

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

Выполнил:
Марченко В.А.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета.....	2
Задания	3
Задание 1. Сортировка слиянием.....	3
Задание 2. Сортировка слиянием+.....	7
Задание 3. Число инверсий.....	9
Задание 4. Бинарный поиск	11
Задание 5. Представитель большинства.....	13
Задание 7. Поиск максимального подмассива за линейное время	15
Вывод.....	18

Задания

Задание 1. Сортировка слиянием

Используя псевдокод процедур Merge и Merge-sort из презентации к Лекции 2, написать программу сортировки слиянием на Python.

Листинг кода:

```
import random
from utils import measure_performance, writefile

def merge(A, p, q, r, data=[]):
    n1 = q - p + 1
    n2 = r - q
    L = [0] * n1
    R = [0] * n2

    for i in range(n1):
        L[i] = A[p + i]
    for j in range(n2):
        R[j] = A[q + 1 + j]

    i = j = 0
    k = p
    inv_count = 0

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

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

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

    data.append((p + 1, r + 1, A[p], A[r]))
    return inv_count
```

```

def mergesort(A, p, r, data=[]):
    inv_count = 0

    if p < r:
        q = (p + r) // 2
        inv_count += mergesort(A, p, q, data)[1]
        inv_count += mergesort(A, q + 1, r, data)[1]
        inv_count += merge(A, p, q, r, data)

    return A, inv_count, data

def mergesort_file(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    mergesort(arr, 0, n - 1)

    writefile(output_name, ' '.join(map(str, arr)))

    input_file.close()

@measure_performance
def example1():
    worst_array = sorted([random.randint(0, 1000) for _ in range(10**4)],
reverse=True)
    mergesort(worst_array, 0, len(worst_array) - 1)
    print("[WORST CASE]")

@measure_performance
def example2():
    best_array = sorted([random.randint(0, 1000) for _ in range(10**4)])
    mergesort(best_array, 0, len(best_array) - 1)
    print("[BEST CASE]")

@measure_performance
def example3():
    mergesort_file("../txtf/input.txt", "../txtf/output.txt")
    print("[FROM FILE]")

if __name__ == '__main__':
    example1()
    example2()
    example3()

```

Текстовое объяснение решения:

Этот код реализует сортировку слиянием, которая является классическим алгоритмом “разделяй и властвуй”. В нём массив разделяется на две половины, каждая из которых сортируется рекурсивно, а затем результаты сливаются. Функция `merge` отвечает за слияние двух отсортированных подмассивов. Входные подмассивы копируются во временные массивы `L` и `R`, а затем элементы сравниваются и копируются обратно в исходный массив `A`, при этом подсчитывается количество инверсий. Инверсия – это случай, когда элементы в левом подмассиве больше элементов в правом подмассиве, что говорит о том, сколько раз меньшие элементы следуют за большими. После каждого слияния, индексы граничных элементов подмассива и их значения записываются для отслеживания хода сортировки и выводятся в выходной файл.

Результат работы кода:

	Время выполнения	Затраты памяти
Наихудший случай	313.03 ms	1.8255 MB

Лучший случай	275.56 ms	1.9148 MB
---------------	-----------	-----------

Вывод по задаче:

Алгоритм сортировки слиянием, реализованный в данной задаче, позволяет не только эффективно упорядочивать массивы, но и подробно отслеживать каждый этап слияния.

Задание 2. Сортировка слиянием+

Дан массив целых чисел. Задача – отсортировать его в порядке неубывания с помощью сортировки слиянием. После каждого осуществленного слияния, выводить индексы граничных элементов и их значения.

Листинг кода:

```
from lab2.task1.src.mergesort import mergesort
from utils import measure_performance, writefile

def mergesort_fileplus(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    data = []
    mergesort(arr, 0, n - 1, data)

    result = ""
    for info in data:
        result += f"{info[0]} {info[1]} {info[2]} {info[3]}\n"
    result += ' '.join(map(str, arr))
    writefile(output_name, result)

    input_file.close()

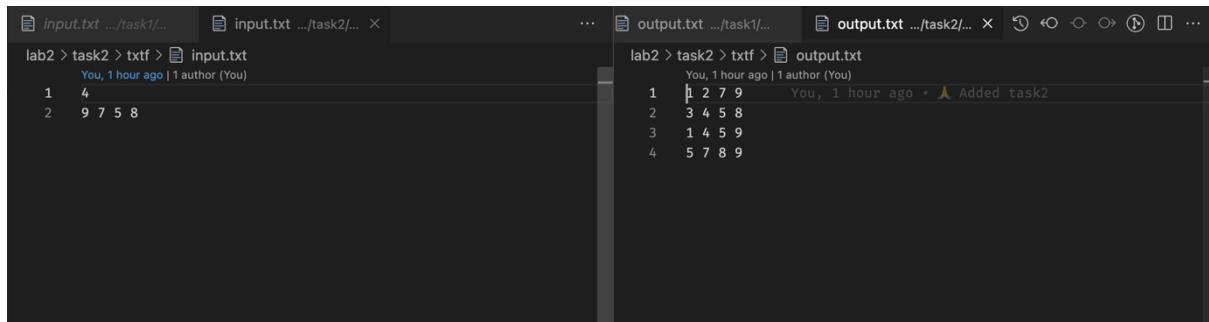
@measure_performance
def main():
    mergesort_fileplus('../txtf/input.txt', '../txtf/output.txt')

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

Текстовое объяснение решения:

Скрипт использует сортировку слиянием для упорядочивания массива, прочитанного из файла, и выводит результаты каждого шага слияния в другой файл. Скрипт читает размер массива и сам массив из входного файла, производит сортировку, сохраняя при этом подробности каждого шага слияния в список. После завершения сортировки, итоговый отсортированный массив и данные о слияниях выводятся в выходной файл

Результат работы кода:



```
lab2 > task2 > txf > input.txt
You, 1 hour ago | 1 author (You)
1 4
2 9 7 5 8

lab2 > task2 > txf > output.txt
You, 1 hour ago | 1 author (You)
1 1 2 7 9
2 3 4 5 8
3 1 4 5 9
4 5 7 8 9
```

Вывод по задаче:

В данной задаче успешно реализован алгоритм сортировки слиянием с детальной фиксацией каждого шага слияния, что позволяет не только эффективно сортировать массивы, но и тщательно анализировать процесс сортировки.

Задание 3. Число инверсий

Задача – посчитать количество инверсий в массиве при сортировке слиянием.

Листинг кода:

```
from lab2.task1.src.mergesort import mergesort
from utils import measure_performance, writefile

def inversions_file(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    _, inv_count, _ = mergesort(arr, 0, n - 1)

    writefile(output_name, str(inv_count))

    input_file.close()

@measure_performance
def main():
    inversions_file('../txtf/input.txt', '../txtf/output.txt')
    print("[FROM FILE]")

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

Текстовое объяснение решения:

Скрипт предназначен для подсчёта количества инверсий в массиве, используя алгоритм сортировки слиянием. Функция `inversions_file` читает массив целых чисел из файла, применяет алгоритм сортировки слиянием для сортировки массива и подсчёта инверсий, а затем записывает полученное количество инверсий в выходной файл.

Результат работы кода:

```
lab2 > task3 > txtf > input.txt
You, 2 hours ago | 1 author (You)
1 10
2 1 8 2 1 4 7 3 2 3 6

lab2 > task3 > txtf > output.txt
You, 2 hours ago | 1 author (You)
1 17
```

Вывод по задаче:

В данной задаче успешно выполнен подсчёт количества инверсий в массиве с помощью алгоритма сортировки слиянием. Процесс не только демонстрирует использование сортировки слиянием для определения степени упорядоченности массива, но и позволяет оценить производительность алгоритма благодаря встроенной функции измерения.

Задание 4. Бинарный поиск

В этой задаче необходимо реализовать алгоритм бинарного поиска, который позволяет очень эффективно искать объект в списках при условии, что список отсортирован.

Листинг кода:

```
from utils import measure_performance, writefile

def binarysearch(arr, x, reversed=False):
    low = 0
    high = len(arr) - 1
    result = -1

    while low <= high:
        mid = (low + high) // 2
        mid_val = arr[mid]

        if mid_val < x:
            low = mid + 1
        elif mid_val > x:
            high = mid - 1
        else:
            result = mid
            if reversed:
                low = mid + 1
            else:
                high = mid - 1

    return result

def binarysearch_file(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    array = list(map(int, input_file.readline().split()))
    k = int(input_file.readline())
    queries = list(map(int, input_file.readline().split()))

    results = [binarysearch(array, query) for query in queries]

    writefile(output_name, ' '.join(map(str, results)))

    input_file.close()

@measure_performance
def main():
    binarysearch_file("../txtf/input.txt", "../txtf/output.txt")
```

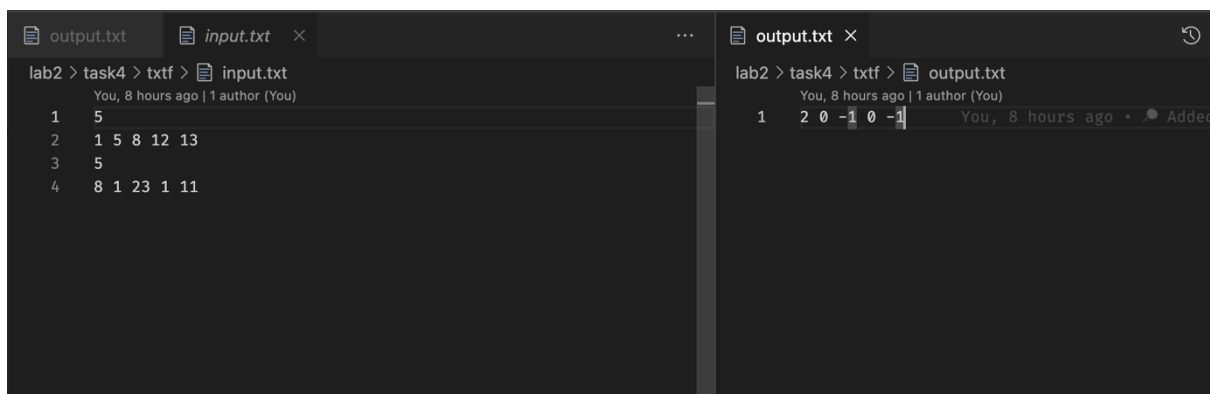
```
print("[FROM FILE]")

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

Текстовое объяснение решения:

Скрипт выполняет бинарный поиск в предварительно отсортированном массиве целых чисел. Функция `binarysearch` осуществляет поиск значения `x` в массиве `arr`, позволяя также опционально проводить обратный поиск с помощью параметра `reversed`. Скрипт включает функцию `binarysearch_file`, которая читает массив и список запросов из файла, затем использует `binarysearch` для поиска каждого запроса в массиве и записывает результаты в выходной файл.

Результат работы кода:



The image shows two side-by-side screenshots of a code editor. The left screenshot displays the content of `input.txt`, which contains four lines of data: a single number 5, a sorted array [1, 5, 8, 12, 13], the number 5, and another sorted array [8, 1, 23, 1, 11]. The right screenshot displays the content of `output.txt`, showing the results of the binary search for each query: 0, 1, -1, 0, and -1. Both screenshots show the file path `lab2 > task4 > txtf > [filename]` and a timestamp of 'You, 8 hours ago | 1 author (You)'.

Вывод по задаче:

В данной задаче успешно реализован бинарный поиск для обработки запросов на поиск элементов в предварительно отсортированном массиве. Эта реализация подчеркивает преимущества использования бинарного поиска для оптимизации процессов поиска в условиях ограниченного времени выполнения.

Задание 5. Представитель большинства

В этой задаче необходимо реализовать алгоритм поиска представителя большинства. Правило большинства - это когда выбирается элемент, имеющий больше половины голосов.

Листинг кода:

```
from lab2.task1.src.mergesort import mergesort
from lab2.task4.src.binarysearch import binarysearch
from utils import measure_performance, writefile

def majority(arr):
    n = len(arr)
    mergesort(arr, 0, n - 1)
    candidate = arr[n // 2]

    first_index = binarysearch(arr, candidate)
    last_index = binarysearch(arr, candidate, True)

    if last_index - first_index + 1 > n // 2:
        return candidate
    return -1

def majority_file(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    array = list(map(int, input_file.readline().split()))

    result = majority(array)

    writefile(output_name, str(result))

    input_file.close()

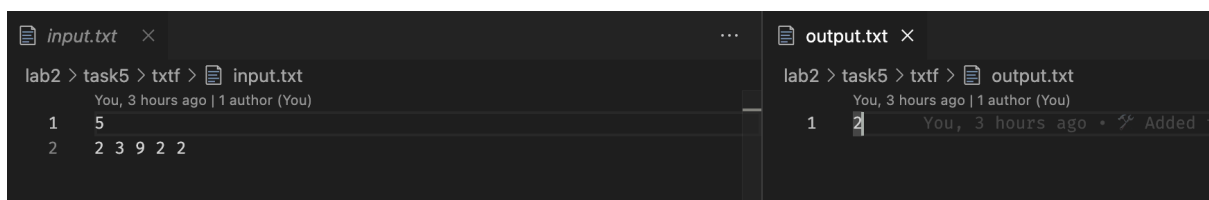
@measure_performance
def main():
    majority_file("../txtf/input.txt", "../txtf/output.txt")
    print("[FROM FILE]")

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

Текстовое объяснение решения:

Этот скрипт решает задачу нахождения элемента большинства в массиве, используя алгоритм сортировки слиянием и бинарный поиск. Функция `majority` сначала сортирует массив, после чего выбирает кандидата на элемент большинства (элемент в середине отсортированного массива) и проверяет, действительно ли этот элемент встречается более чем в половине случаев с помощью двух бинарных поисков: первый поиск находит первое вхождение кандидата, а второй – последнее. Если разница между индексами этих вхождений больше половины длины массива, кандидат считается элементом большинства. Функция `majority_file` обрабатывает ввод и вывод через файлы, прочитывая данные, выполняя проверку и записывая результат.

Результат работы кода:



The screenshot shows a code editor with two tabs: `input.txt` and `output.txt`. The `input.txt` tab is active and shows the following content:

```
lab2 > task5 > txtf > input.txt
You, 3 hours ago | 1 author (You)
1 5
2 2 3 9 2 2
```

The `output.txt` tab is also visible and shows the following content:

```
lab2 > task5 > txtf > output.txt
You, 3 hours ago | 1 author (You)
1 2
```

Вывод по задаче:

В решенной задаче успешно применены алгоритмы сортировки слиянием и бинарного поиска для определения наличия элемента большинства в массиве. Эффективность подхода демонстрируется возможностью точного и быстрого определения, превышает ли частота встречаемости кандидата половину размера массива.

Задание 7. Поиск максимального подмассива за линейное время

Необходимо найти максимальный подмассив за линейное время эффективным способом.

Листинг кода:

```
import random
from utils import measure_performance, writefile

def find_max_crossing_subarray(A, low, mid, high):
    left_sum = float('-inf')
    sum = 0
    max_left = mid
    for i in range(mid, low - 1, -1):
        sum += A[i]
        if sum > left_sum:
            left_sum = sum
            max_left = i

    right_sum = float('-inf')
    sum = 0
    max_right = mid + 1
    for j in range(mid + 1, high + 1):
        sum += A[j]
        if sum > right_sum:
            right_sum = sum
            max_right = j

    return (max_left, max_right, left_sum + right_sum)

def find_maximum_subarray(A, low, high):
    if high == low:
        return (low, high, A[low])
    else:
        mid = (low + high) // 2
        (left_low, left_high, left_sum) = find_maximum_subarray(A, low,
mid)
        (right_low, right_high, right_sum) = find_maximum_subarray(A, mid
+ 1, high)
        (cross_low, cross_high, cross_sum) =
find_max_crossing_subarray(A, low, mid, high)

        if left_sum >= right_sum and left_sum >= cross_sum:
            return (left_low, left_high, left_sum)
        elif right_sum >= left_sum and right_sum >= cross_sum:
            return (right_low, right_high, right_sum)
        else:
            return (cross_low, cross_high, cross_sum)
```

```

def maximum_subarray_file(input_name, output_name):
    input_file = open(input_name, 'r')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    writefile(output_name, ' '.join(map(str, find_maximum_subarray(arr,
0, n - 1))))

    input_file.close()

@measure_performance
def example1():
    find_maximum_subarray([random.randint(-100, 100) for _ in
range(10**4)], 0, 10**4 - 1)
    print("[RANDOM 10**4 ARRAY]")

@measure_performance
def example2():
    maximum_subarray_file('../txtf/input.txt', '../txtf/output.txt')
    print("[FROM FILE]")

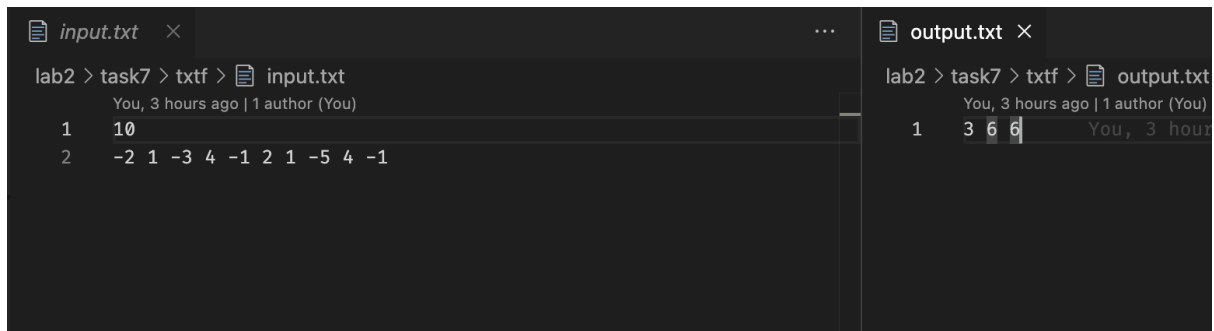
if __name__ == '__main__':
    example1()
    example2()

```

Текстовое объяснение решения:

Скрипт реализует алгоритм для нахождения подмассива с максимальной суммой в заданном массиве целых чисел, используя метод “разделяй и властвуй”. В коде определены три основные функции: `find_max_crossing_subarray`, `find_maximum_subarray` и `maximum_subarray`. Функция `find_max_crossing_subarray` вычисляет максимальную сумму подмассива, пересекающего середину, а `find_maximum_subarray` рекурсивно находит максимальный подмассив в левой и правой частях, а также среднем пересечении. В итоге, `maximum_subarray` объединяет эти компоненты, запуская процесс поиска от начала до конца массива. Результатом работы является кортеж, включающий индексы начала и конца максимального подмассива, а также его сумму, что позволяет эффективно решить задачу нахождения подмассива.

Результат работы кода:



The screenshot shows a code editor with two tabs: `input.txt` and `output.txt`. The `input.txt` tab is active and shows the following content:

```
lab2 > task7 > txtf > input.txt
You, 3 hours ago | 1 author (You)
1 10
2 -2 1 -3 4 -1 2 1 -5 4 -1
```

The `output.txt` tab is also visible and shows the following content:

```
lab2 > task7 > txtf > output.txt
You, 3 hours ago | 1 author (You)
1 3 6 6
```

Вывод по задаче:

В решенной задаче успешно применён метод “разделяй и властвуй” для нахождения подмассива с максимальной суммой в массиве целых чисел. Использование рекурсивных функций для анализа левой, правой и центральной частей массива позволяет эффективно и точно определить максимально возможную сумму подмассива.

Вывод

В ходе работы над лабораторной работой были реализованы и исследованы различные алгоритмы, включая сортировку слиянием, бинарный поиск, подсчёт инверсий, нахождение подмассива с максимальной суммой и алгоритмы для определения представителя большинства. Эти задачи демонстрируют эффективность методов разделяй и властвуй, а также бинарного поиска в решении сложных задач на данных. Использование данных методик позволило не только разработать эффективные решения, но и глубоко анализировать процесс выполнения алгоритмов.