

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

Выполнил:

Буй Тхук Хуен - К3139

Проверила:

Афанасьев А.В.

Санкт-Петербург

2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №3. Число инверсий	6
Задача №4. Бинарный поиск	11
Дополнительные задачи	
Задача №5. Представитель большинства	13
Задача №7. Поиск максимального подмассива за линейное время	16
Вывод	21

Задачи по варианту

TASK 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 случайных 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

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr)//2
    left = arr[:mid]
    right = arr[mid:]
    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)

def merge(left, right):
    result = []
    while len(left) > 0 and len(right) > 0:
        if left[0] > right[0]:
            result.append(right.pop(0))
        result.append(left.pop(0))
```

```

    result.extend(left)
    result.extend(right)
    return result

if __name__ == '__main__':
    f1 = open('input.txt', 'r')
    f2 = open('output.txt', 'w')

    n = int(f1.readline())
    num = list(map(int, f1.readline().strip().split()))
    if (n <= 1 or n >= 2 * (10 ** 4)) or not all([abs(i) <= 10 ** 9 for i in
num]):
        print('Ввод неверен')

    f2.write(' '.join(map(str, merge_sort(num))))

```

- Текстовое объяснение решения

Сначала открываю файлы для чтения (*input.txt*) и записи (*output.txt*) с помощью функции *open()*, входные данные взяты из файла *input.txt*, использую алгоритм сортировки слиянием. Результаты записываются в файл *output.txt*.

Функция *merge_sort()* принимает список *arr* в качестве входных данных и возвращает отсортированный список.

- + если длина входного списка *arr* равна 1 или меньше, вернуть исходный список (так как он уже отсортирован).
- + разделить входной список на две половины, левую и правую, используя среднюю точку *mid*.
- + вызвать *merge_sort()* для каждой половины, левой и правой, чтобы отсортировать их по отдельности.
- + вызвать *merge()*, чтобы объединить отсортированные левый и правый списки в один отсортированный список.

Функция *merge()* принимает два отсортированных списка, левый и правый, и возвращает один отсортированный список.

- + создать пустой список *result* для хранения объединенного результата.

- + хотя и левый, и правый списки имеют элементы, сравнить наименьший элемент из каждого списка и добавить его к *result*. Если один список пуст, добавить все оставшиеся элементы из другого списка.
- + Если в одном списке все еще есть элементы, добавить их к результату.
- + Возврат полностью объединенного и отсортированного списка.

- Результат работы кода

```
Merge sort on array of size 1000: 0.00100 seconds, memory usage: 0.01 MB
Merge sort on array of size 10000: 0.06663 seconds, memory usage: 0.27 MB
Merge sort on array of size 100000: 6.70234 seconds, memory usage: 2.79 MB
Insertion sort on array of size 1000: 0.01500 seconds, memory usage: 0.00 MB
Insertion sort on array of size 10000: 1.65362 seconds, memory usage: 0.00 MB
```

Тест		Время (s)	Затраты памяти (MB)
лучший случай	Insertion sort	0.01500	0.00
	Merge sort	0.00100	0.01
средний случай	Insertion sort	1.65362	0.00
	Merge sort	0.06663	0.27
худший случай	Insertion sort	лимит времени превышен	
	Merge sort	6.70234	2.79

Insertion sort имеют временную сложность $O(n^2)$ в худшем случае поэтому, когда n слишком велико, программа не запустится. Merge sort имеют временную сложность $O(n \log n)$ во всех случаях, поэтому имеет более высокую производительность, чем Insertion sort, когда список большой, но требует больше места в памяти и имеет более высокую сложность.

TASK 3: 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

```
import time
import tracemalloc

def merge(arr, temp_arr, left, mid, right):
    i = left
    j = mid+1
    m = left
    count = 0

    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
```

```

        temp_arr[m] = arr[i]
        i += 1
    else:
        temp_arr[m] = arr[j]
        count += mid - i + 1
        j += 1
    m += 1

while i <= mid:
    temp_arr[m] = arr[i]
    i += 1
    m += 1

while j <= right:
    temp_arr[m] = arr[j]
    j += 1
    m += 1

for i in range(left, right+1):
    arr[i] = temp_arr[i]

return count

def merge_sort(arr, temp_arr, left, right):
    count = 0
    if left < right:
        mid = (left + right)//2

        count += merge_sort(arr, temp_arr, left, mid)
        count += merge_sort(arr, temp_arr, mid + 1, right)
        count += merge(arr, temp_arr, left, mid, right)

    return count

def count_inversion(arr):
    temp_arr = [0]*len(arr)
    return merge_sort(arr, temp_arr, 0, len(arr)-1)

if __name__ == '__main__':
    f1 = open('input.txt', 'r')
    f2 = open('output.txt', 'w')

    start = time.perf_counter()
    n = int(f1.readline())
    num = list(map(int, f1.readline().strip().split()))
    if (n <= 1 or n >= 10 ** 5) or not all([abs(i) <= 10 ** 9 for i in num]):

```

```
print('Ввод неверен')

result = count_inversion(num)
f2.write(str(result))
stop = time.perf_counter()
print("time:", stop - start)
print('memory usage:', tracemalloc.get_traced_memory()[1], 'bytes')
```

- Текстовое объяснение решения.

Сначала открываю файлы для чтения (*input.txt*) и записи (*output.txt*) с помощью функции *open()*, входные данные взяты из файла *input.txt*, Результаты записываются в файл *output.txt*. (*arr*: Исходный массив целых чисел. *temp_arr*: временный массив, используемый для слияния. *left*, *mid*, *right*: индексы, определяющие текущий подмассив.)

Функция *merge()*: объединяет две отсортированные половины массива и подсчитывает инверсии между ними.

- + Если $arr[i] \leq arr[j]$, он копирует $arr[i]$ в $temp_arr[k]$ и перемещает i вперед.
- + Если $arr[i] > arr[j]$, все оставшиеся элементы в левом подмассиве (от i до $middle$) будут образовывать инверсии с $arr[j]$. Количество этих инверсий равно $(mid - i + 1)$.
- + После слияния оставшиеся элементы из обоих подмассивов копируются в $temp_arr$ и, наконец, отсортированные элементы копируются обратно в arr .

Функция *merge_sort()*: Если *left* не меньше правого, это означает, что в подмассиве есть 1 или 0 элементов, которые не могут иметь инверсий.

- + делит массив на две половины: слева к середине и от середины + 1 вправо.
- + Функция вызывает себя рекурсивно для сортировки левой половины (*left*, *mid*) и правой половины (*mid* + 1, *right*).
- + количество инверсий обеих половин добавляются к *count*.

- + После сортировки обеих половин вызывается функция `merge()`, которая объединяет две половины вместе и подсчитывает любые инверсии, которые могут пересечь среднюю точку.
- + возвращает общее количество инверсий, найденных в текущем подмассиве.

Функция `count_conversions()`: создает временный массив той же длины, что и `arr`. Она вызывает `merge_sort()`, чтобы начать рекурсивный подсчет инверсий.

- Результат работы кода на примерах из текста задачи:

The screenshot shows a code editor with two files: `input.txt` and `output.txt`. `input.txt` contains two lines: `10` and `1 8 2 1 4 7 3 2 3 6`. `output.txt` contains one line: `17`. Below the files, the execution statistics are displayed: `time: 0.00027459999910206534` and `memory usage: 8708 bytes`.

Тест	Время выполнения (s)	Затраты памяти (bytes)
10 1 8 2 1 4 7 3 2 3 6	0.00027459999910206534	8708

TASK 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.

- **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 $a_0 < a_1 < \dots < a_{n-1}$ 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, \dots, 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.

- Листинг кода

```
import time
import tracemalloc

def binary_search(arr, x):
    left, right = 0, len(arr)
    while left <= right:
        mid = (left + right) // 2
        if mid < 0 or mid >= len(arr):
            return -1
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1

def count_indices(list1, list2):
    result = []
    for j in list2:
        index = binary_search(list1, j)
        result.append(index)
    return result
```

```

if __name__ == '__main__':
    f1 = open('input.txt', 'r')
    f2 = open('output.txt', 'w')
    start = time.perf_counter()

    n = int(f1.readline())
    a = list(map(int, f1.readline().strip().split()))
    m = int(f1.readline())
    b = list(map(int, f1.readline().strip().split()))
    if (n < 1 or n > 10**5) and (m < 1 or m > 10**5) or not all([abs(i) <=
10**9 for i in a]) or not all([abs(j) <= 10**9 for j in b]):
        print('Ввод неверен')

    result = count_indices(a, b)
    f2.write(' '.join(map(str, result)))
    stop = time.perf_counter()
    print(f'time: {stop-start: .20f} s')
    print('memory usage:', tracemalloc.get_traced_memory()[1], 'bytes')

```

- Текстовое объяснение решения.

Сначала открываю файлы для чтения (*input.txt*) и записи (*output.txt*) с помощью функции *open()*, входные данные взяты из файла *input.txt*, Результаты записываются в файл *output.txt*. (arr: Отсортированный массив, в котором мы хотим выполнить поиск. x: Значение, которое мы ищем. left: 0, начальным индексом массива. right : len(arr).)

Функция *binary()*:

- + он вернет -1, если *mid* находится за пределами допустимого диапазона индекса.
- + Если средний элемент равен *x*, мы возвращаем *mid*
- + Если средний элемент < *x*, это означает, что *x* должен находиться в правой половине массива, поэтому мы устанавливаем *left = mid + 1*.
- + Если средний элемент > *x*, это означает, что *x* должен находиться в левой половине массива, поэтому мы устанавливаем *right = mid - 1*.
- + Если цикл завершается без нахождения *x*, функция возвращает -1, что указывает на отсутствие *x* в массиве.

Функция `count_indices()`: `list1`: The sorted list where we are searching for values. `list2`: The list of values we want to check against `list1`.

- + Для каждого элемента `j` в `list2` функция вызывает `binary_search` с `list1` и `j`, чтобы найти его индекс.
- + `index`, возвращаемый `binary_search`, добавляется к списку результатов.
- + После проверки всех элементов в `list2` функция возвращает список результатов

- Результат работы кода на примерах из текста задачи:

The screenshot shows a code editor with two files: `input.txt` and `output.txt`. The `input.txt` file contains four lines of data: `5`, `1 5 8 12 13`, `5`, and `8 1 23 1 11`. The `output.txt` file contains one line of data: `2 0 -1 0 -1`. Below the files, a terminal window displays the execution time and memory usage: `time: 0.00005239999882178381 s` and `memory usage: 0 bytes`.

Тест	Время выполнения (s)	Затраты памяти (bytes)
5 1 5 8 12 13 5 8 1 23 1 11	0.00005239999882178381	0

TASK 5: Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a_1, a_2, \dots, 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

```
import time
import tracemalloc

def majority_num(arr, left, right):
    if left == right:
        return arr[left]

    mid = (left + right) // 2
    left_num = majority_num(arr, left, mid)
    right_num = majority_num(arr, mid + 1, right)
    if left_num == right_num:
```

```

        return left_num

    left_count = sum(1 for i in range(left, right + 1) if arr[i] == left_num)
    right_count = sum(1 for i in range(left, right + 1) if arr[i] == right_num)
    if left_count > (right - left + 1) // 2:
        return left_num
    return right_num

def find_majority_num(arr):
    n = len(arr)
    num = majority_num(arr, 0, n-1)
    count = sum(1 for x in arr if x == num)
    start = time.perf_counter()
    n = int(f1.readline())
    num = list(map(int, f1.readline().strip().split()))
    if (n < 1 or n > 10 ** 5) or not all([abs(num)
    return 1 if count > n//2 else 0

if __name__ == '__main__':
    f1 = open('input.txt', 'r')
    f2 = open('output.txt', 'w')
    (i) <= 10 ** 9 for i in num]):
        print('Ввод неверен')

    f2.write(str(find_majority_num(num)))
    stop = time.perf_counter()
    print(f'time: {stop-start: .20f} s')
    print('memory usage:', tracemalloc.get_traced_memory()[1], 'bytes')

```

- Текстовое объяснение решения

Открываю файлы для чтения (*input.txt*) и записи (*output.txt*) с помощью функции *open()*, входные данные взяты из файла *input.txt*, использую алгоритм сортировки выбором. Результаты записываются в файл *output.txt*.

Функция *majority_num()*: реализует подход «разделяй и властвуй» для поиска кандидата на элемент большинства.

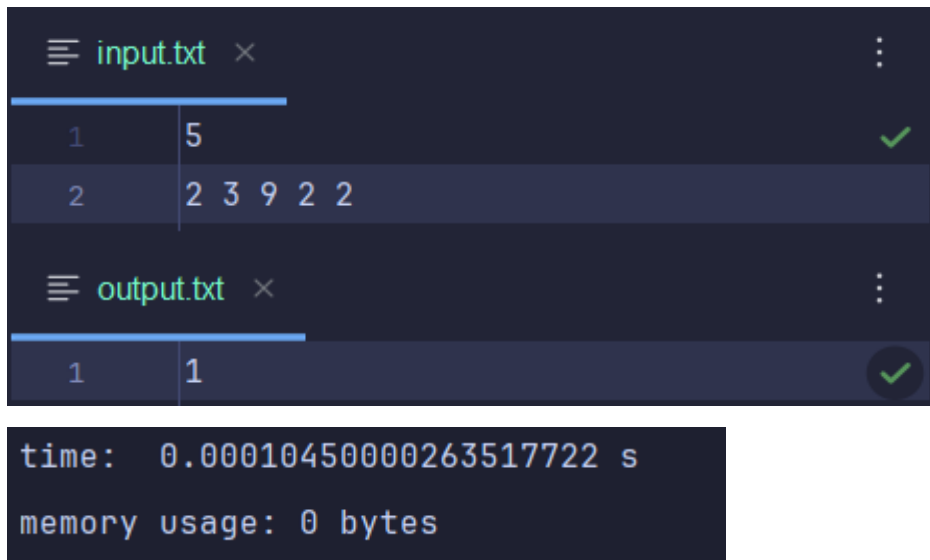
- + Если *left = right* , это означает, что у нас остался один элемент. В этом случае этот элемент возвращается как кандидат.
- + Функция вызывает себя рекурсивно для левой половины и правой половины, чтобы найти кандидатов в обеих половинах.

- + Если оба кандидата одинаковы, это означает, что они потенциально могут быть элементом большинства, поэтому мы возвращаем этого кандидата.
- + Если кандидаты различны, мы подсчитываем, сколько раз каждый кандидат появляется в текущем сегменте массива.
- + Функция возвращает кандидата, который встречается чаще, чем половина размера сегмента.

Функция `find_majority_num()`:

- + Вызывает `majority_num()` для получения потенциального кандидата из всего массива.
- + Подсчитывает, сколько раз кандидат появляется во всем массиве.
- + Если количество кандидатов больше половины размера массива, возвращается 1. В противном случае возвращается 0.

● Результат работы кода:



The screenshot shows a code editor with two files: `input.txt` and `output.txt`. Below each file is a table showing the results of its execution. The `input.txt` table has two rows: the first row contains the number 1 and the value 5, and the second row contains the number 2 and the array [2, 3, 9, 2, 2]. The `output.txt` table has one row: the number 1 and the value 1. Both tables have a green checkmark in the rightmost column. Below the tables, a terminal window displays the execution time and memory usage.

input.txt		
1	5	✓
2	2 3 9 2 2	

output.txt		
1	1	✓

time: 0.00010450000263517722 s
memory usage: 0 bytes

```

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

```

```

time: 0.00006099999882280827 s
memory usage: 0 bytes

```

Тест	Время выполнения (s)	Затраты памяти (bytes)
5 2 3 9 2 2	0.00010450000263517722	0
4 1 2 3 4	0.00006099999882280827	0

TASK 7: 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 randomного 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

```
# Поиск максимального подмассива за линейное время

import time

import tracemalloc

def find_max_subarray(arr):

    max_sum = float('-inf')

    current_sum = 0

    start_index = 0

    end_index = 0

    temp_start_index = 0

    for i in range(len(arr)):

        current_sum += arr[i]

        if current_sum > max_sum:

            max_sum = current_sum

            start_index = temp_start_index

            end_index = i

        if current_sum < 0:

            current_sum = 0

            temp_start_index = i+1

    return start_index, end_index, max_sum

if __name__ == '__main__':

    f1 = open('input.txt', 'r')

    f2 = open('output.txt', 'w')

    start = time.perf_counter()

    n = int(f1.readline())

    num = list(map(int, f1.readline().strip().split()))

    if (n <= 1 or n >= 2 * (10 ** 4)) or not all([abs(i) <= 10 ** 9 for i in num]):
```

```

print('Ввод неверен')

start_i, end_i, max_sum = find_max_subarray(num)
f2.write(' '.join(map(str, num[start_i:end_i + 1])))

stop = time.perf_counter()

print(f'time: {stop - start: .20f} s')

print('memory usage:', tracemalloc.get_traced_memory()[1], 'bytes')

```

- Текстовое объяснение решения

Сначала открываю файлы для чтения (*input.txt*) и записи (*output.txt*) с помощью функции *open()*, входные данные взяты из файла *input.txt*. Результаты записываются в файл *output.txt*.

Функция *find_max_subarray()*:

- + *max_sum*: используется для хранения наибольшей найденной суммы.
- + *current_sum*: используется для хранения текущей суммы рассматриваемого подмассива.
- + *start_index* и *end_index*: сохраняют начальный и конечный индекс подмассива с наибольшей суммой.
- + *temp_start_index*: временный индекс для отслеживания начальной позиции текущего подмассива.
- + Этот цикл перебирает каждый элемент массива *arr*, добавляя значение элемента к *current_sum*.
- + Если *current_sum* больше *max_sum*, обновите *max_sum*, *start_index* и *end_index* текущим значением.
- + Если *current_sum* меньше 0, сбросьте его на 0 и обновите *temp_start_index*, чтобы он начинался со следующего элемента.

- Unit test:

```

import unittest

from random import randint

```

```

from Lab2.Task7.task7 import find_max_subarray

class TestMaxSubarray(unittest.TestCase):

    def test_find_max_subarray_length_1000(self):
        arr = [randint(-10 ** 9, 10 ** 9) for _ in range(1000)] # Sửa đổi ở đây
        start, end, max_sum = find_max_subarray(arr)
        self.assertIsInstance(max_sum, int)

    def test_find_max_subarray_length_10000(self):
        arr = [randint(-10 ** 9, 10 ** 9) for _ in range(10000)]
        start, end, max_sum = find_max_subarray(arr)
        self.assertIsInstance(max_sum, int)

    def test_find_max_subarray_length_100000(self):
        arr = [randint(-10 ** 9, 10 ** 9) for _ in range(100000)]
        start, end, max_sum = find_max_subarray(arr)
        self.assertIsInstance(max_sum, int)

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

```

- Kết quả chạy code:

```

time: 0.00006069999653846025 s
memory usage: 0 bytes

```

Вывод

Знать, как использовать алгоритмы сортировки слиянием и бинарный поиска. Применить метод «разделяй и властвуй» для решения таких задач, как поиск Представитель Большинства, Число Инверсий. Использовать алгоритм Кадане для решения задачи поиска максимального подмассива за линейное время.