1 Структуры данных

1.1 Куча

1. Реализовать функции работы с кучей:

```
def heapify( heap, position_of_modified_item ):
  ...
def push( heap, item ):
  ...
def getMax( heap ):
  ...
```

heap.py

heapify восстановление основного свойства кучи после изменения в элементе на указанной позиции;

push добавление элемента в кучу;

getMax возвращение максимального элемента из кучи.

2. Воспользуйтесь готовым решением *python*: функциями *heapify*, *heappush*, *heappop* из модуля *heapq* для решения задачи 754. *heapq* (созвучно с *heap queue*) — модуль, реализующий очередь с приоритетами на основе бинарной кучи.

```
from heapq import heappush, heappop
heap = []
data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
for item in data:
  heappush( heap, item )
ordered = []
while heap:
  ordered.append( heappop( heap ) )
data.sort()
data == ordered
```

heapq_usage.py

Где:

heapify создание кучи на массиве (вызываем *heapify(list_of_unordered_items)* и на *list_of_unordered_items* будет построена куча. Соответственно, сложность этой операции порядка $O(log_2 N)$;

heappush добавление элемента в кучу;

heappop возвращение максимального элемента из кучи.

Заметьте, стандартные средтства работы с кучей в *python* — реализуют минимизирующую кучу.