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 1

Выполнила:

Аксянова А.Р

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc184510273)

[Задачи 3](#_Toc184510274)

[Задача №1. Стек 3](#_Toc184510275)

[Задача №2. Высота дерева 4](#_Toc184510276)

[Задача №4. Построение пирамиды 6](#_Toc184510277)

[Задача №7 . Снова сортировка. 8](#_Toc184510278)

[Результат работы всех задач 9](#_Toc184510279)

[Вывод 9](#_Toc184510280)

# Задачи

## Задача №1. Стек

1. Текст задачи

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 ≤ *i* ≤ *n* выполняются условия:

1. если 2*i* ≤ *n*, то *ai* ≤ *a*2*i*,
2. если 2*i* + 1 ≤ *n*, то *ai* ≤ *a*2*i*+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* ≤ 106). Вторая строка содержит *n* целых чисел, по модулю не превосходящих 2 · 109.
* **Формат выходного файла (output.txt).** Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.

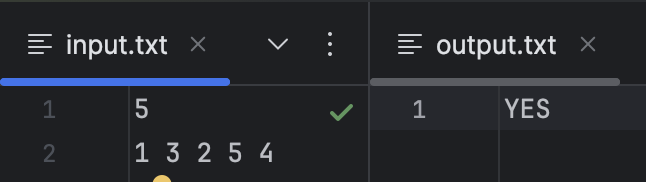
1. Листинг кода

import time  
import random  
import os  
from lab5.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 1  
  
def Pyramid(n, s):  
 for i in range(1, n + 1):  
 if 2 \* i <= n:  
 if s[i - 1] > s[2 \* i - 1]:  
 return "NO"  
 if 2 \* i + 1 <= n:  
 if s[i - 1] > s[2 \* i]:  
 return "NO"  
 return "YES"  
  
def task1():  
 n, s = inp(PATH\_INPUT)  
 res = Pyramid(n, s)  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

Основная функция Pyramid - В ней реализована проверка условий, соответствующих свойству "пирамиды". Для каждого элемента списка проверяется, выполняются ли условия, что значение текущего элемента не больше значений его потомков (если они существуют). Если какое-либо из условий не выполняется, функция возвращает "NO". В противном случае возвращается "YES"

1. Результат работы кода на примерах:



\*Результат работы кода (скрин терминала) так же будет приложен ниже

1. Вывод по задаче:

В результате выполнения задачи я научилась реализовывать алгоритм на проверку неубывающей пирамиды.

## Задача №2. Высота дерева

1. Текст задачи

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 числоузлов*n*(1 ≤ *n* ≤ 105).Втораястрокасодержит*n*целыхчиселот−1до *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 ≤ *i* ≤ *n* − 1). **Индексы считать с 0.** Гарантируется, что дан только один корневой узел, и что входные данные предстваляют дерево.
* **Формат вывода или выходного файла (output.txt).** Выведите целое число – высоту данного дерева.

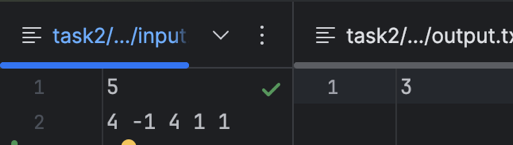
1. Листинг кода

import time  
import random  
import os  
from lab5.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 2  
  
  
def tree\_height(n, parents):  
 tree = [[] for \_ in range(n)]  
 root = -1  
  
 for i in range(n):  
 if parents[i] == -1:  
 root = i  
 else:  
 tree[parents[i]].append(i)  
  
 def height(node):  
 if not tree[node]:  
 return 1  
 max\_height = 0  
 for child in tree[node]:  
 max\_height = max(max\_height, height(child))  
 return max\_height + 1   
 return height(root)  
  
def task1():  
 n, s = inp(PATH\_INPUT)  
 res = str(tree\_height(n, s))  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 time = float(time.perf\_counter() - start)  
 print(caption( task\_numb, time))

1. Текстовое объяснение решения.

Функция tree\_height принимает два аргумента: количество вершин дерева n и массив parents, где каждый элемент указывает на индекс родителя текущей вершины. На основе этих данных строится дерево в виде списка, где каждая вершина содержит список своих потомков. Функция вычисления высоты дерева проверяет, есть ли у вершины потомки. Если их нет, высота для данной вершины равна единице. В противном случае функция вызывает себя для всех потомков текущей вершины, находя максимальную глубину и добавляя единицу, чтобы учесть текущий уровень.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась реализовывать алгоритм для нахождения высоты дерева

## Задача №4. Построение пирамиды

1. Текст задачи

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 ≤ *n* ≤ 105), вторая содержит *n* целых чисел *ai* входного массива, разделенных пробелом (0 ≤ *ai* ≤ 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 ≤ *m* ≤ 4*n*. 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 ≤ *i* ≤ *n*−1 должны выполняться условия:

* 1. если 2*i* + 1 ≤ *n* − 1, то *ai < a*2*i*+1,
  2. если 2*i* + 2 ≤ *n* − 1, то *ai < a*2*i*+2.

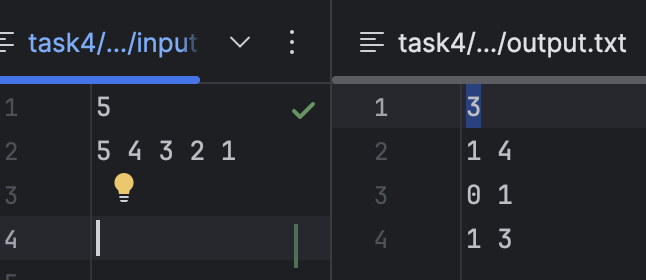
1. Листинг кода

import time  
import os  
from lab5.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 4  
  
  
def heapify(arr, n, i, swaps):  
 smallest = i  
 left = 2 \* i + 1  
 right = 2 \* i + 2  
  
 if left < n and arr[left] < arr[smallest]:  
 smallest = left  
  
 if right < n and arr[right] < arr[smallest]:  
 smallest = right  
  
 if smallest != i:  
 arr[i], arr[smallest] = arr[smallest], arr[i]  
 swaps.append((i, smallest))  
  
 heapify(arr, n, smallest, swaps)  
  
  
def build\_heap(arr):  
 n = len(arr)  
 swaps = []  
 for i in range(n // 2 - 1, -1, -1):  
 heapify(arr, n, i, swaps)  
 return swaps  
  
  
def task1():  
 n, arr = inp(PATH\_INPUT)  
 swaps = build\_heap(arr)  
 res = str(len(swaps)) + "\n"  
 res += "\n".join(f"{swap[0]} {swap[1]}" for swap in swaps)  
 outp(PATH\_OUTPUT, res.strip())  
 print(caption(task\_numb, res.strip()))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 elapsed\_time = time.perf\_counter() - start  
 print(caption(task\_numb, elapsed\_time))

1. Текстовое объяснение решения.

Основная функция build\_heap строит кучу, вызывая вспомогательную функцию heapify для каждого узла дерева, начиная с последнего уровня и поднимаясь вверх. Функция heapify находит наименьший элемент среди текущей вершины и её дочерних узлов. Если текущий узел не минимален, он меняется местами с меньшим дочерним узлом, и функция вызывается для нового положения. Перестановки фиксируются в списке swaps.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм преобразования массива в пирамиду.

## Задача №7 . Снова сортировка.

1. Текст задачи

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* ≤ 105) — число элементов в массиве. Во второй строке находятся *n* различных целых чисел, ***помодулю*** не превосходящих

109.

**Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированнымпоневозрастаниюмассивом.Междулюбымидвумячислами должен стоять ровно один пробел.

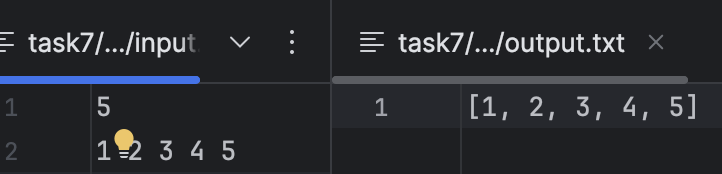
1. Листинг кода

import time  
import os  
from lab5.utils import inp, outp, caption  
  
current\_script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
os.chdir(current\_script\_dir)  
  
PATH\_INPUT = os.path.join('..', '..', 'files', 'input.txt')  
PATH\_OUTPUT = os.path.join('..', '..', 'files', 'output.txt')  
task\_numb = 7  
  
def heapify(arr, n, i):  
 largest = i  
 left = 2 \* i + 1  
 right = 2 \* i + 2  
  
 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]  
 heapify(arr, n, largest)  
  
def heap\_sort(arr):  
 n = len(arr)  
  
 for i in range(n // 2 - 1, -1, -1):  
 heapify(arr, n, i)  
  
 for i in range(n - 1, 0, -1):  
 arr[i], arr[0] = arr[0], arr[i]  
 heapify(arr, i, 0)  
 return arr  
  
def task1():  
 n, s = inp(PATH\_INPUT)  
 res = str(heap\_sort(s))  
 outp(PATH\_OUTPUT, res)  
 print(caption(task\_numb, res))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start = time.perf\_counter()  
 task1()  
 elapsed\_time = time.perf\_counter() - start  
 print(caption(task\_numb, elapsed\_time))

1. Текстовое объяснение решения.

функция heapify сравнивает текущий узел с его левым и правым потомками, определяет наибольший элемент и, если нужно, меняет местами текущий узел с одним из потомков. Затем heapify вызывается для узла, с которым был произведён обмен. Функция heap\_sort отвечает за выполнение сортировки. Сначала она преобразует исходный массив в макс-кучу, начиная с внутренних узлов дерева и поднимаясь вверх. Затем она постепенно извлекает максимальные элементы и перемещает их в конец массива, после чего вызывает heapify для оставшейся части массива. Процесс продолжается до тех пор, пока весь массив не будет отсортирован.

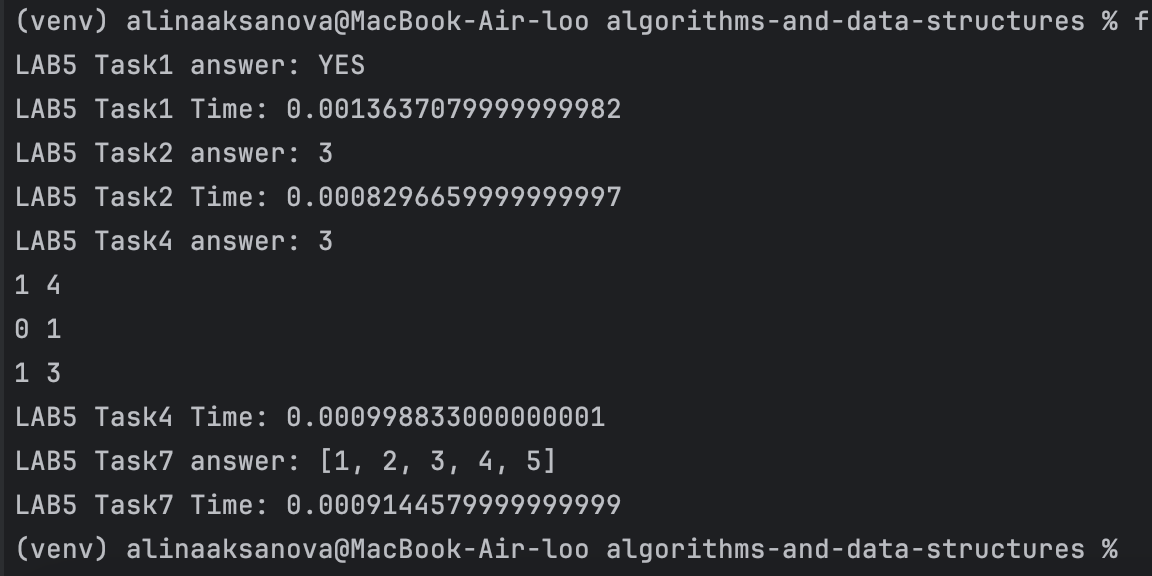
1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я написала алгоритм сортировки.

## Результат работы всех задач

****

# Вывод

В результате лабораторной я написала алгоритм сортировки, алгоритм построения пирамиды и подсчета длинны дерева.