ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

**Отчет по лабораторной работе дисциплины**

**«Алгоритмы и анализ сложности»**

**Лабораторная работа № 2**

**Алгоритмы сортировки строк**

***Вариант 6***

***Пирамидальная сортировка HeapTree***

Студент: Новгородов Дмитрий Андреевич,

РИ-290018, АТ-14

Преподаватель: Трофимов Сергей Павлович,

кандидат физико-математических наук, доцент

2021

**Оглавление**

Задание………………………………………………………………………….2

Теоретическая часть……………………………………………………………3

*- Задание 1* …………………………………………………………………...…5

*- Задание 2* ………………………………………………………...………...….8

Инструкция для пользователя………………………………………………...12

Инструкция для программиста ……………………………………………….13

Тестирование………………………………………………………...…………14

Выводы……..………………………………………………………...…………15

Литература ...………………………………………………………...…………16

Приложение ...……………………………………………..………...…………17

**Задание**

**Лабораторная работа №2\***

**Алгоритмы сортировки строк**

1 задание

Реализовать алгоритмы пирамидальной сортировки строк HeapTree.

2 задание

Определить порядок сложности этих алгоритмов относительно операций сравнения, а также коэффициенты пропорциональности: F(N) = C\*N\*log2(N). Для аппроксимации можно использовать метод наименьших квадратов и сервис «Поиск решения».

**Теоретическая часть**

Пирамидальная сортировка (или сортировка кучей, HeapSort) — это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча. Она похожа на сортировку выбором, где мы сначала ищем максимальный элемент и помещаем его в конец. Далее мы повторяем ту же операцию для оставшихся элементов.

Давайте сначала определим законченное двоичное дерево. Законченное двоичное дерево — это двоичное дерево, в котором каждый уровень, за исключением, возможно, последнего, имеет полный набор узлов, и все листья расположены как можно левее.

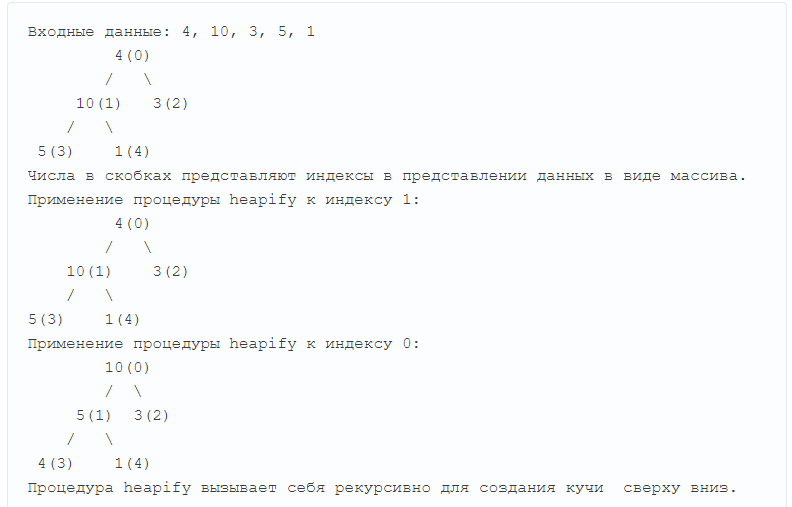
Двоичная куча — это законченное двоичное дерево, в котором элементы хранятся в особом порядке: значение в родительском узле больше (или меньше) значений в его двух дочерних узлах. Первый вариант называется max-heap, а второй — min-heap. Куча может быть представлена двоичным деревом или массивом.

Поскольку двоичная куча — это законченное двоичное дерево, ее можно легко представить в виде массива, а представление на основе массива является эффективным с точки зрения расхода памяти. Если родительский узел хранится в индексе I, левый дочерний элемент может быть вычислен как 2 I + 1, а правый дочерний элемент — как 2 I + 2 (при условии, что индексирование начинается с 0).

Алгоритм пирамидальной сортировки в порядке по возрастанию:

1. Постройте max-heap из входных данных.
2. На данном этапе самый большой элемент хранится в корне кучи. Замените его на последний элемент кучи, а затем уменьшите ее размер на 1. Наконец, преобразуйте полученное дерево в max-heap с новым корнем.
3. Повторяйте вышеуказанные шаги, пока размер кучи больше 1.

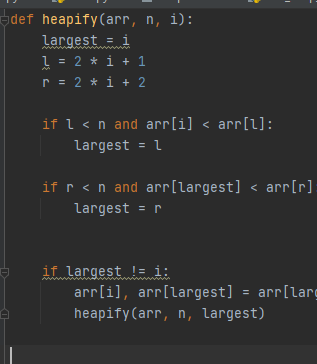
Процедура преобразования в кучу (далее процедура heapify) может быть применена к узлу, только если его дочерние узлы уже преобразованы. Таким образом, преобразование должно выполняться снизу вверх. Давайте разберемся с помощью примера:

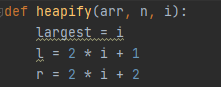


***Задание 1***

Реализация алгоритма состоит из двух этапов. Преобразования необходимого узла в двоичную кучу поддерева и создание основной функции для сортировки массива заданного размера.

Сначала напишем преобразование в двоичную кучу с корневым узлом i, что является индексом в arr[]. n - размер кучи.





Проверяем существует ли левый дочерний элемент больший, чем корень



Проверяем существует ли правый дочерний элемент больший, чем корень



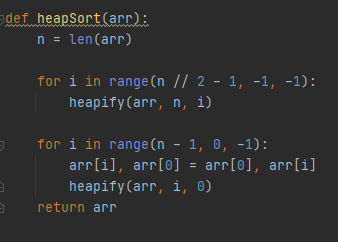
Заменяем корень, если нужно



Применяем heapify к корню.



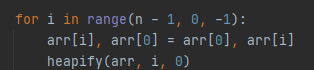
Теперь можно написать основную функцию для сортировки массива заданного размера



Построение max-heap.



Один за другим извлекаем элементы



***2 задание***

*Исследование сложности алгоритма на массовой задаче.*

План исследования алгоритма:

<experiments>

<experiment name=”Tree”>

<nodes name=”Arithmetic Progression” minElement=0 maxElement=150 startLength=0 diff=1 maxLength=150 repeat=1>

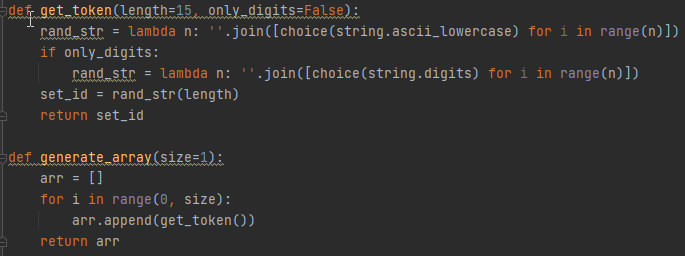
</nodes>

</experiment>

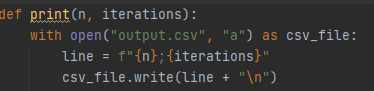
</experiments>

Для исследования сложности алгоритма было реализовано несколько методов, чтобы в автоматическом режиме записывать результаты исследований и генерировать исходные данные.

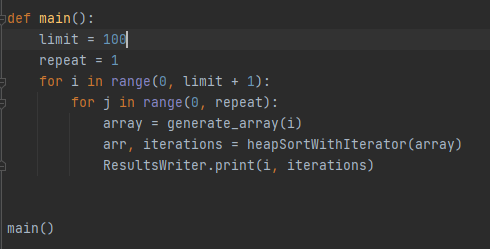
Реализация методов для генерации данных



Реализация метода для записи результатов исследований



Реализация метода для проведения исследования



Данные исследований были экспортированы в csv и загружены в Excel. Визуализация данных с помощью Excel:

Так как алгоритм HeapSort относится к классу оптимальных, необходимо найти аппроксимацию получившегося графика по исходным данным. Для нахождения аппроксимации использовался способ с построением линии тренда на точечном графике

· Найденная аппроксимация: y = 0.0116x2 + 5.3536x - 25.99,

· Величина достоверности: R² = 0.9992



**Инструкция для пользователя**

Для реализации проекты был выбрал язык Python. Он разделен на 2 части + тесты

1 часть – Реализация алгоритма HeapTree.

2 часть – Реализация алгоритма HeapTree с количеством итераций.

- Если вы хотите повторить эксперимент, то

1. Удалите весь текст в output.csv
2. Запустите в вашей среде файл main.py
3. Посмотрите ваш результат сортировки в output.csv

Примечание: для запуска файла необходим python не ниже версии 3.7, установленный на компьютер.

**Инструкция для программиста**

Для реализации проекты был выбрал язык Python версии выше 3.7 и библиотеку Unittest. Он разделен на 2 части + тесты

1 часть – Реализация алгоритма HeapTree - sort\_heap\_tree.py

2 часть – Реализация алгоритма HeapTree с количеством итераций - sort\_heap\_tree\_iter.py

Тесты – tests.py

Файл sort\_tree\_with\_iterator.py дублирует алгоритм из файла sort\_tree.py. Единственное его отличие, что вместо отсортированного массива он возвращает кортеж массив + количество операций. Рекомендую в будущем каким-либо образом интегрировать подсчет кол-ва операций в сам алгоритм сортировки.

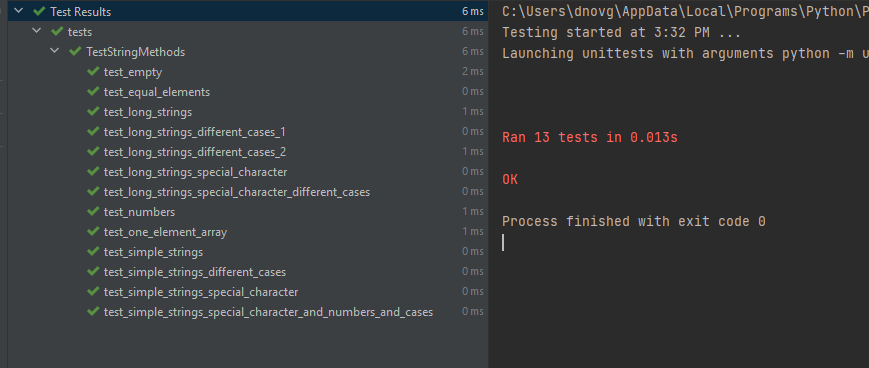
При изменении кода необходимо проверять работоспособность на тестах. Они находятся в файле tests.py

**Тестирование**

Были составлены несколько десятков тестов для алгоритма HeapTree. На каждую операцию был написан тест, закрывающий недопустимые значения и возможные ошибки.

Для тестов использована библиотека unittest. Реализованный набор тестов:

Программа прошла все тесты успешно.



**Выводы**

В ходе лабораторной работы был реализован алгоритм HeapSort и произведена оценка его эффективности, в результате чего удалось построить аппроксимацию графика сложности алгоритма с R² = 0.9992 и аппроксимацией y = 0.0116x2 + 5.3536x - 25.99. Фактические значение совпали с теоретическими.

Над реализованным алгоритмом возможно провести дополнительные улучшения, чтобы снизить потребление памяти, добавить дополнительные параметры сортировки, сделать его Generic классом.

**Литература**

1. Сайт <https://www.geeksforgeeks.org/python-program-for-heap-sort/>
2. Сайт <https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0>
3. Сайт <https://habr.com/ru/company/otus/blog/460087/>

**Приложение**

Ссылка на код лабораторной работы - https://github.com/Red-Stalker/heapsort