mass = data\_n[1:]  
 k = int(data\_k[0])  
 mass\_f = data\_k[1:]  
 write\_file("", "../txtf/output.txt", mode="w")  
 for i in mass\_f:  
 ans = binary\_search(mass, i)  
 write\_file(f"{ans} ", "../txtf/output.txt", mode='a')

Текстовое объяснение задачи:

Внутри функция также ввояться две переменные – левая и правая граница. На каждом шаге массив делиться на две части и в работе остается только та часть, где находится искомый объект.

Тесты:

import unittest  
from lab2.task4.src.bin\_search import binary\_search  
  
class TestBinarySearch(unittest.TestCase):  
 def test\_binarysearch1(self):  
 self.assertEqual(binary\_search([1, 2, 3, 4, 5], 3), 2)  
 self.assertEqual(binary\_search([1, 2, 3, 4, 5], 8), -1)  
  
 def test\_binarysearch2(self):  
 self.assertEqual(binary\_search([], 1), -1)

|  |
| --- |
|  |

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

## Задача №5. Представитель большинства

Текст задачи:

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)

Листинг кода:

from lab2.utils import open\_file, write\_file  
  
def majority(a):  
 dic = {}  
 for i in a:  
 dic[i] = dic.get(i, 0) + 1  
  
 for l, m in dic.items():  
 if m > len(a) // 2:  
 return 1  
 return 0  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 n\_str, m = open\_file("../txtf/input.txt")  
 n = int(n\_str[0])  
 ans = majority(m)  
 write\_file(ans, "../txtf/output.txt")

Текстовое объяснение задачи:

В данной задаче мною реализован алгоритм для нахождения элемента, который встречается больше чем n//2 раз. Для этого я использую словарь, прохожу по его элементам и если какое-то значение встречается больше, чем n//2 раз, то вывожу 1, иначе 0

Тесты:

import unittest  
from lab2.task5.src.majority\_element import majority  
  
class TestMajorityElement(unittest.TestCase):  
  
 def test\_majority1(self):  
 self.assertEqual(majority([2, 3, 9, 2, 2]), 1)  
 self.assertEqual(majority([1, 2, 3, 4]), 0)  
  
 def test\_majority2(self):  
 self.assertEqual(majority([1]), 1)

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

## Задача №7. Поиск максимального подмассива за линейное время

Текст задачи:

Найти максимальный подмассив за линейное время

Листинг кода:

*'''Поиск максимального подмассива'''*from lab2.utils import open\_file, write\_file  
  
  
def max\_subarray(array):  
 mx\_summ = 0  
 summ = 0  
 for i in range(len(array)):  
 if summ == 0:  
 start = i  
 summ += array[i]  
 if mx\_summ < summ:  
 mx\_summ = summ  
 finish = i  
 if summ < 0:  
 summ = 0  
 return array[start:finish + 1]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 n\_str, m = open\_file("../txtf/input.txt")  
 n = int(n\_str[0])  
  
 ans = max\_subarray(m)  
 write\_file(ans, "../txtf/output.txt")

Текстовое объяснение задачи:

Данный алгоритм ищет максимальный подмассив исходного массива. Для его реализации я ввожу два счетчика – максимальную сумму и текущую. Далее я прохожзу по исходному массиву и прибавляю к текущей сумме элемент. Если сумма меньше нуля, то я обнуляю текущую сумму. Если текущая сумма начинает превышать максимальную, то я записываю в нее текущую сумму.

Тесты:

import unittest  
from lab2.task7.src.mx\_subarray import max\_subarray  
  
class TestMaxSubarray(unittest.TestCase):  
 def test1(self):  
 self.assertEqual(max\_subarray([1, 2, 3, 4, 5, 6]), [1, 2, 3, 4, 5, 6])  
 self.assertEqual(max\_subarray([-1, 2, 3, -9]), [2, 3])  
  
 def test\_one\_element(self):  
 self.assertEqual(max\_subarray([-1, 2, 3, -9, 11]), [11])  
 self.assertEqual(max\_subarray([100, -9, 2, -3, 5]), [100])  
  
 def test\_single\_array(self):  
 self.assertEqual(max\_subarray([6]), [6])  
 self.assertEqual(max\_subarray([1]), [1])

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

## Задача №8. Умножение многочленов

Текст задачи:

Реализовать алгоритм умножения двух многочленов.

Листинг кода:

from lab2.utils import open\_file, write\_file  
  
def multiplication\_polynomials(a, b, n):  
 result = [0] \* (n + n - 1)  
 for cof\_a in range(n):  
 for cof\_b in range(n):  
 result[cof\_a + cof\_b] += a[cof\_b] \* b[cof\_a]  
 return result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 a, b = open\_file("../txtf/input.txt")  
 n = a[0]  
 a = a[1:]  
  
 ans = multiplication\_polynomials(a, b, n)  
 write\_file(ans, "../txtf/output.txt")

Тесты:

import unittest  
from lab2.task8.src.polynom import multiplication\_polynomials  
  
class MyltiplicationPolynomialsTest(unittest.TestCase):  
 def test1(self):  
 self.assertEqual(multiplication\_polynomials([8, 9, 10], [10, 12, 2], 3), [80, 186, 224, 138, 20])  
 self.assertEqual(multiplication\_polynomials([3, 9, 5], [5, 12, 2], 3), [15, 81, 139, 78, 10])  
  
 def test\_negative\_numbers(self):  
 self.assertEqual(multiplication\_polynomials([-1, -9, -100], [-1, 12, 2], 3), [1, -3, -10, -1218, -200])  
 self.assertEqual(multiplication\_polynomials([-1, -2, -5], [-1, -4, 2], 3), [1, 6, 11, 16, -10])  
  
 def test\_fifth\_degree(self):  
 self.assertEqual(multiplication\_polynomials([1, -4, -3, -2, -1, -1], [5, 0, 0, -4, 0, 0], 6), [5, -20, -15, -14, 11, 7, 8, 4, 4, 0, 0])

Результат работы кода на примерах из текста задачи:

Результат работы кода на максимальных и минимальных значениях:

# Вывод

В данной лабораторной работе я научилась писать различный алгоритмы, такие как: сортировка слиянием, бинарный поиск, поиск максимального подмассива, число инверсий, представитель большинства, умножение многочленов. А также протестировала данные алгоритмы.