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:
Левчук С.А.
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 $q$.

Решение:

```
def merge(array, left, middle, right):
    l_array = array[left:middle+1] + [float('inf')]
    r_array = array[middle+1:right+1] + [float('inf')]
    i = j = 0
    for k in range(left, right+1):
        if l_array[i] <= r_array[j]:
            array[k] = l_array[i]
            i += 1
        else:
            array[k] = r_array[j]
            j += 1


def merge_sort(array, left, right):
    if left < right:
        middle = (left + right) // 2
        merge_sort(array, left, middle)
```

```
        merge_sort(array, middle+1, right)
        merge(array, left, middle, right)
        return array
```

```
from lab2.task1.src.merge_sort import merge_sort

with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 2 * 10**4:
    with open('output.txt', 'w') as file:
                file.write('  '.join(map(str,  merge_sort(numbers,  0,
len(numbers)-1))))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Сначала опишем основные функции - merge и merge_sort.

   1) Функция merge создает два подмассива l_array и r_array из изначального массива array: l_array содержит элементы от left до middle включительно и добавляет бесконечность в конец, а r_array содержит элементы от middle + 1 до right включительно и также добавляет бесконечность в конец.

   Затем происходит слияние двух подмассивов l_array и r_array в изначальный массив array. С помощью двух индексов i и j (начиная с 0) функция проходит по элементам подмассивов l_array и r_array соответственно.

   Если элемент из l_array меньше или равен элементу из r_array, то данный элемент помещается в массив array, и индекс i увеличивается. В противном случае элемент из r_array помещается в массив array, и индекс j увеличивается.

   Таким образом, в результате выполнения этой функции два отсортированных подмассива объединяются так, что получается новый отсортированный массив.

   2) Функция merge_sort принимает массив array, индексы left и right (левую и правую границы массива). На каждом шаге функция находит середину массива (middle), после чего вызывает саму себя для левой и правой половин массива. После рекурсивных вызовов выполняется функция merge, которая объединяет две отсортированные половины массива в один отсортированный массив.

Рекурсивные вызовы продолжаются пока left меньше right. Когда это условие не выполняется, массив считается отсортированным и возвращается в исходное состояние.

2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:

Входные данные:

```
10
1 8 2 1 4 7 3 2 3 6
```

Выходные данные:

```
1 1 2 2 3 3 4 6 7 8
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab2.task1.src.merge_sort import merge_sort

test_nums1 = [1, 8, 2, 1, 4, 7, 3, 2, 3, 6]

start_time1 = time.perf_counter()
test1 = merge_sort(test_nums1, 0, len(test_nums1)-1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:

Время: 4.899999998997373e-05 секунд

Память: 14.39453125 Мбайт

2. Минимальное значение

```
import time
import random
import psutil
from lab2.task1.src.merge_sort import merge_sort

test_nums2   =   sorted(random.randint(10**5,   10**9+1)   for   _   in
range(1, 10**5+1))

start_time2 = time.perf_counter()
```

```
test2 = merge_sort(test_nums2, 0, len(test_nums2)-1)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 0.2403553000000329 секунд
Память: 18.2265625 Мбайт

3. Максимальное значение

```
import time
import psutil
import random
from lab2.task1.src.merge_sort import merge_sort

test_nums3   =   sorted([random.randint(10**5,   10**9+1)   for   _   in
range(1, 10**5+1)], reverse=True)

start_time3 = time.perf_counter()
test3 = merge_sort(test_nums3, 0, len(test_nums3)-1)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 0.24453510000000733 секунд
Память: 18.08984375 Мбайт

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять сортировку слиянием массива целых чисел и выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях.

## 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 $n$ ($1 \le n \le 10^5$) — 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.

- Пример:

| input.txt | output.txt |
|---|---|
| 10<br>1 8 2 1 4 7 3 2 3 6 | 17 |

Решение:

```
def inversions_merge(array, temp_array, left, middle, right):
    i = left
    j = middle + 1
    k = left
    inversions_count = 0
    while i <= middle and j <= right:
        if array[i] <= array[j]:
            temp_array[k] = array[i]
            i += 1
        else:
            temp_array[k] = array[j]
            inversions_count += (middle - i + 1)
            j += 1
        k += 1
    while i <= middle:
        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(left, right + 1):
        array[i] = temp_array[i]

    return inversions_count


def inversions_merge_sort(array, temp_array, left, right):
    inversions_count = 0
    if left < right:
        middle = (left + right) // 2
                inversions_count  +=  inversions_merge_sort(array,
temp_array, left, middle)
                inversions_count  +=  inversions_merge_sort(array,
temp_array, middle+1, right)
          inversions_count += inversions_merge(array, temp_array,
left, middle, right)
    return inversions_count
```

```
from lab2.task3.src.inversions_count import inversions_merge_sort

with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 10**5:
    with open('output.txt', 'w') as file:
                        file.write(str(inversions_merge_sort(numbers,
[0]*len(numbers), 0, len(numbers)-1)))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Сначала опишем основные функции - merge и merge_sort.
   1) Функция inversions_merge принимает массив array, временный массив temp_array, и индексы left, middle и right, которые указывают на различные части массива.
   Внутри функции происходит слияние двух сортированных подмассивов (от left до middle и от middle+1 до right) в один сортированный массив temp_array. При этом считается количество инверсий, т.е. пар элементов (i, j), где i < j, но array[i] > array[j]. Количество инверсий увеличивается на (middle - i + 1), когда элемент array[j] меньше элемента array[i].
   После слияния подмассивов и переноса их в temp_array, они копируются обратно в массив array.
   Функция возвращает общее количество инверсий в массиве.

2) Функция inversions_merge_sort принимает массив array, временный массив temp_array, индексы левой (left) и правой (right) границ сортируемого подмассива.
Сначала проверяется условие left < right, при выполнении которого определяется середина массива middle. Затем рекурсивно вызываются функции inversions_merge_sort для сортировки и подсчета инверсий в левой и правой половинах массива, а также inversions_merge для объединения их в отсортированный массив и подсчета инверсий между ними.
Функция inversions_merge_sort возвращает общее количество найденных инверсий в массиве.

2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:
Входные данные:

```
10
1 8 2 1 4 7 3 2 3 6
```

Выходные данные:

```
17
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab2.task3.src.inversions_count import inversions_merge_sort

test_nums1 = [1, 8, 2, 1, 4, 7, 3, 2, 3, 6]

start_time1 = time.perf_counter()
test1 = inversions_merge_sort(test_nums1, [0]*len(test_nums1), 0,
len(test_nums1)-1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:  {psutil.Process().memory_info().rss  /  1024**2}
Мбайт')
```

Результат:
Время: 2.27999998969608e-05 секунд

Память: 14.59765625 Мбайт

2. Минимальное значение

```
import time
import psutil
from lab2.task3.src.inversions_count import inversions_merge_sort

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = inversions_merge_sort(test_nums2, [0]*len(test_nums2), 0,
len(test_nums2)-1)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память:  {psutil.Process().memory_info().rss  /  1024**2}
Мбайт')
```

Результат:
Время: 2.2999993234407157e-06 секунд
Память: 14.453125 Мбайт

3. Максимальное значение

```
import time
import psutil
import random
from lab2.task3.src.inversions_count import inversions_merge_sort

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**5+1)]

start_time3 = time.perf_counter()
test3 = inversions_merge_sort(test_nums3, [0]*len(test_nums3), 0,
len(test_nums3)-1)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память:  {psutil.Process().memory_info().rss  /  1024**2}
Мбайт')
```

Результат:
Время: 0.2747676999997566 секунд
Память: 18.33984375 Мбайт

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять поиск количества инверсий в массиве целых чисел и выяснили, каково время его выполнения и затраты памяти на предельных и средних значениях.

## 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:

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<br>2 3 9 2 2 | 1 |

Число "2"встречается больше 5/2 раз.

- Пример 2:

| input.txt | output.txt |
|---|---|
| 4<br>1 2 3 4 | 0 |

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

Решение:

```
def majority_element(array):
    def find_candidate(array):
        count = 0
        candidate = None
        for number in array:
            if count == 0:
                candidate = number
                count = 1
            elif number == candidate:
                count += 1
            else:
                count -= 1

        return candidate
```

```
    def is_majority(array, candidate):
        count = sum(1 for number in array if number == candidate)
        return count > len(array) // 2

    candidate = find_candidate(array)

    if is_majority(array, candidate):
        return 1
    else:
        return 0
```

1. Внутри функции majority_element определены две вспомогательные функции:
   1) find_candidate ищет кандидата на мажоритарный элемент в массиве. Она проходит по всем элементам массива, увеличивая счетчик, если текущий элемент совпадает с кандидатом, и уменьшает счетчик в остальных случаях. Кандидатом становится элемент с наибольшей частотой встречаемости.
   2) is_majority проверяет, является ли кандидат мажоритарным элементом массива. Она считает количество вхождений кандидата в массив и возвращает True, если его частота больше половины длины массива.
   3) В основной части кода вызывается find_candidate для поиска кандидата на мажоритарный элемент, затем проверяется с помощью is_majority, является ли этот кандидат действительно мажоритарным элементом.
   Если мажоритарный элемент найден, функция majority_element возвращает 1, в противном случае возвращает 0.
2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:

1) Входные данные:

```
5
2 3 9 2 2
```

Выходные данные:

```
1
```

2) Входные данные:

```
4
1 2 3 4
```

Выходные данные:

```
0
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab2.task5.src.majority_element import majority_element

test_nums1 = [2, 3, 9, 2, 2]

start_time1 = time.perf_counter()
test1 = majority_element(test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:

Время: 4.999999873689376e-06 секунд

Память: 14.140625 Мбайт

2. Минимальное значение

```
import time
import psutil
from lab2.task5.src.majority_element import majority_element

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = majority_element(test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:

Время: 4.400000761961564e-06 секунд

Память: 14.1875 Мбайт

3. Максимальное значение

```
import time
import psutil
import random
from lab2.task5.src.majority_element import majority_element
```

```
test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**5+1)]

start_time3 = time.perf_counter()
test2 = majority_element(test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 0.005556999999498657 секунд
Память: 19.05859375 Мбайт

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять поиск мажоритарного элемента среди массива целых чисел и выяснили, каково время его выполнения и затраты памяти на предельных и средних значениях.

## Дополнительные задачи

## Задание №2. Сортировка слиянием +

### 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 $n$ ($1 \leq n \leq 10^5$) — 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.

– Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

- Пример:

| input.txt | output.txt |
|---|---|
| 10<br>1 8 2 1 4 7 3 2 3 6 | 1 2 1 8<br>3 4 1 2<br>1 4 1 8<br>5 6 4 7<br>1 6 1 8<br>7 8 2 3<br>9 10 3 6<br>7 10 2 6<br>1 10 1 8<br>1 1 2 2 3 3 4 6 7 8 |

**Любая корректная сортировка слиянием, делящая подмассивы на две части (необязательно равных!), будет зачтена, если успеет завершиться, уложившись в ограничения.**

Решение:

```
def merge_indexes(array, left, middle, right):
    l_array = array[left:middle+1] + [float('inf')]
    r_array = array[middle+1:right+1] + [float('inf')]
    i = j = 0
     print(f'Индексы: {left+1}, {right+1}. Элементы: {array[left]},
{array[right]}')
    for k in range(left, right+1):
        if l_array[i] <= r_array[j]:
            array[k] = l_array[i]
            i += 1
        else:
            array[k] = r_array[j]
            j += 1


def merge_sort_indexes(array, left, right):
    if left < right:
        middle = (left + right) // 2
        merge_sort_indexes(array, left, middle)
        merge_sort_indexes(array, middle+1, right)
        merge_indexes(array, left, middle, right)
        return array
```

1. Функция merge_indexes принимает список array, левую границу left, средний индекс middle и правую границу right, разбивает список на две половины, объединяет их в отсортированный список.

Функция merge_sort_indexes рекурсивно разделяет список пополам, сортирует две половины и объединяет их, используя функцию merge_indexes.

При вызове функции merge_sort_indexes(array, 0, len(array)-1), начинается сортировка с индекса 0 и заканчивается на последнем элементе списка.

2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Входные данные:

```
10
1 8 2 1 4 7 3 2 3 6
```

Выходные данные:

Индексы: 1, 2. Элементы: 1, 8
Индексы: 1, 3. Элементы: 1, 2
Индексы: 4, 5. Элементы: 1, 4
Индексы: 1, 5. Элементы: 1, 4
Индексы: 6, 7. Элементы: 7, 3
Индексы: 6, 8. Элементы: 3, 2
Индексы: 9, 10. Элементы: 3, 6
Индексы: 6, 10. Элементы: 2, 6
Индексы: 1, 10. Элементы: 1, 7

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab2.task2.src.merge_sort_indexes import merge_sort_indexes

test_nums1 = [1, 8, 2, 1, 4, 7, 3, 2, 3, 6]

start_time1 = time.perf_counter()
test1 = merge_sort_indexes(test_nums1, 0, len(test_nums1)-1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 7.280000045284396e-05 секунд
Память: 14.1796875 Мбайт

2. Минимальное значение

```
import time
import psutil
from lab2.task2.src.merge_sort_indexes import merge_sort_indexes

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = merge_sort_indexes(test_nums2, 0, len(test_nums2)-1)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 1.7000002117129043e-06 секунд
Память: 14.2109375 Мбайт

3. Максимальное значение

```
import time
import psutil
import random
from lab2.task2.src.merge_sort_indexes import merge_sort_indexes

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**5+1)]

start_time3 = time.perf_counter()
test3 = merge_sort_indexes(test_nums3, 0, len(test_nums3)-1)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 1.3333453999994163 секунд
Память: 18.23828125 Мбайт

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

каково время ее выполнения и затраты памяти на предельных и средних значениях.

## 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<br>1 5 8 12 13<br>5<br>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.

Решение:

```
def binary_search(array, target):
    left, right = 0, len(array) - 1
    while left <= right:
        middle = left + (right - left) // 2
        if array[middle] == target:
            return middle
        elif array[middle] < target:
            left = middle + 1
        else:
            right = middle - 1
    return -1

def search_elements(array, targets):
    result = []
    for target in targets:
        index = binary_search(array, target)
        result.append(index)
    return result
```

1. Функция binary_search принимает отсортированный массив array и искомый элемент target. Затем устанавливаются индексы left и right для указания на начало и конец массива. Затем запускается цикл, в котором пока left <= right, вычисляется средний индекс middle. Если элемент array[middle] равен target, функция возвращает middle. Если элемент array[middle] меньше target, индекс left увеличивается на 1. Иначе, индекс right уменьшается на 1. Если элемент не найден, возвращается -1.

   Функция search_elements принимает массив array и массив targets с элементами, которые нужно найти. Она проходит по каждому элементу в targets, вызывая binary_search для поиска каждого элемента в array. Результат каждого поиска добавляется в список result, который в итоге возвращается.
2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:

Входные данные:

```
5
1 5 8 12 13
5
8 1 23 1 11
```

Выходные данные:

```
2, 0, -1, 0, -1
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab2.task4.src.binary_search import search_elements

test_nums1 = [1, 5, 8, 12, 13]
test_targets1 = [8, 1, 23, 1, 11]

start_time1 = time.perf_counter()
test1 = search_elements(test_nums1, test_targets1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
```

```
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 6.199999916134402e-06 секунд
Память: 14.48046875 Мбайт

2. Максимальное значение

```
import time
import psutil
import random
from lab2.task4.src.binary_search import search_elements

test_nums2  =  [random.randint(-10**9,  10**9+1)  for  _  in  range(1,
10**5+1)]
test_targets2 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**5+1)]

start_time1 = time.perf_counter()
test1 = search_elements(test_nums2, test_targets2)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 0.182346100000359 секунд
Память: 23.859375 Мбайт

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять бинарный поиск некоторых элементов среди массива элементов и выяснили, каково время его выполнения и затраты памяти.

**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**.

Решение:

```
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
```

1. Функция find_max_subarray принимает два аргумента: n - длина массива и сам массив array. Внутри функции создаются переменные max_sum (максимальная сумма), left (индекс начала подмассива с максимальной суммой), right (индекс конца подмассива с максимальной суммой) и current_sum (текущая сумма подмассива).
   Затем происходит итерация по элементам массива array. Для каждого элемента массива обновляется текущая сумма подмассива (current_sum) путем добавления значения элемента. Если текущая сумма становится отрицательной, то она обнуляется, так как мы ищем максимальную сумму подмассива. Затем проверяется, является ли текущая сумма больше максимальной суммы. Если да, то обновляется max_sum и запоминаются индексы left и right.

В результате функция возвращает максимальную сумму подмассива (max_sum), индексы начала и конца этого подмассива (left и right).

2. Открываем файл input.txt, содержащий входные данные. Считываем их и сохраняем в нужные нам переменные, после чего закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, вызываем функцию, открываем файл output.txt, записываем в него результат работы программы и закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:

Входные данные:

```
4
1 8 2 10
```

Выходные данные:

```
Максимальная сумма: 21, начальный индекс: 0, конечный индекс: 3
```

Тесты времени выполнения и затраченной памяти.

1. Пример

```
import time
import psutil
from lab2.task7.src.find_max_subarray import find_max_subarray

test_nums1 = [1, 8, 2, 10]

start_time1 = time.perf_counter()
test1 = find_max_subarray(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память:  {psutil.Process().memory_info().rss  /  1024**2} Мбайт')
```

Результат:

Время: 5.200000487093348e-06 секунд

Память: 14.2109375 Мбайт

2. Минимальное значение

```
import time
import psutil
from lab2.task7.src.find_max_subarray import find_max_subarray

test_nums2 = [0]
```

```
start_time2 = time.perf_counter()
test2 = find_max_subarray(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 3.9000005926936865e-06 секунд
Память: 14.453125 Мбайт

3. Максимальное значение

```
import time
import psutil
import random
from lab2.task7.src.find_max_subarray import find_max_subarray

test_nums3   =   sorted(random.randint(10**5,   10**9+1)   for   _   in
range(1, 10**5+1))

start_time3 = time.perf_counter()
test3 = find_max_subarray(len(test_nums3), test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память:   {psutil.Process().memory_info().rss   /   1024**2}
Мбайт')
```

Результат:
Время: 0.010346500000196102 секунд
Память: 18.375 Мбайт

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять поиск максимального подмассива массива целых чисел за линейное время и выяснили, каково время его выполнения и затраты памяти на предельных и средних значениях.

## 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. Помимо этого протестировали время работы данных алгоритмов и затрачиваемую на их выполнение память.