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: Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами
Вариант 14

Выполнила:
Рудникова Виктория Олеговна
К3125

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №4. 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 ai и aj 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 ($1 \le n \le 105$), вторая содержит n целых чисел ai входного массива, разделенных пробелом ($0 \le ai \le 109$, все ai - различны.)

• Формат выходного файла (output.txt). Первая строка ответа должна содержать целое число m - количество сделанных свопов. Число m должно удовлетворять условию $0 \le m \le 4n$. 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 $0 \le i \le n-1$ должны выполняться условия:

1. если $2i + 1 \le n - 1$, то $ai < a2i+1$,
2. если $2i + 2 \le n - 1$, то $ai < a2i+2$.

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

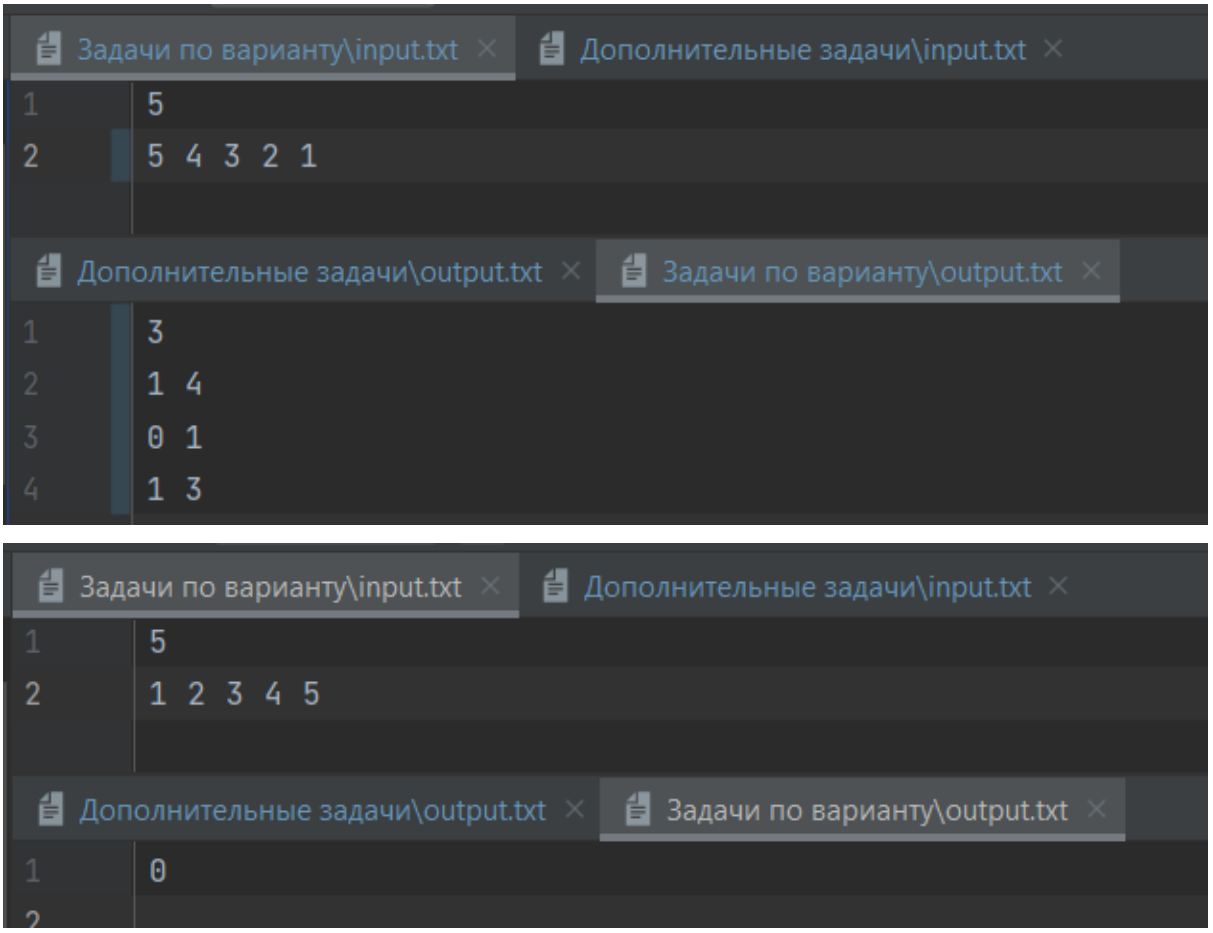
• Ограничение по времени. 3 сек.
• Ограничение по памяти. 512 мб.

```
import math
import time
import os
import psutil

process = psutil.Process(os.getpid())
t_start = time.perf_counter()


def left(idx):
    return 2 * idx + 1


def right(idx):
    return 2 * idx + 2


def min_heapify(array, idx):
    lf = left(idx)
    rt = right(idx)
    global heap_size
    if lf <= heap_size and array[lf] < array[idx]:
        small = lf
    else:
        small = idx
    if rt <= heap_size and array[rt] < array[small]:
        small = rt
    if small != idx:
        global m, swaps
        m += 1
        swaps.append([idx, small])
        array[idx], array[small] = array[small], array[idx]
        min_heapify(array, small)


def build_min_heap(array):
    global heap_size
    heap_size -= 1
    for i in range(math.floor(heap_size/2), -1, -1):
        min_heapify(array, i)


m = 0
swaps = []
with open('input.txt') as f:
    with open('output.txt', 'w') as g:
        n = int(f.readline())
        heap_size = n
        pyr = list(map(int, f.readline().split()))
```

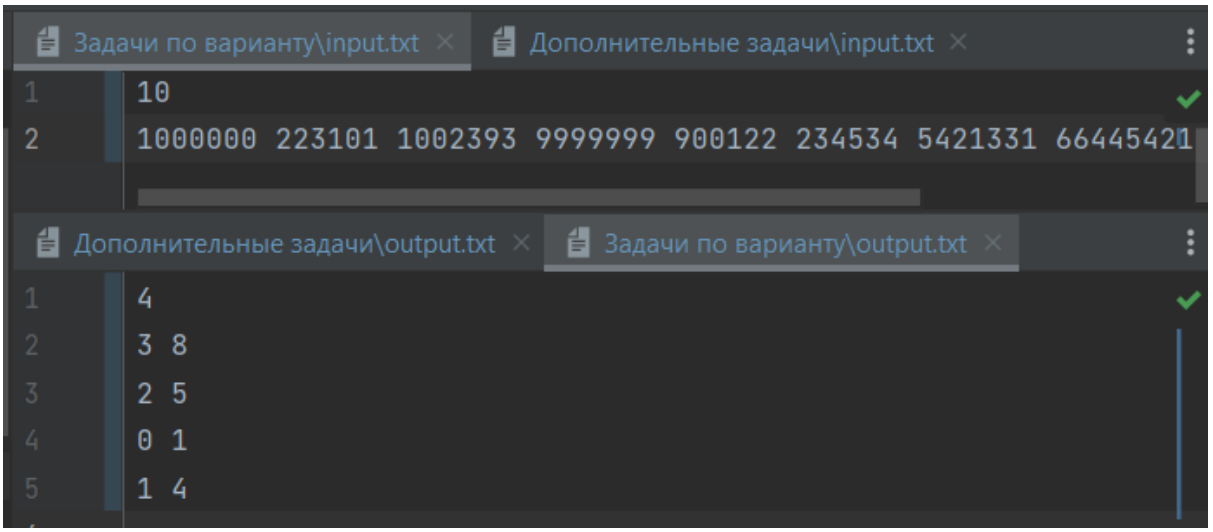
```
        build_min_heap(pyr)
        g.write(str(m) + '\n')
        for i in range(len(swaps)):
            g.write(str(swaps[i][0]) + ' ' + str(swaps[i][1]) + '\n')

print("время работы", (time.perf_counter() - t_start), "секунд")
print('память', process.memory_info().rss / 1024 ** 2, 'mb')
```

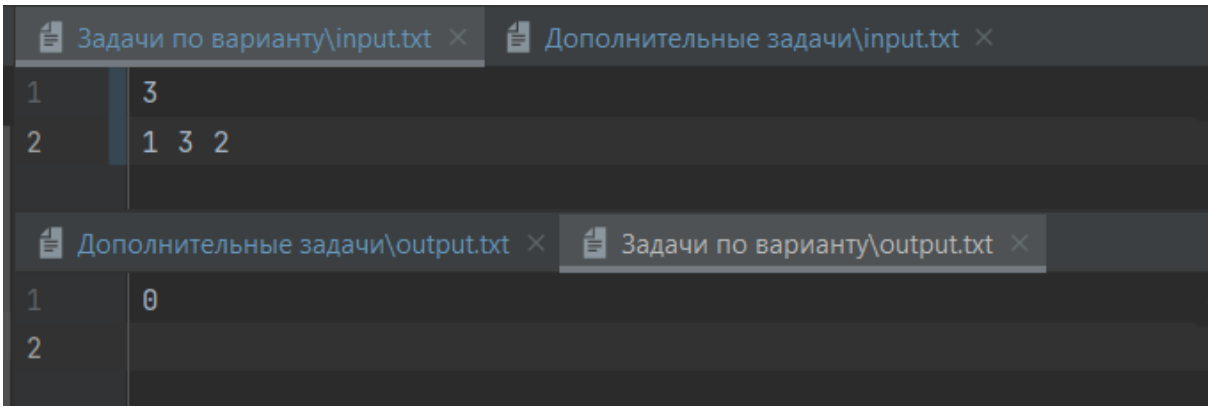
Введённый массив с помощью перестановок преобразуется в невозрастающую пирамиду, то есть преобразуется таким образом, чтобы для каждого элемента его дети (элементы с определёнными индексами относительно данного) были не больше своего родителя.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB

Вывод по задаче: в задаче я выполнила основную операцию для пирамидальной сортировки - реализовала алгоритм построения пирамиды.

Задача №7. 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 ≤ n ≤ 10**5) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10**9.

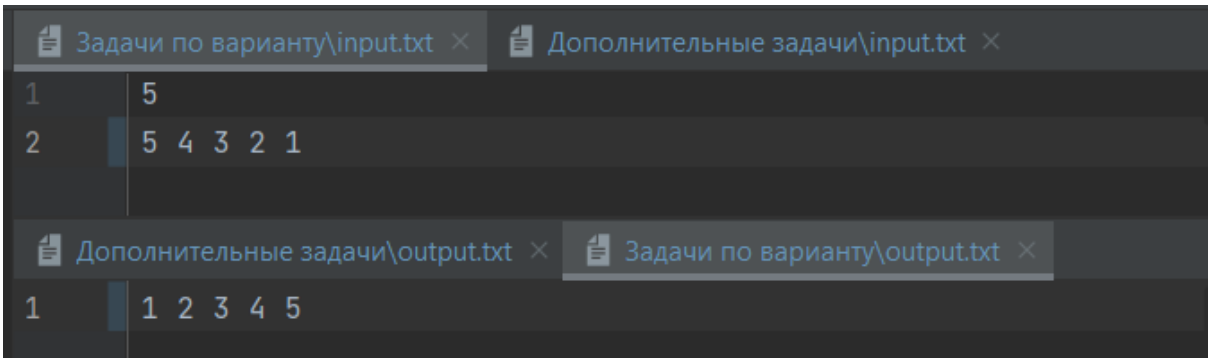
• Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным по невозрастанию массивом. Между любыми двумя числами должен стоять ровно один пробел.

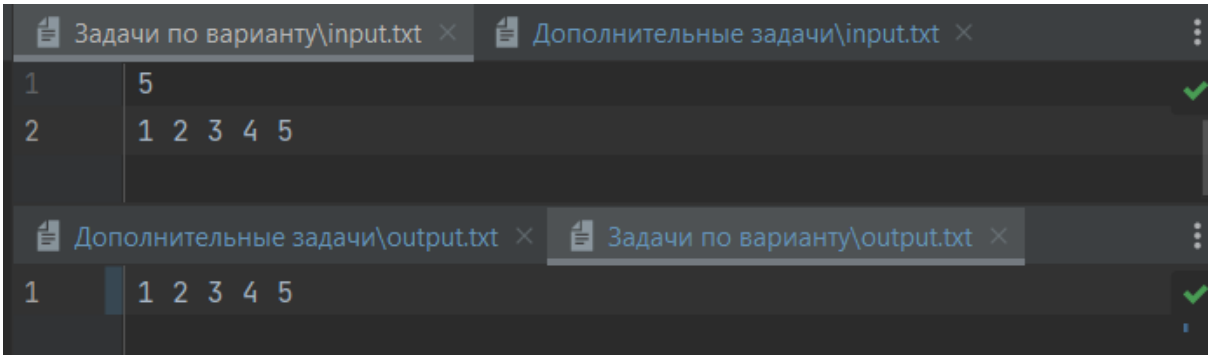
```
import math
import time
import os
import psutil

process = psutil.Process(os.getpid())
t_start = time.perf_counter()


def left(idx):
    return 2 * idx + 1


def right(idx):
    return 2 * idx + 2


def max_heapify(array, idx):
    lf = left(idx)
    rt = right(idx)
    global heap_size
    if lf <= heap_size and array[lf] > array[idx]:
        big = lf
    else:
        big = idx
    if rt <= heap_size and array[rt] > array[big]:
        big = rt
    if big != idx:
        array[idx], array[big] = array[big], array[idx]
        max_heapify(array, big)


def build_max_heap(array):
    global heap_size
    heap_size -= 1
    for idx in range(math.floor(heap_size/2), -1, -1):
```

```
        max_heapify(array, idx)
    return array


def heapsort(array):
    global heap_size, n
    build_max_heap(array)
    for i in range(n-1, 0, -1):
        array[0], array[i] = array[i], array[0]
        heap_size -= 1
        max_heapify(array, 0)


with open('input.txt') as f:
    with open('output.txt', 'w') as g:
        n = int(f.readline())
        heap_size = n
        arr = list(map(int, f.readline().split()))
        heapsort(arr)
        if arr == sorted:
            print('*')
        for i in range(n):
            g.write(str(arr[i]) + ' ')

print("время работы", (time.perf_counter() - t_start), "секунд")
print('память', process.memory_info().rss / 1024 ** 2, 'mb')
```

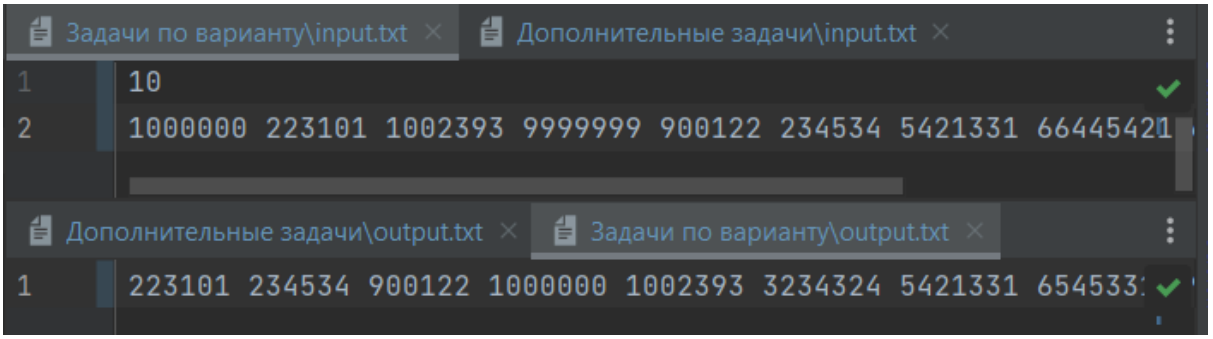
Реализован метод, превращающий массив в неубывающую пирамиду. С помощью него производится пирамидальная сортировка, которая сортирует элементы пирамиды в правильном порядке.

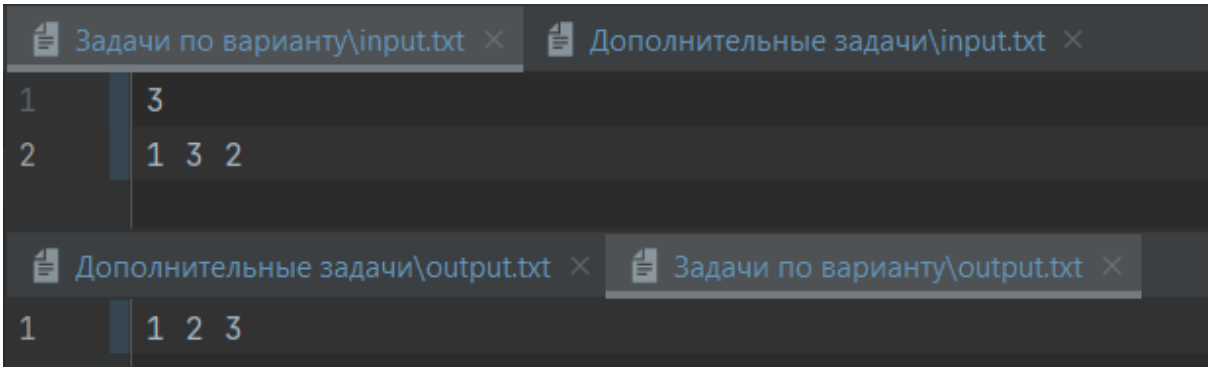
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB

Вывод по задаче: в задаче я реализовала алгоритм пирамидальной сортировки - heap sort.

Дополнительные задачи

Задача №1. Куча ли?

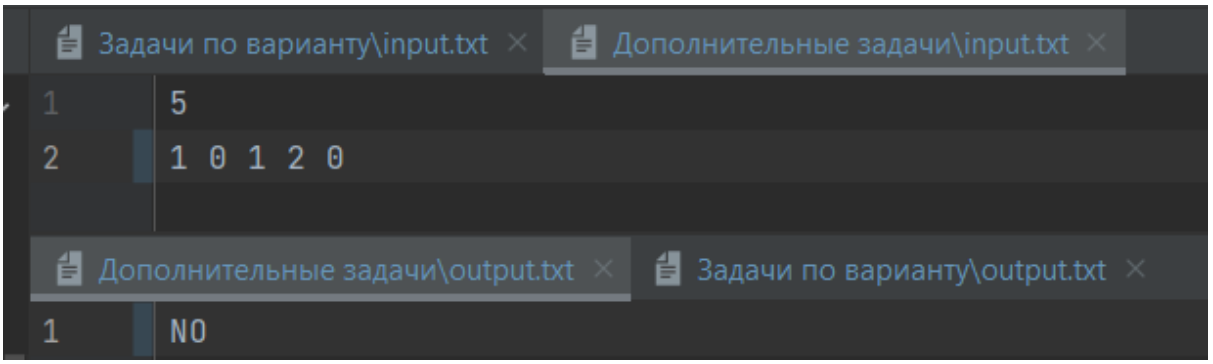
Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива. Для этого должно выполнятся основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \leq i \leq n$ выполняются условия:
1. если $2i \leq n$, то $ai \leq a2i$,
2. если $2i + 1 \leq n$, то $ai \leq a2i+1$.
Дан массив целых чисел. Определите, является ли он неубывающей пирамидой.
• Формат входного файла (input.txt). Первая строка входного файла содержит целое число n (1 ≤ n ≤ 10**6). Вторая строка содержит n целых чисел, по модулю не превосходящих 2 * 10**9.
• Формат выходного файла (output.txt). Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.
• Ограничение по времени. 2 сек.
• Ограничение по памяти. 256 мб.

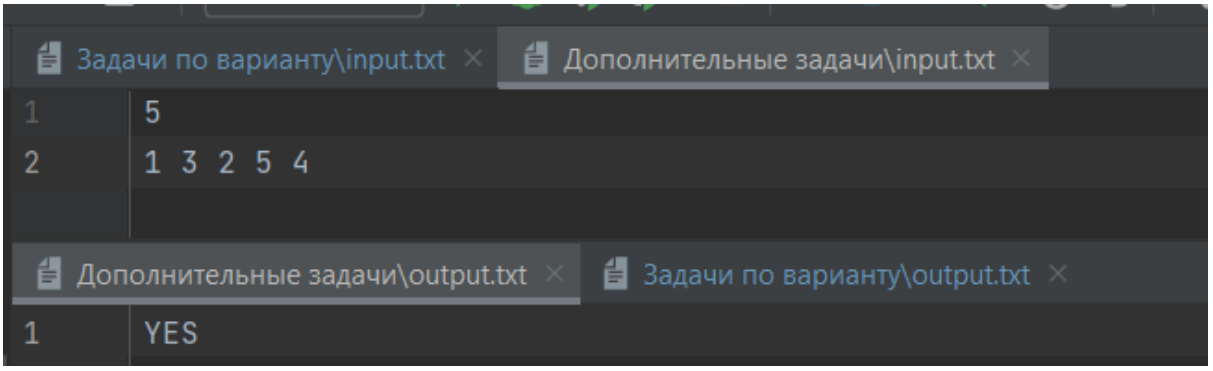
```
import math
import time
import os
import psutil

process = psutil.Process(os.getpid())
t_start = time.perf_counter()


def left(idx):
    return 2 * idx + 1


def right(idx):
    return 2 * idx + 2


with open('input.txt') as f:
    with open('output.txt', 'w') as g:
        n = int(f.readline())
        arr = list(map(int, f.readline().split()))
        for i in range(math.floor(n / 2) - 1):
            if arr[i] > arr[left(i)] or arr[i] > arr[right(i)]:
                g.write('NO')
                exit()
```

```
        g.write('YES')

print("время работы", (time.perf_counter() - t_start), "секунд")
print('память', process.memory_info().rss / 1024 ** 2, 'mb')
```

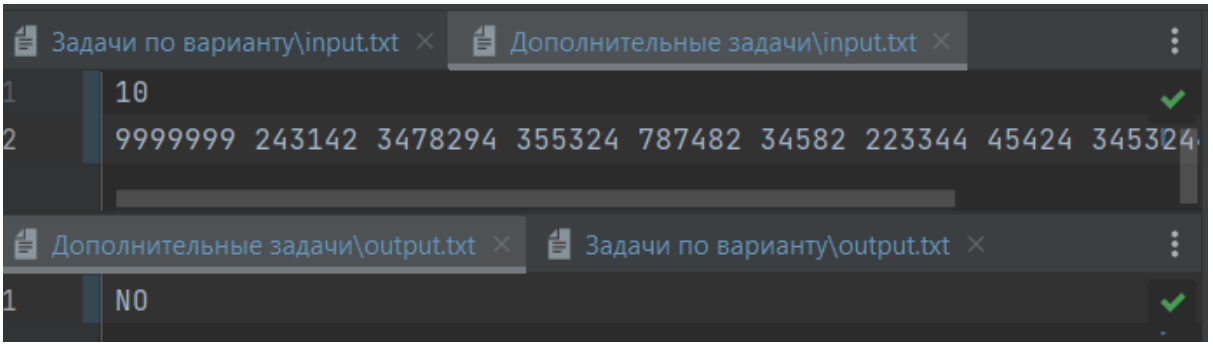
Реализованы методы left и right, возвращающие индекс левого и правого ребёнка каждого элемента. Проверяется, что все дети не меньше своего родителя. Если это верно, массив является кучей, в противном случае - нет.

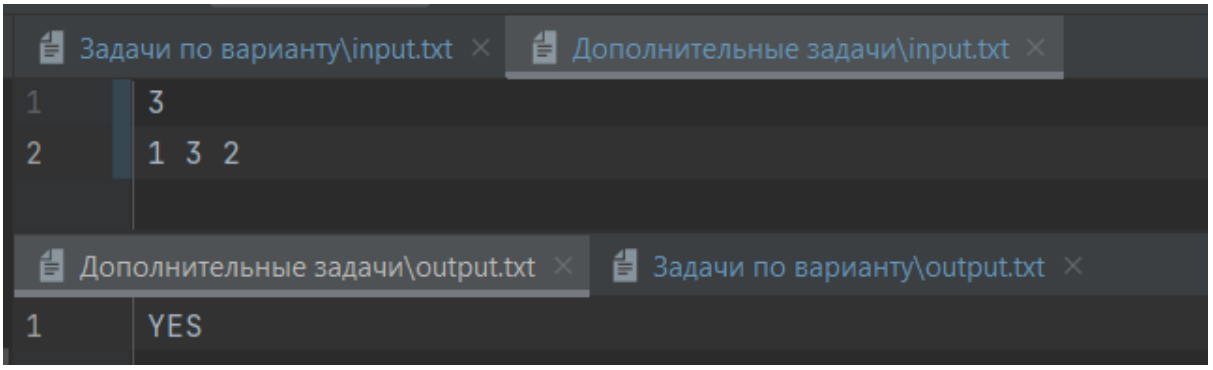
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB

Вывод по задаче: в этой задаче я реализовала алгоритм проверки, является ли введённый массив кучей.

Задача №2. Высота дерева

В этой задаче ваша цель - привыкнуть к деревьям. Вам нужно будет прочитать описание дерева из входных данных, реализовать структуру данных, сохранить дерево и вычислить его высоту.

• Вам дается корневое дерево. Ваша задача - вычислить и вывести его высоту. Напомним, что высота (корневого) дерева - это максимальная глубина узла или максимальное расстояние от листа до корня. Вам дано произвольное дерево, не обязательно бинарное дерево.

• Формат ввода или входного файла (input.txt). Первая строка содержит число узлов n ($1 \le n \le 10^{**}5$). Вторая строка содержит n целых чисел от -1 до $n-1$ – указание на родительский узел. Если i-ое значение равно -1, значит, что узел i - корневой, иначе это число является обозначением индекса родительского узла этого i-го узла ($0 \le i \le n-1$). Индексы считать с 0. Гарантируется, что дан только один корневой узел, и что входные данные представляют дерево.

• Формат вывода или выходного файла (output.txt). Выведите целое число – высоту данного дерева.

• Ограничение по времени. 3 сек.

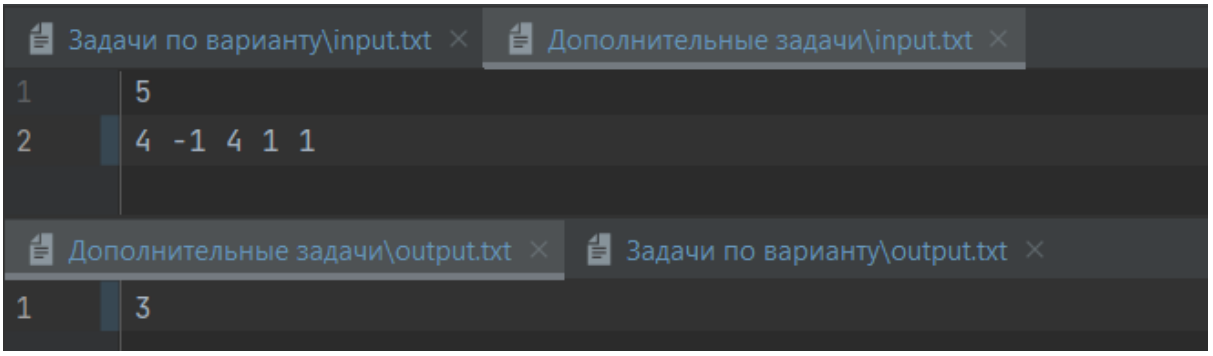
• Ограничение по памяти. 512 мб.

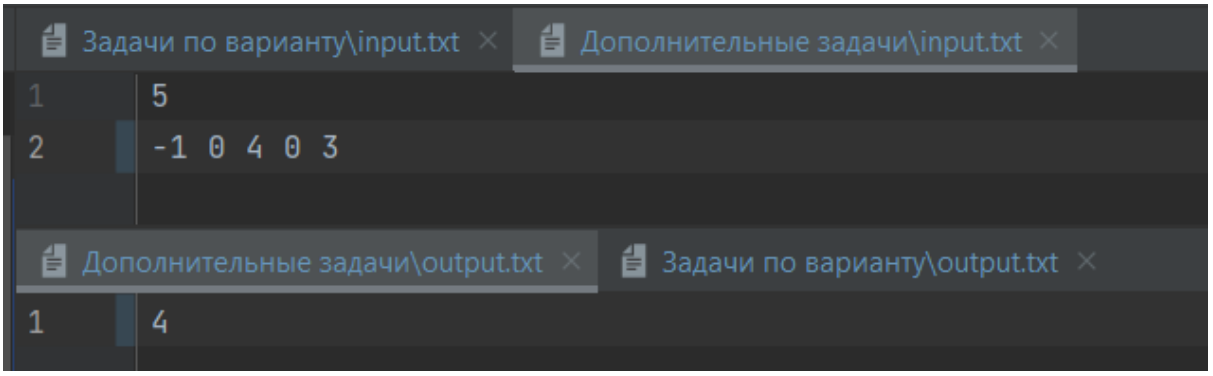
```
import time
import os
import psutil

process = psutil.Process(os.getpid())
t_start = time.perf_counter()

with open('input.txt') as f:
    with open('output.txt', 'w') as g:
        n = int(f.readline())
        arr = list(map(int, f.readline().split()))
        tree = set(range(n)) - set(arr)
        arr = {i: [arr[i], 0] for i in range(n)}
        max_h = 0
        for i in tree:
            h = 1
            elem = i
            while arr[elem][0] != -1:
                if h+1 > arr[elem][1]:
                    h += 1
                    arr[elem][1] = h
                    elem = arr[elem][0]
                else:
```

```
                    break
            max_h = max(max_h, h)
        print(time.perf_counter() - t_start)
        g.write(str(max_h))

print("время работы", (time.perf_counter() - t_start), "секунд")
print('память', process.memory_info().rss / 1024 ** 2, 'mb')
```

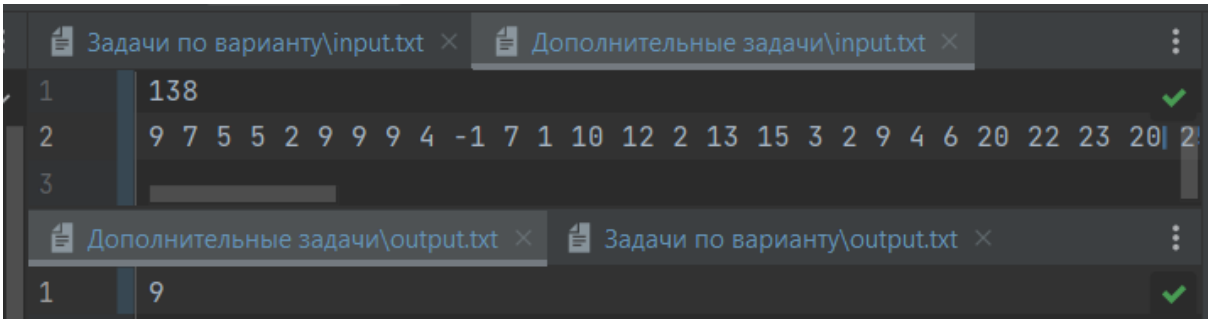
Из файла считывается дерево, хранится в памяти, реализуется алгоритм вычисления его высоты.

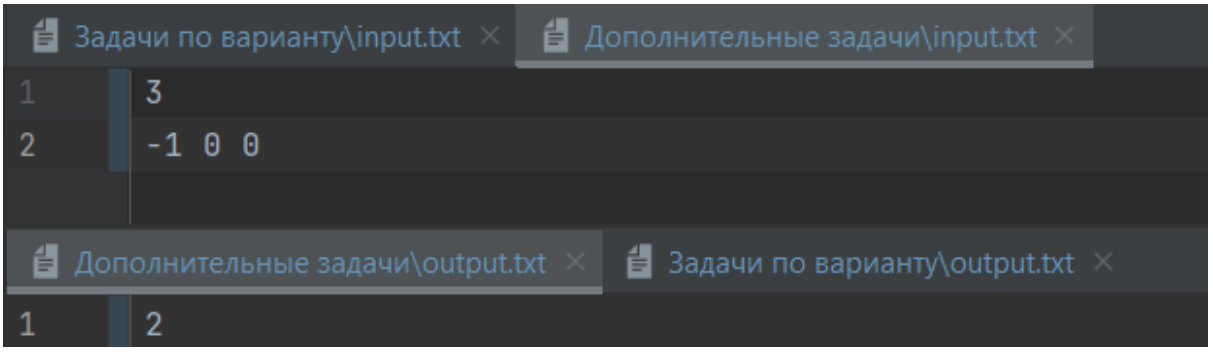
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.1875 sec	17.5947265625 KB
Пример из задачи	0.125 sec	17.7724609375 KB
Пример из задачи	0.125 sec	17.7724609375 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.15625 sec	156.7783203125 KB

Вывод по задаче: в задаче я реализовала алгоритм нахождения высоты введённого дерева.

Вывод:

В работе я вспомнила деревья и кучи, научилась строить пирамиду на массиве и реализовала пирамидальную сортировку.