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 8

Выполнила:
Иванова Аайа Гаврильевна
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задание №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.

1.

Листинг кода:

```
def merge(array, p, q, r):
    left_array = array[p:q + 1] + [float('inf')]
    right_array = array[q + 1:r + 1] +
[float('inf')]
    i, j = 0, 0
    for k in range(p, r + 1):
        if left_array[i] <= right_array[j]:
            array[k] = left_array[i]
            i += 1
```

```
        else:
            array[k] = right_array[j]
            j += 1


def merge_sort(array, p, r):
    if p >= r:
        return
    q = (p + r) // 2
    merge_sort(array, p, q)
    merge_sort(array, q + 1, r)
    merge(array, p, q, r)



if __name__ == '__main__':
    with open('input.txt') as f:
        n, array1 = f.readlines()
    array = list(map(int, array1.split()))
    merge_sort(array, 0, len(array) - 1)
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))),
file=f)
```

Текстовое объяснение задачи:
В результате выполнения функции два отсортированных подмассива объединяются так, что получается новый отсортированный массив. Рекурсивные вызовы продолжаются пока left_array меньше right_array. Когда это условие не выполняется, массив считается отсортированным и возвращается в исходное состояние.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.pat
h.dirname(__file__), '..', 'src')))

from merge_sort import merge_sort
```

```
class TestMergeSort(unittest.TestCase):

    def test_basic_sort(self):
        array = [2, 11, 4, 2, 100, 1]
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [1, 2, 2, 4, 11,
100])

    def test_already_sorted(self):
        array = [1, 2, 3, 4, 5, 6, 7, 8]
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [1, 2, 3, 4, 5, 6,
7, 8])

    def test_reverse_sorted(self):
        array = [8, 7, 6, 5, 4, 3, 2, 1]
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [1, 2, 3, 4, 5, 6,
7, 8])

    def test_single_element(self):
        array = [100]
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [100])

    def test_empty_array(self):
        array = []
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [])

    def test_large_numbers(self):
        array = [1000000000, 999999999, 999999998]
        merge_sort(array, 0, len(array) - 1)
        self.assertEqual(array, [999999998,
999999999, 1000000000])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку слиянием двух массивов.

Задание №3 Число инверсий

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 ($1 \leq n \leq 10^5$) — 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.

Листинг кода:

```
def merge_and_count(array, temp_array, p, q,
right):
    i = p
    j = q + 1
    k = p
    counter = 0
    while i <= q and j <= right:
        if array[i] <= array[j]:
            temp_array[k] = array[i]
            i += 1
        else:
            temp_array[k] = array[j]
            counter += (q - i + 1)
            j += 1
        k += 1
    while i <= q:
```

```
        temp_array[k] = array[i]
        i += 1
        k += 1
    while j <= right:
        temp_array[k] = array[j]
        j += 1
        k += 1
    for i in range(p, right + 1):
        array[i] = temp_array[i]
    return counter


def merge_sort_and_count(array, temp_array, p,
right):
    counter = 0
    if p < right:
        middle = (p + right) // 2
        counter += merge_sort_and_count(array,
temp_array, p, middle)
        counter += merge_sort_and_count(array,
temp_array, middle + 1, right)
        counter += merge_and_count(array,
temp_array, p, middle, right)
    return counter


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array1 = f.readlines()
    array = list(map(int, array1.split()))
    temp_array = [0] * len(array)
    inv_count = merge_sort_and_count(array,
temp_array, 0, len(array) - 1)
    with open('output.txt', 'w') as f:
        print(inv_count, file=f)
```

Текстовое объяснение задачи:
Первая функция объединяет два отсортированных подмассива (от p до q и от q+1 до right) в один отсортированный массив и одновременно подсчитывает количество инверсий между этими подмассивами. В процессе слияния, если элемент из второго подмассива меньше элемента из первого, это означает, что все оставшиеся элементы в первом

подмассиве также больше, и количество таких инверсий добавляется к счётчику. Вторая функция возвращает общее количество инверсий в массиве, суммируя инверсии из обеих половин и инверсии, найденные при слиянии.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.pat
h.dirname(__file__), '..', 'src')))

from inv import merge_sort_and_count
class TestInversionsCount(unittest.TestCase):

    def test_inversions(self):
        array = [10, 20, 1, 4, 5]
        temp_array = [0] * len(array)
        result = merge_sort_and_count(array,
temp_array, 0, len(array) - 1)
        self.assertEqual(result, 6)

    def test_no_inversions(self):
        array = [1, 2, 3, 4, 5]
        temp_array = [0] * len(array)
        result = merge_sort_and_count(array,
temp_array, 0, len(array) - 1)
        self.assertEqual(result, 0)

    def test_reverse_sorted(self):
        array = [5, 4, 3, 2, 1]
        temp_array = [0] * len(array)
        result = merge_sort_and_count(array,
temp_array, 0, len(array) - 1)
        self.assertEqual(result, 10)

    def test_empty_array(self):
        array = []
        temp_array = []
        result = merge_sort_and_count(array,
temp_array, 0, len(array) - 1)
```

```
        self.assertEqual(result, 0)



if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы научились осуществлять поиск количества инверсий в массиве целых чисел.

Задание №4. Бинарный поиск

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:

```
def binary_search(array, target):
    p = 0
    r = len(array) - 1

    while p <= r:
        q = (p + r) // 2
        if array[q] == target:
            return q
        elif array[q] < target:
            p = q + 1
        else:
            r = q - 1
    return -1


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array1, target = f.readlines()
    array = list(map(int, array1.split()))
    result = binary_search(array,
int(target.strip()))
    with open('output.txt', 'w') as f:
        print(result, file=f)
```

Текстовое объяснение задачи:
Функция binary_search реализует алгоритм бинарного поиска, который используется для нахождения индекса заданного элемента (цели) в отсортированном массиве. Пока p меньше или равен r, выполняется следующе – вычисляется средний индекс q как (p + r) // 2. Если элемент в позиции q равен целевому значению (target), функция возвращает индекс q. Если элемент в позиции q меньше целевого значения, это означает, что целевое значение находится в правой половине массива, и p обновляется до q + 1. Если элемент в позиции q больше целевого значения, это означает, что целевое значение находится в левой половине массива, и r обновляется до q - 1. Возврат результата: Если целевое значение не найдено, функция возвращает -1.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.pat
h.dirname(__file__), '..', 'src')))

from search import binary_search
class TestBinarySearch(unittest.TestCase):

    def test_binary_search_found(self):
        array = [1, 2, 3, 4, 5]
        self.assertEqual(binary_search(array, 3), 2)

    def test_binary_search_not_found(self):
        array = [1, 2, 3, 4, 5]
        self.assertEqual(binary_search(array, 6),
-1)

    def test_empty_array(self):
        array = []
        self.assertEqual(binary_search(array, 1),
-1)

    def test_single_element_found(self):
        array = [55]
        self.assertEqual(binary_search(array, 55),
0)

    def test_single_element_not_found(self):
        array = [55]
        self.assertEqual(binary_search(array, 56),
-1)


if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы научились осуществлять бинарный поиск

Дополнительные задачи

Задание №2. Сортировка вставкой +

2 задача. 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 ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** 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, $1 \le I_f \le I_l \le n$). **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.

Листинг кода:

```
def merge_with_indices(array, p, q, r):
    left_array = array[p:q + 1] + [float('inf')]
    right_array = array[q + 1:r + 1] +
[float('inf')]
```

```
    i, j = 0, 0
    print(f"Индексы {p + 1}-{r + 1}, значения:
{array[p]}-{array[r]}")
    for k in range(p, r + 1):
        if left_array[i] <= right_array[j]:
            array[k] = left_array[i]
            i += 1
        else:
            array[k] = right_array[j]
            j += 1


def merge_sort_with_indices(array, p, r):
    if p >= r:
        return

    middle = (p + r) // 2
    merge_sort_with_indices(array, p, middle)
    merge_sort_with_indices(array, middle + 1, r)
    merge_with_indices(array, p, middle, r)


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array1 = f.readlines()
    array = list(map(int, array1.split()))
    merge_sort_with_indices(array, 0, len(array) -
1)
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array))),
file=f)
```

Текстовое объяснение задачи:
Создаются два вспомогательных массива: left_array, который содержит элементы от array[p] до array[q], и right_array, который содержит элементы от array[q + 1] до array[r]. Для каждого индекса k в диапазоне от p до r: Сравниваются текущие элементы из left_array[i] и right_array[j]. Наименьший элемент помещается в array[k], и соответствующий указатель (i или j) увеличивается. Вторая функция выполняет сортировку массива с помощью рекурсивного деления и последующего слияния. Если границы p и r указывают на одну и ту же позицию, значит, массив длиной 1 уже

отсортирован, и функция возвращается. Находится средний индекс middle как (p + r) // 2. Функция рекурсивно вызывает саму себя для левой половины массива (p до middle) и правой половины (middle + 1 до r). После сортировки обеих половин вызывается функция merge_with_indices, чтобы объединить отсортированные подмассивы в один отсортированный массив.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.pat
h.dirname(__file__), '..', 'src')))

from merge_sort_i import merge_sort_with_indices
class TestMergeSort(unittest.TestCase):

    def test_basic_sort(self):
        array = [2, 11, 4, 2, 100, 1]
        merge_sort_with_indices(array, 0, len(array)
- 1)
        self.assertEqual(array, [1, 2, 2, 4, 11,
100])

    def test_already_sorted(self):
        array = [1, 2, 3, 4, 5, 6, 7, 8]
        merge_sort_with_indices(array, 0, len(array)
- 1)
        self.assertEqual(array, [1, 2, 3, 4, 5, 6,
7, 8])

    def test_reverse_sorted(self):
        array = [8, 7, 6, 5, 4, 3, 2, 1]
        merge_sort_with_indices(array, 0, len(array)
- 1)
        self.assertEqual(array, [1, 2, 3, 4, 5, 6,
7, 8])

    def test_single_element(self):
        array = [100]
        merge_sort_with_indices(array, 0, len(array)
```

```
- 1)
        self.assertEqual(array, [100])

    def test_empty_array(self):
        array = []
        merge_sort_with_indices(array, 0, len(array)
- 1)
        self.assertEqual(array, [])

    def test_large_numbers(self):
        array = [1000000000, 999999999, 999999998]
        merge_sort_with_indices(array, 0, len(array)
- 1)
        self.assertEqual(array, [999999998,
999999999, 1000000000])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы научились осуществлять сортировку слиянием с выводом индексов элементов массива целых чисел.

Задание №5. Представитель большинства

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.

Листинг кода:

```python
def majority_element(array):
    tmp = None
    counter = 0
    for i in array:
        if counter == 0:
            tmp = i
        counter += 1 if i == tmp else -1
    if array.count(tmp) > len(array) // 2:
```

```
        return 1
    return None


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array1 = f.readlines()
    array = list(map(int, array1.split()))
    result = majority_element(array)
    with open('output.txt', 'w') as f:
        print(result if result else "Нет элемента
большинства", file=f)
```

Текстовое объяснение задачи:
Обход массива:
Для каждого элемента i в массиве выполняется следующее: Если счетчик counter равен нулю, это означает, что текущий кандидат (tmp) больше не имеет поддержки, и мы выбираем i как нового кандидата, присваивая его переменной tmp. Затем counter увеличивается на 1, если текущий элемент равен кандидату tmp, и уменьшается на 1, если они не равны.
Проверка кандидата:
После завершения прохода по массиву, функция проверяет, является ли выбранный кандидат (tmp) действительным элементом большинства, проверяя, встречается ли он более чем в половине массива с помощью array.count(tmp) > len(array)//2. Если да, функция возвращает 1, указывая на то, что найден элемент большинства. Если нет, возвращается None.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.pat
h.dirname(__file__), '..', 'src')))

from m_el import majority_element
class TestMajorityElement(unittest.TestCase):

    def test_majority_element_exists(self):
        array = [2, 3, 9, 2, 2]
```

```
        self.assertEqual(majority_element(array), 1)

    def test_no_majority_element(self):
        array = [1, 2, 3, 4]
        self.assertIsNone(majority_element(array))

    def test_single_element(self):
        array = [1]
        self.assertEqual(majority_element(array), 1)

    def test_empty_array(self):
        array = []
        self.assertIsNone(majority_element(array))

    def test_majority_element_boundary(self):
        array = [1, 1, 2, 1, 2, 1]
        self.assertEqual(majority_element(array), 1)



if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять поиск мажоритарного элемента среди массива целых чисел.

Задание №7. Поиск максимального подмассива за линейное время

7 задача. 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 рандомного 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**.

Листинг кода:

```
from lab2.task7.utils import read, write

def find_max_subarray(n, array):
    max_sum = 0
    left = 0
    right = 0
    current_sum = 0
    for i in range(n):
        if current_sum == 0:
            left = i
        current_sum += array[i]
        if current_sum < 0:
            current_sum = 0
        if current_sum > max_sum:
            max_sum = current_sum
            right = i
    return [max_sum, [left, right]]


def main():
    write(end='')
    (n,), array = read(type_convert=int)
    write(*find_max_subarray(n, array), to_end=True)
```

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

Текстовое объяснение задачи:
Для каждого индекса i в диапазоне от 0 до n-1 происходит следующее: Если current_sum равен 0, это означает, что текущая сумма не имеет положительного значения, и мы можем начать новый подмассив. Индекс left обновляется на текущее значение i. К current_sum добавляется значение элемента массива array[i]. Если current_sum становится отрицательным, это означает, что подмассив не может быть продолжен, и мы сбрасываем current_sum до 0. Если current_sum больше, чем max_sum, обновляем значение max_sum и правую границу right на текущий индекс i. Функция возвращает максимальную сумму max_sum и список с индексами left и right, которые представляют начало и конец подмассива с максимальной суммой.

Листинг тестов:

```
import unittest
from lab2.task7.utils import memory_data, time_data
from lab2.task7.src.m_suberray import
find_max_subarray, main


class TestFindMaxSubarray(unittest.TestCase):

    def
test_should_find_max_subarray_example_array(self):
        # given
        n = 4
        array = [1, 8, 2, 10]
        expected_result = [21, [0, 3]]

        # when
        result = find_max_subarray(n, array)

        # then
        self.assertEqual(result, expected_result)
```

```
    def
test_should_find_max_subarray_single_element_array(
self):
        # given
        n = 1
        array = [1]
        expected_result = [1, [0, 0]]

        # when
        result = find_max_subarray(n, array)

        # then
        self.assertEqual(result, expected_result)

    def
test_should_find_max_subarray_empty_array(self):
        # given
        n = 0
        array = []
        expected_result = [0, [0, 0]]

        # when
        result = find_max_subarray(n, array)

        # then
        self.assertEqual(result, expected_result)

    def test_should_check_time_data(self):
        # given
        expected_time = 2

        # when
        time = time_data(main)

        # then
        self.assertLess(time, expected_time)

    def test_should_check_memory_data(self):
        # given
        expected_memory = 256

        # when
```

```
        current, peak = memory_data(main)

        # then
        self.assertLess(current, expected_memory)
        self.assertLess(peak, expected_memory)


if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять поиск максимального подмассива массива целых чисел за линейное время.

Вывод

В ходе выполнения лабораторной работы №2 мы изучили сортировку слиянием, сортировку слиянием с выводом индексов элементов, а также научились осуществлять бинарный поиск. поиск максимального подмассива за линейное время. поиск мажоритарного элемента массива и вычисление количества инверсий в массиве.