Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича»

Факультет иксс

кафедра пиИвт

(СПбГУТ)

ОтчЁт  
ЛАБОРАТОРНАЯ РАБОТА №3

**Разработка программы интроспективной сортировки для чисел, считанных из файла, при помощи стека типа struct вместо массива**

Руководитель,  
старший преподаватель Ерофеев С. А.

подпись, дата

Исполнитель,  
группа ИКПИ-33 Коньков М. Д.

подпись, дата

Постановка задачи

Данная лабораторная работа направлена на реализацию интроспективной сортировки чисел, считанных из файла, с использованием структуры данных стек типа struct. Программа состоит из следующих основных шагов:

**1. Считывание чисел из файла**: Программа открывает файл для чтения и последовательно считывает числа из него. Считанные числа помещаются в стек.

**2. Интроспективная сортировка с использованием стека**: Интроспективная сортировка является усовершенствованной версией быстрой сортировки, которая использует сортировку вставками, когда размер сортируемого массива становится малым. Основной алгоритм состоит из следующих шагов:

Если размер массива меньше определенного порога или глубина рекурсии становится слишком большой, используется сортировка вставками.

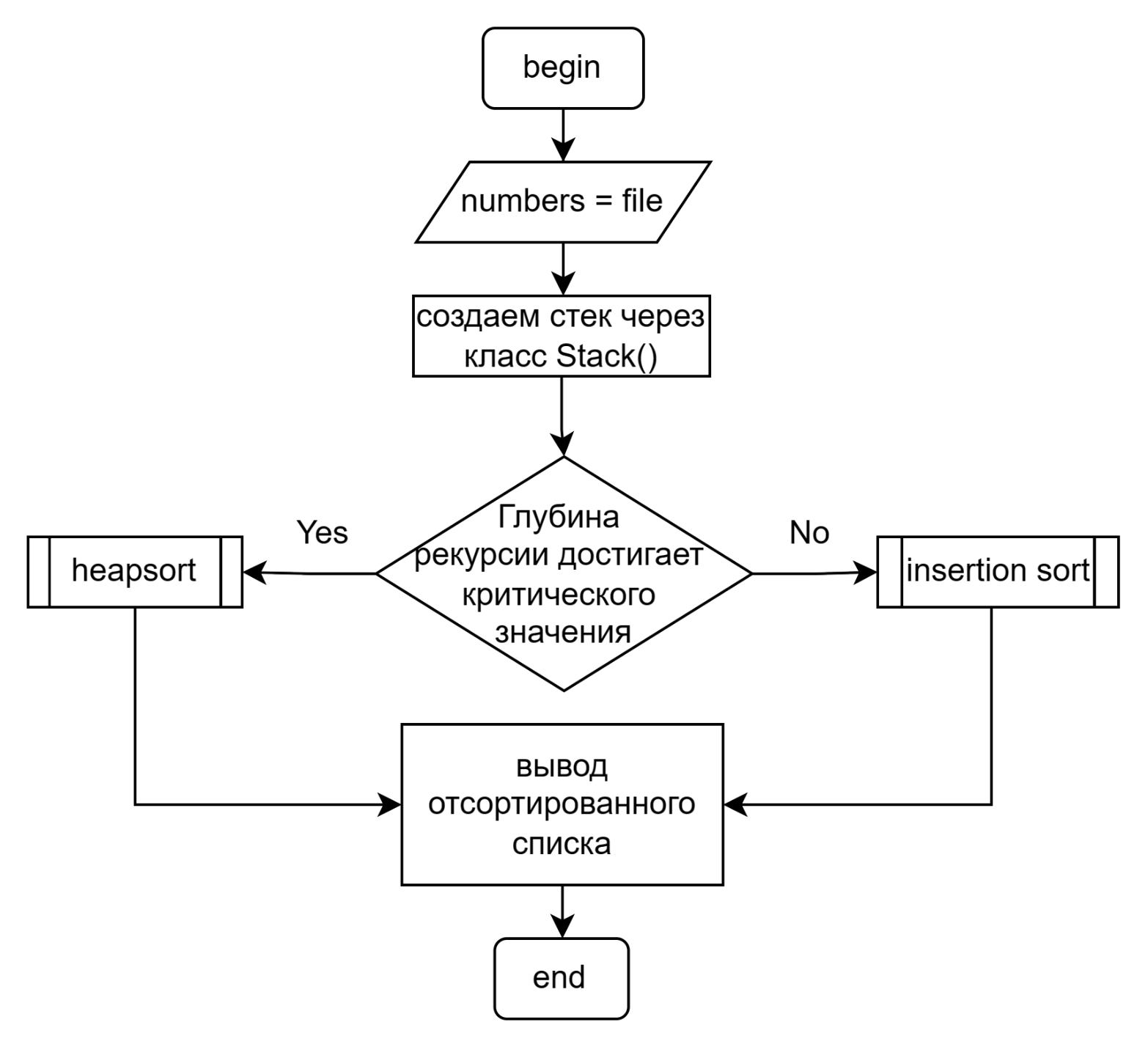
Иначе выбирается опорный элемент, разбивается массив на две части, элементы которых меньше или равны опорному, и элементы, которые больше опорного.

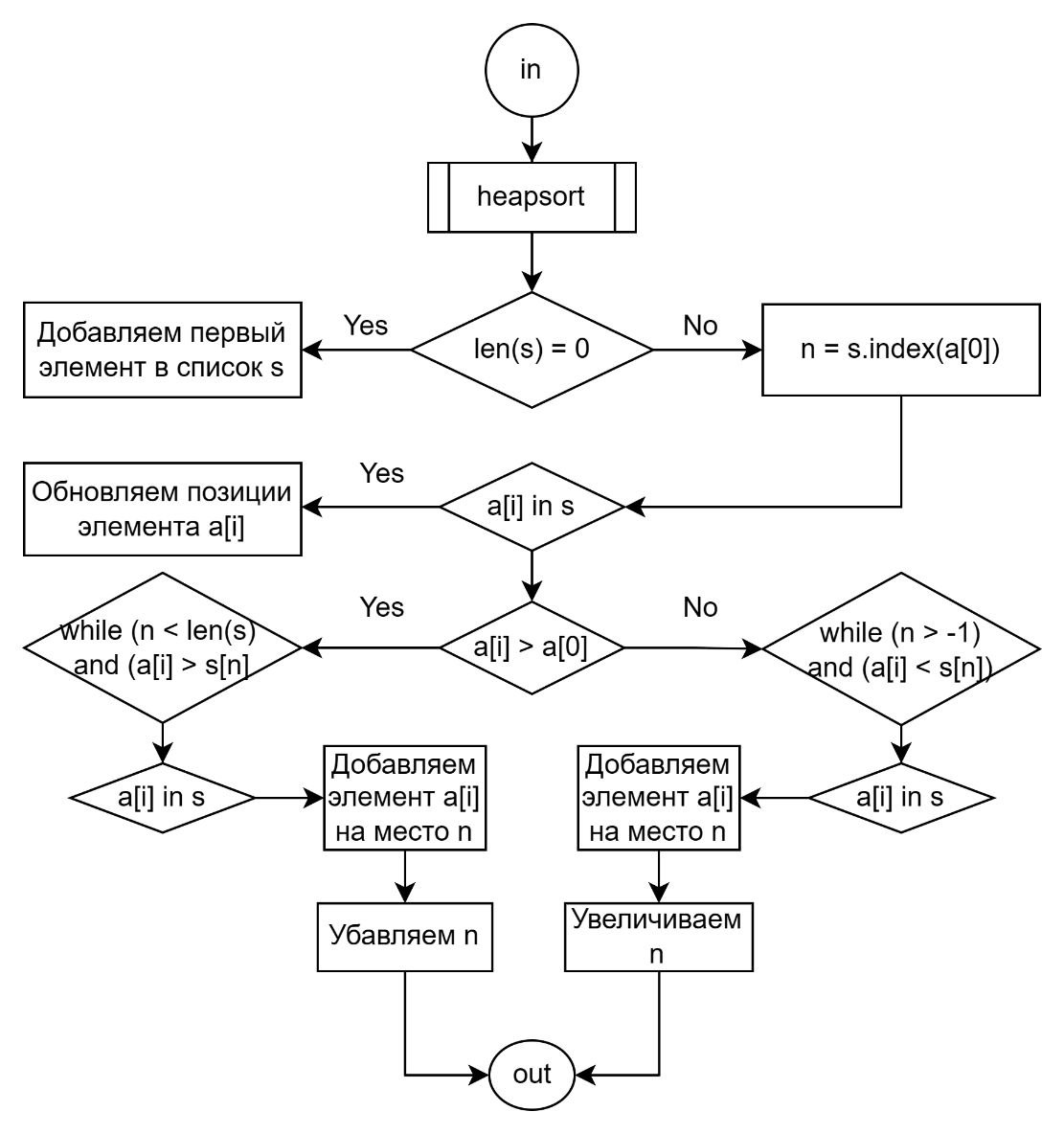
Рекурсивно сортируются обе части массива.

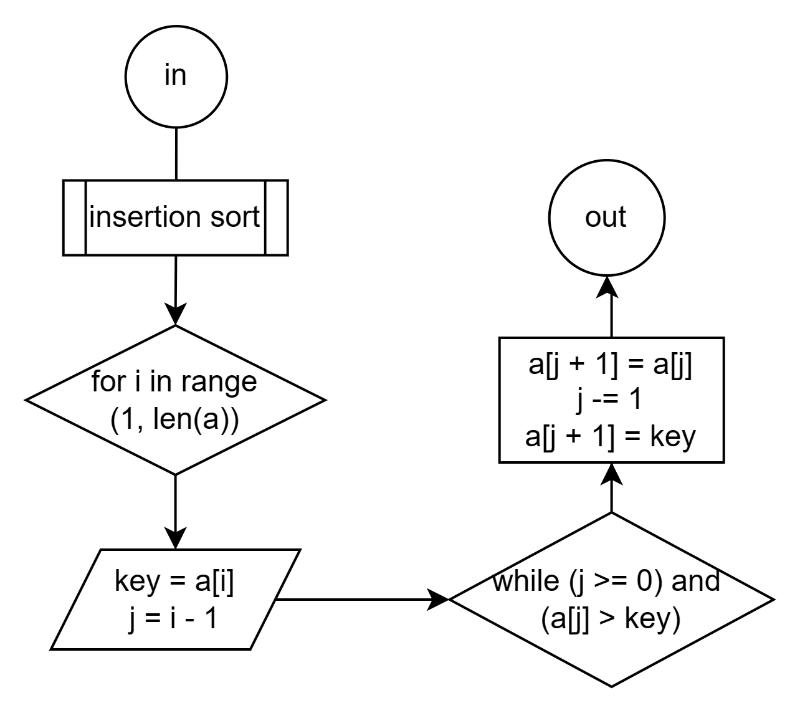
После этого массив становится почти отсортированным, и можно применить сортировку вставками, чтобы завершить сортировку.

**3. Вывод отсортированных чисел**: Отсортированные числа выводятся на экран.

## Общий алгоритм решения:

****

**Алгоритм пирамидальной сортировки:**

**Алгоритм сортировки вставками:**

**Пирамидальная сортировка:**

Пирамидальная сортировка - это эффективный алгоритм сортировки, который использует структуру данных "куча" для упорядочивания элементов. Он состоит из двух этапов: построение кучи и собственно сортировка.

Давайте рассмотрим простой массив [5, 2, 9, 1, 7, 6] и опишем шаги пирамидальной сортировки:

1. Построение кучи (Heapify):

* Преобразуем массив в кучу. Начнем с середины массива и будем проводить процедуру просеивания вниз (heapify-down) для каждого элемента.
* Начнем с последнего индекса, который имеет дочерние узлы (в данном случае - 2 и 3).
* Процедура heapify-down переносит текущий элемент вниз по куче до тех пор, пока не будет удовлетворено свойство кучи (в данном случае, максимальная куча). Это означает, что дочерние элементы всегда меньше родительского.
* После первого прохода по массиву, у нас будет корректно построенная куча.

1. Сортировка:

* После построения кучи, самый большой элемент будет находиться в корне кучи.
* Мы будем поочередно извлекать максимальный элемент из кучи и помещать его в конец массива.
* Затем уменьшаем размер кучи на 1 и восстанавливаем свойство кучи, вызвав heapify-down на оставшейся части кучи (то есть на неотсортированной части массива).
* После каждой итерации максимальный элемент будет перемещаться в конец массива.
* Повторяем этот процесс до тех пор, пока куча не опустеет.

Процесс сортировки для нашего массива будет выглядеть следующим образом:

Исходный массив: [5, 2, 9, 1, 7, 6]

1. Построение кучи: [9, 7, 6, 1, 2, 5]
2. Сортировка:

* Извлечение максимального элемента (9) и помещение его в конец: [5, 2, 6, 1, 7, 9]
* Уменьшение размера кучи: [7, 5, 6, 1, 2]
* Переупорядочивание кучи: [7, 5, 6, 1, 2] (heapify-down)
* Извлечение максимального элемента (7) и помещение его в конец: [2, 5, 6, 1, 7]
* Уменьшение размера кучи: [6, 5, 2, 1]
* Переупорядочивание кучи: [6, 5, 2, 1] (heapify-down)
* Извлечение максимального элемента (6) и помещение его в конец: [1, 5, 2, 6]
* Уменьшение размера кучи: [5, 1, 2]
* Переупорядочивание кучи: [5, 1, 2] (heapify-down)
* Извлечение максимального элемента (5) и помещение его в конец: [1, 2, 5]
* Уменьшение размера кучи: [2, 1]
* Переупорядочивание кучи: [2, 1] (heapify-down)
* Извлечение максимального элемента (2) и помещение его в конец: [1, 2]
* Уменьшение размера кучи: [1]
* Переупорядочивание кучи: [1] (heapify-down)
* Извлечение максимального элемента (1) и помещение его в конец: [1]
* Отсортированный массив: [1, 2, 5, 6, 7, 9]

**Быстрая сортировка:**

Алгоритм быстрой сортировки (или quicksort) - это один из наиболее эффективных алгоритмов сортировки. Он основан на стратегии "разделяй и властвуй". Вот как он работает на примере массива:

Предположим, у вас есть массив чисел:

[7, 2, 1, 6, 8, 5, 3, 4]

1. **Выбор опорного элемента**

Выбираем опорный элемент. В данном случае, для простоты, возьмём последний элемент - 4.

2. **Разделение массива**

Проходим по массиву и перемещаем элементы так, чтобы все элементы, меньшие опорного, оказались слева от него, а все элементы, большие или равные опорному, справа от него. В нашем случае, после разделения, массив может выглядеть так:

[2, 1, 3, 4, 8, 5, 7, 6]

3. **Рекурсивная сортировка**

Теперь рекурсивно применяем быструю сортировку к двум подмассивам, левому и правому от опорного элемента. Для каждого из них повторяе шаги 1 и 2.

Для левого подмассива (меньшие элементы):

[2, 1, 3]

Выбираем опорный элемент (последний) - 3. Разделяем массив:

[1, 2, 3]

Для правого подмассива (большие элементы):

[8, 5, 7, 6]

Выбираем опорный элемент (последний) - 6. Разделяем массив:

[5, 6, 7, 8]

4. **Объединение**

После завершения рекурсии объединяем отсортированные подмассивы. Теперь весь массив отсортирован: [1, 2, 3, 4, 5, 6, 7, 8]

**Сортировка вставками:**

Сортировка вставками используется , когда количество элементов небольшое . Это также может быть полезно, когда входной массив почти отсортирован и в большом массиве потеряно лишь несколько элементов.

Исходный массив: [5, 2, 9, 1, 7, 6]

**Шаги алгоритма:**

1. Начинаем со второго элемента (индекс 1) массива. Сравниваем его со всеми предыдущими элементами в упорядоченной части массива и перемещаемся влево, пока не найдем подходящее место для вставки.

[2, 5, 9, 1, 7, 6]

2. Следующий элемент - 9. Сравниваем его с предыдущими элементами в упорядоченной части массива и вставляем его в нужное место.

[2, 5, 9, 1, 7, 6]

3. Элемент 1 меньше всех элементов в упорядоченной части массива. Мы перемещаемся влево и вставляем его на начало упорядоченной части.

[1, 2, 5, 9, 7, 6]

4. Элемент 7 вставляется на свое место в упорядоченной части массива.

[1, 2, 5, 7, 9, 6]

5. Элемент 6 также вставляется на свое место в упорядоченной части массива.

Отсортированный массив: [1, 2, 5, 6, 7, 9]

**Оценка сложности сортировок:**

**Быстрая сортировка (quicksort) -** в среднем: O(n log n)

**Пирамидальная сортировка (heapsort) -** Всегда O(n log n)

**Сортировка вставками (insertsort)** - Наихудшая (и средняя) сложность алгоритма сортировки вставками равна O(n²)

Время выполнения программы для 100, 1000 и 10000 элементов соответственно (по результатам поиска среднего арифметического из 3 замеров: 0, 110 с., 0,145 с., 2.7 с.

T(N) – полином, описывающий время выполнения программы

O(N) – считывание чисел в стек (в начале), вывод отсортированных чисел (в конце)

O(N^2) – время выполнения сортировки вставками

O(N\*logN) – время выполнения сортировки кучей

O(NlogN) – время выполнения быстрой сортировки

**Сортировка вставками: (N <= 10)**

T(N) = O(N) \*(O(N^2) + O(Nlog N)) + O(N)

**Сортировка кучей: (N > 10)**

T(N) = O(N) \* (O(N\*log N) + O(Nlog N)) + O(N)

Поэтому для файла со 100, 1000, 10000 чисел будет применяться пирамидальная сортировка:

N(100) = 100 \* (100 \* log100 + 100 \* log100) + 100 = 1020 операций

N(1000) = 13900 операций

N(10000) = 194000 операций

Зависимость кол-ва операций от времени: N/O(N)

**Итоговый полином:**

T(N)=O(N)×(0,2​×O(N^2)+​O(NlogN)+0,8​×O(NlogN))+O(N),

где 0,2 – вероятность применения сортировки вставками, а 0,8 – кучей. Числа эти выражаются ассимптотической сложностью, где сортировка вставками будет применятся намного реже.

**Переменные и их назначение:**

* class StackNode - класс, представляющий элемент стека. Содержит поле для хранения данных и ссылку на следующий элемент.
* root - Ссылка на вершину стека.
* data - Хранит числовое значение элемента стека.
* num - Переменная, используемая для считывания чисел из файла.

**Диапазоны переменных:**

**data**: Диапазон значений зависит от типа данных, используемого для хранения чисел. Обычно это целые числа, поэтому диапазон будет ограничен значением INT\_MIN до INT\_MAX, где INT\_MIN и INT\_MAX определены в модуле ‘sys’.

**num**: Диапазон значений также зависит от типа данных. Однако он определяется числами, считываемыми из файла.

## Тестирование:

## Sorted numbers: [0, 1, 1, 6, 7, 7, 7, 3, 6, 3, 2, 7, 28, 8, 17, 24, 26, 9, 8, 26, 26, 32, 53, 52, 53, 57, 73, 75, 63, 63, 73, 70, 76, 82, 84, 85, 82, 85, 96, 338, 236, 282, 387, 574, 393, 5724, 28284, 843, 534, 523, 626727] --- [Finished in 110ms]

**Исходный файл numbers.txt:** 32, 1, 2, 3, 1, 26, 24, 82, 626727, 523, 82, 17, 8, 393, 28, 236, 843, 6, 3, 7, 75, 7, 574, 73, 7, 85, 85, 6, 26, 26, 96, 0, 7, 70, 9, 84, 73, 63, 282, 28284, 63, 52, 338, 53, 8, 76, 387, 53, 57, 5724, 534

## Листинг программы introsort.py (Python 3)

# Insertion sorting function

def insertion\_sort(arr):

for i in range(1, len(arr)):

key = arr[i]

j = i - 1

while j >= 0 and arr[j] > key:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

# Stackelement (contains data and link to next element)

class StackNode:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

# Checking if stack is empty

def is\_empty(root):

return not root

# Adding new element in root

def push(root, data):

stack\_node = StackNode(data)

stack\_node.next = root

root = stack\_node

return root

# Removing element from root and changing on another one

def pop(root):

if is\_empty(root):

return None, -1

temp = root

root = root.next

popped = temp.data

return root, popped

# Quicksort partition

def partition(arr, low, high):

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] <= pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

# Introsort (partition)

def introsort(arr, low, high, max\_depth):

if low < high:

if high - low + 1 <= 10:

insertion\_sort(arr[low:high+1])

elif max\_depth == 0:

arr = heapsort(arr, low, high)

return arr

else:

pivot\_index = partition(arr, low, high)

introsort(arr, low, pivot\_index - 1, max\_depth - 1)

introsort(arr, pivot\_index + 1, high, max\_depth - 1)

# Pyramidal sorting function (heapsort)

def heapsort(arr, low, high):

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)

n = high - low + 1

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

# Introsort function

def introspective\_sort(stack):

arr = []

while not is\_empty(stack):

stack, num = pop(stack)

arr.append(num)

introsort(arr, 0, len(arr) - 1, 2 \* len(arr))

return arr

if \_\_name\_\_ == "\_\_main\_\_":

try:

# Oppening file

with open("numbers.txt", "r") as file:

stack = None

# Replacing numbers from file to stack

for line in file:

num = int(line.strip())

stack = push(stack, num)

except FileNotFoundError:

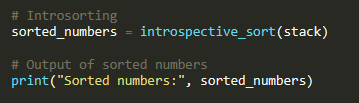
print("Error while openning file.")

exit(1)

# Introsorting

sorted\_numbers = introspective\_sort(stack)

# Output of sorted numbers

print("Sorted numbers:", sorted\_numbers)

**Вывод:**

В ходе выполнения данной лабораторной работы был написан алгоритм интроспективной сортировки на языке Python в среде разработки Sublime Text 3. Было выяснено, что интроспективная сортировка включает в себя выполнение трёх подсортировок: быстрой (для разбиения массива на блоки), вставками (если размер массива меньше числа n = 16) и кучей (в иных случаях). Массив считывал числа из файла numbers.txt. В прорамме использовался стек типа struct. Также оценена сложность выполнения алгоритма. Функциональные задачи, поставленные перед программой, были успешно выполнены.