САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Тема: Heap

Выполнил:
Лазарев Марк Олегович
К3241

Санкт-Петербург
2025 г.

Задачи по варианту

Задача №1. Построение пирамиды
Задача №2. Снова сортировка

Задачи по варианту

4 задача. Построение пирамиды

В этой задаче вы преобразуете массив целых чисел в пирамиду. Это важнейший шаг алгоритма сортировки под названием `HeapSort`. Гарантированное время работы в худшем случае составляет $O(n \log n)$, в отличие от ***среднего*** времени работы `QuickSort`, равного $O(n \log n)$. `QuickSort` обычно используется на практике, потому что обычно он быстрее, но HeapSort используется для внешней сортировки, когда вам нужно отсортировать огромные файлы, которые не помещаются в памяти вашего компьютера.

Первым шагом алгоритма `HeapSort` является создание пирамиды (heap) из массива, который вы хотите отсортировать.

Ваша задача - реализовать этот первый шаг и преобразовать заданный массив целых чисел в пирамиду. Вы сделаете это, применив к массиву определенное количество перестановок (swaps). Перестановка - это операция, как вы помните, при которой элементы a_i и a_j массива меняются местами для некоторых i и j. Вам нужно будет преобразовать массив в пирамиду, используя только $O(n)$ перестановок. Обратите внимание, что в этой задаче вам нужно будет использовать `min-heap` вместо `max-heap`.

- **Формат ввода или входного файла (input.txt).** Первая строка содержит целое число n ($1 \leq n \leq 10^5$), вторая содержит n целых чисел a_i входного массива, разделенных пробелом ($0 \leq a_i \leq 10^9$, все a_i - различны.)

- **Формат выходного файла (output.txt).** Первая строка ответа должна содержать целое число m - количество сделанных свопов. Число m должно удовлетворять условию $0 \leq m \leq 4n$. Следующие m строк должны содержать по 2 числа: индексы i и j сделанной перестановки двух элементов, **индексы считаются с 0**. После всех перестановок в нужном порядке массив должен стать пирамидой, то есть для каждого i при $0 \leq i \leq n-1$ должны выполняться условия:

  1. если $2i + 1 \leq n - 1$, то $a_i < a_{2i+1}$,
  2. если $2i + 2 \leq n - 1$, то $a_i < a_{2i+2}$.

  Обратите внимание, что все элементы входного массива различны. Любая последовательность свопов, которая менее $4n$ и после которой входной массив становится корректной пирамидой, считается верной.

- Ограничение по времени. 3 сек.

- Ограничение по памяти. 512 мб.

- Пример 1:

input.txt	output.txt
5 5 4 3 2 1	3 1 4 0 1 1 3

После перестановки элементов в позициях 1 и 4 массив становится следующим: 5 1 3 2 4.

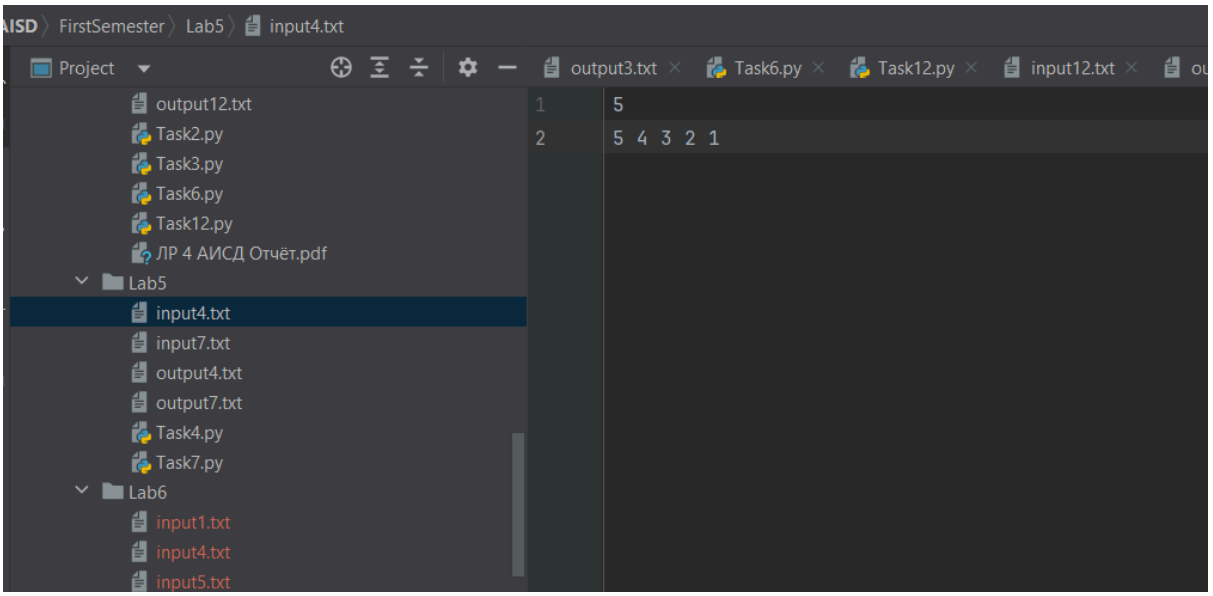
Далее, перестановка элементов с индексами 0 и 1: 1 5 3 2 4. И напоследок, переставим 1 и 3: 1 2 3 5 4, и теперь это корректная неубывающая пирамида.

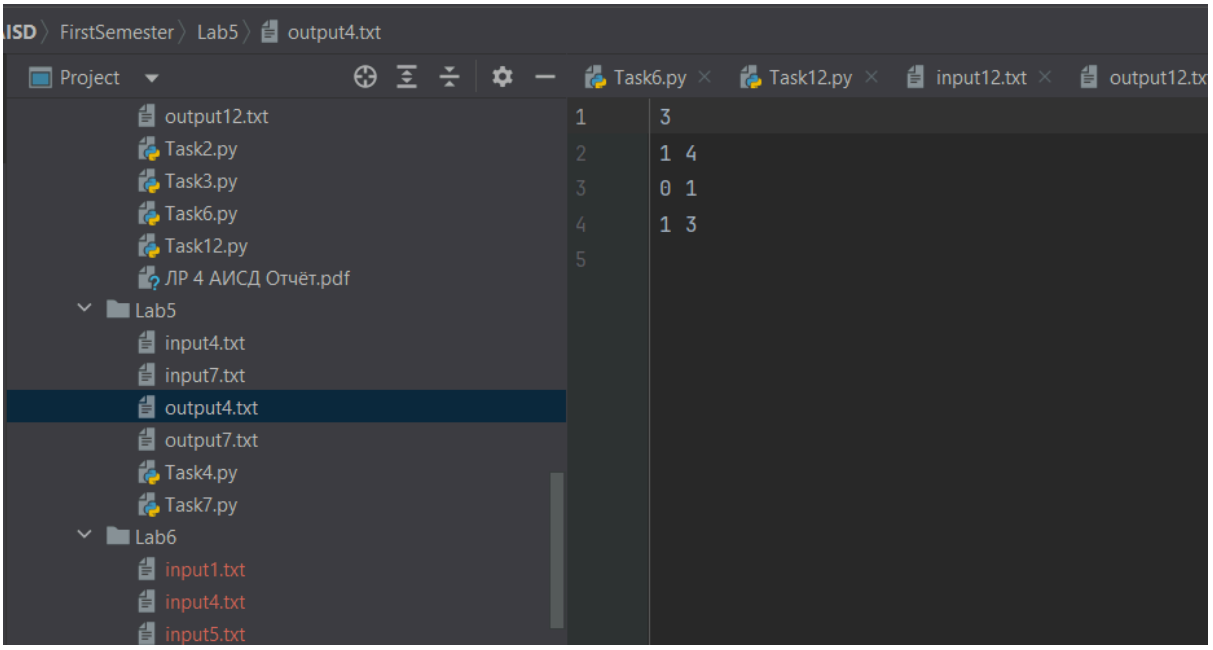
Код программы:

```
def sift_down(arr, i, swaps, n):
    min_index = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[left] < arr[min_index]:
        min_index = left
    if right < n and arr[right] < arr[min_index]:
        min_index = right

    if i != min_index:
```

```
            arr[i], arr[min_index] = arr[min_index],
arr[i]
        swaps.append((i, min_index))
        sift_down(arr, min_index, swaps, n)


def build_min_heap(arr, n):
    swaps = []
    for i in range(n // 2, -1, -1):
        sift_down(arr, i, swaps, n)
    return swaps


# Читаем входные данные
with open("input4.txt", "r") as file:
    n = int(file.readline().strip())
                        arr       =       list(map(int,
file.readline().strip().split()))

# Строим кучу
swaps = build_min_heap(arr, n)

# Записываем результат в output.txt
with open("output4.txt", "w") as file:
    file.write(f"{len(swaps)}\n")
    for i, j in swaps:
        file.write(f"{i} {j}\n")
```

Результат работы кода на примерах:


AISD 〉 FirstSemester 〉 Lab5 〉 input4.txt
Project
output3.txt
Task6.py
Task12.py
input12.txt
output12.txt
Task2.py
Task3.py
Task6.py
Task12.py
ЛР 4 АИСД Отчёт.pdf
Lab5
input4.txt
input7.txt
output4.txt
output7.txt
Task4.py
Task7.py
Lab6
input1.txt
input4.txt
input5.txt
1 5
2 5 4 3 2 1


AISD 〉 FirstSemester 〉 Lab5 〉 output4.txt
Project
Task6.py
Task12.py
input12.txt
output12.txt
output12.txt
Task2.py
Task3.py
Task6.py
Task12.py
ЛР 4 АИСД Отчёт.pdf
Lab5
input4.txt
input7.txt
output4.txt
output7.txt
Task4.py
Task7.py
Lab6
input1.txt
input4.txt
input5.txt
1 3
2 1 4
3 0 1
4 1 3
5

7 задача. Снова сортировка

Напишите программу пирамидальной сортировки на Python для последовательности в **убывающем порядке**. Проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

- **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).** Одна строка выходного файла с отсортированным по невозрастанию массивом. Между любыми двумя числами должен стоять ровно один пробел.

- Для проверки можно выбрать случай, когда сортируется массив размера $10^3, 10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; когда массив уже отсортирован в нужном порядке; когда много одинаковых элементов, всего 4-5 уникальных; средний - случайный. Сравните на данных сетах `Randomized-QuickSort`, `MergeSort`, `HeapSort`, `InsertionSort`.

- Есть ли случай, когда сортировка пирамидой выполнится за $O(n)$?

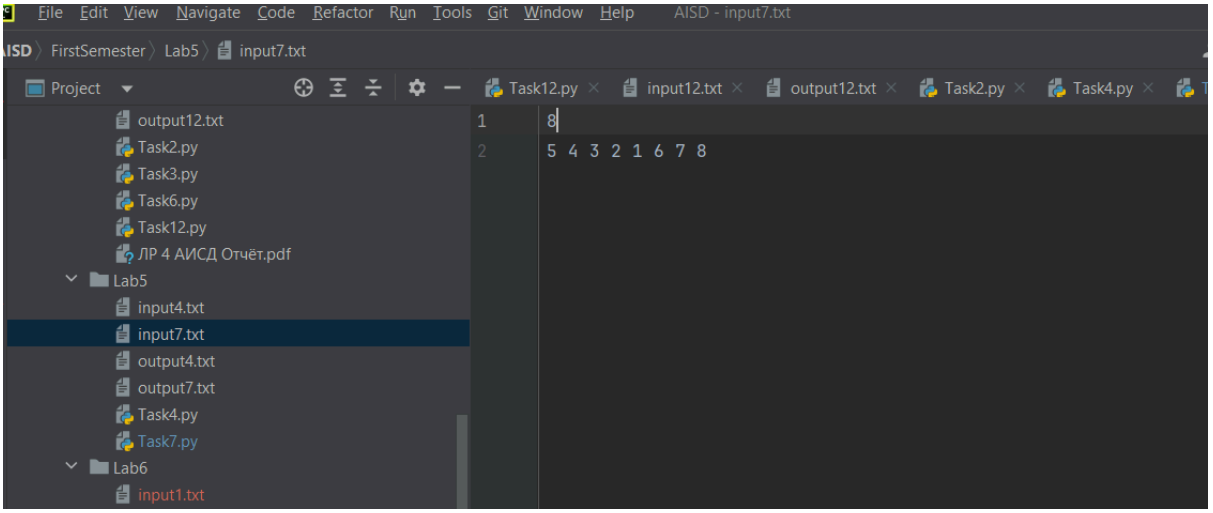
* Напишите процедуру `Max-Heapify`, в которой вместо рекурсивного вызова использовалась бы итеративная конструкция (цикл).

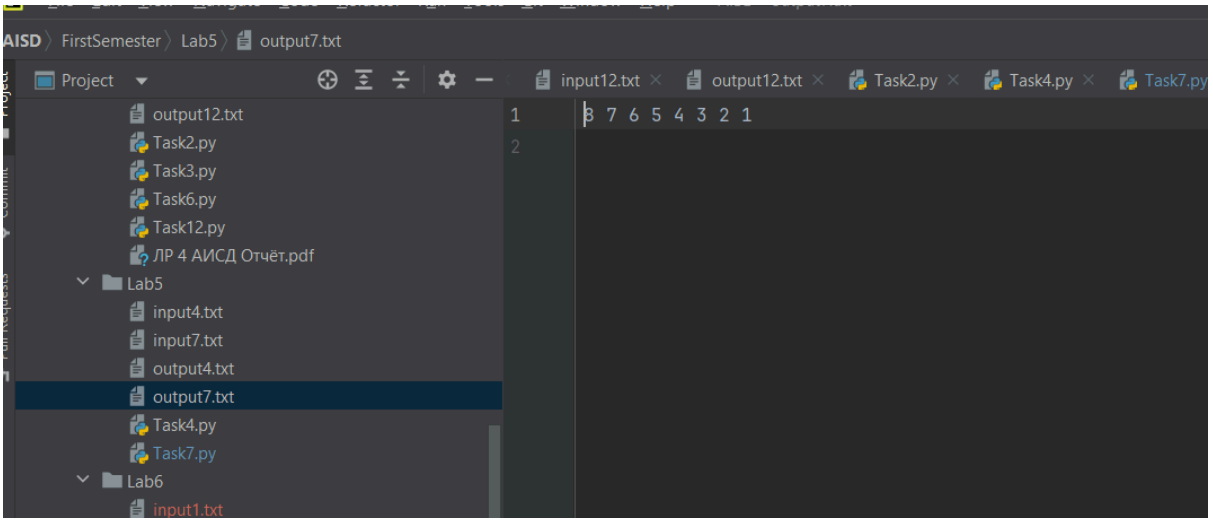
Код программы:

```
ddef max_heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    while True:
        if left < n and arr[left] > arr[largest]:
            largest = left
        if right < n and arr[right] > arr[largest]:
            largest = right
        if largest != i:
                arr[i], arr[largest] = arr[largest],
arr[i]
```

```
            i = largest
            left = 2 * i + 1
            right = 2 * i + 2
        else:
            break

def heap_sort(arr):
    n = len(arr)

    # Построение максимальной кучи
    for i in range(n // 2 - 1, -1, -1):
        max_heapify(arr, n, i)

    # Извлечение элементов из кучи
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        max_heapify(arr, i, 0)

    arr.reverse()

with open('input7.txt', 'r') as file:
    n = int(file.readline().strip())
                    arr       =       list(map(int,
file.readline().strip().split()))

heap_sort(arr)

with open('output7.txt', 'w') as file:
    file.write(' '.join(map(str, arr)) + '\n')
```

Результат работы кода на примерах:


File Edit View Navigate Code Refactor Run Tools Git Window Help
AISD - input7.txt
FirstSemester
Lab5
input7.txt
Project
Task12.py
input12.txt
output12.txt
Task2.py
Task4.py
output12.txt
Task2.py
Task3.py
Task6.py
Task12.py
ЛР 4 АИСД Отчёт.pdf
Lab5
input4.txt
input7.txt
output4.txt
output7.txt
Task4.py
Task7.py
Lab6
input1.txt
8
5 4 3 2 1 6 7 8


AISD
FirstSemester
Lab5
output7.txt
Project
input12.txt
output12.txt
Task2.py
Task4.py
Task7.py
output12.txt
Task2.py
Task3.py
Task6.py
Task12.py
ЛР 4 АИСД Отчёт.pdf
Lab5
input4.txt
input7.txt
output4.txt
output7.txt
Task4.py
Task7.py
Lab6
input1.txt
8 7 6 5 4 3 2 1