Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий и систем связи

Лабораторная работа №4

Вариант №1

Выполнил(и:)

Алексеев Т.Ю.

Оншин Д.Н.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы являлось знакомство с различными алгоритмами сортировки: быстрой, расчёской, пирамидальным и блочным.

Для достижения данной цели необходимо было выполнить следующие задания:

1. Задание 1: написание программы с функциями быстрой сортировки и сортировки расчёской. Использование этой программы, как модуля, в другой программе, где пользователь выбирает желаемый алгоритм. Необходимо посчитать затраченное время с помощью timeit.
2. Задание 2: реализация алгоритмов блочной и пирамидальной сортировок.
3. Задание 3: оценка сложности, достоинства и недостатков изученных алгоритмов.

Решения данных задач будут находиться на GitHub по ссылке: https://github.com/NorthPole0499/Algoritms_task_4

1. ЗАДАНИЕ 1

Для начала, в данном задании необходимо реализовать два алгоритма: быстрой сортировки и сортировки расчёской.

Суть быстрой сортировки состоит в выборе опорного элемента, относительно которого все числа делятся на три списка: большие опорного, меньшие и равные. После этого выполняется рекурсия, выполняется то же самое для списка меньших и больших элементов. В конце алгоритма, списки склеиваются.

Основная идея сортировки расчёской состоит в сравнении элементов, находящемся на определённом расстоянии друг от друга. Постепенно это расстояние сокращается. Сделано это для того, чтобы наиболее маленькие элементы как можно быстрее переместить в начало списка. Сравнение элементов происходит, пока расстояние не будет равно 0, то есть не будут сравниваться соседние элементы.

Таким образом, у нас есть программа task_1_part1.py с двумя алгоритмами сортировки.

Используем полученную программу, как модуль, в файле task_2_part2.py, который непосредственно работает с пользователем, спрашивая желаемый алгоритм и выводя время, посчитанное с помощью модуля timeit. На приложении снизу можно увидеть пример того, как происходит взаимодействие пользователя с программой.

```
1 - quick sort, 2 - comb sort, 0 - exit
Input the sort: 1
Input the list: 5 4 3 2 1
Quick sort: [1.0, 2.0, 3.0, 4.0, 5.0]
Time is 4.38000000002603e-05

1 - quick sort, 2 - comb sort, 0 - exit
Input the sort: 2
Input the list: 5 4 3 2 1
Comb sort: [1.0, 2.0, 3.0, 4.0, 5.0]
Time is 2.9800000000079763e-05

1 - quick sort, 2 - comb sort, 0 - exit
Input the sort: 0
```

Приложение 1 – Пример ввода и вывода программы для Задания 1

2. ЗАДАНИЕ 2

В данном задании необходимо реализовать два алгоритма: блочной и пирамидальной сортировки.

Опишем процесс блочной сортировки. Для начала высчитываем значение переменной znach, необходимой для распределения элементов по контейнерам, путём деления наибольшего элемента списка на его длину. После этого создаём список из пустых списков (контейнеров), число которых равно длине исходного массива. После этого каждый элемент списка мы делим на переменную znach и отнимаем единицу. Полученное значение является номером контейнера, куда необходимо поместить элемент. Когда все элементы будут распределены по соответствующим кейсам, осталось лишь отсортировать все контейнеры и склеить их в один список. Для сортировки контейнеров была выбрана сортировка вставками в виду малого числа элементов в каждом кейсе.

Теперь опишем процесс пирамидальной сортировки. Сортировка основана на построение кучи (дерева) из элементов массива. Причем куча строится по правилу не возрастания, то есть корневой элемент должен быть больше элементов в его ветвях. После того, как построена невозрастающая куча, программа начинает идти по массиву, начиная с последнего элемента, и меняет его с начальным элементом, тем самым меняя кучу. Затем куча строится заново, но в нее не включается максимальный элемент, который уже находится в конце массива. Получается невозрастающая куча и массив, где максимальный элемент находится в конце.

После этого предпоследний элемент меняется с первым и снова строится куча, но уже без двух последних элементов.

Таким образом, максимальные элементы вытесняются из начала кучи в конец массива, и получается отсортированный список.

3. ЗАДАНИЕ 3

Название	Сложность (в худшем случае)	Сложность (в лучшем случае)	Плюсы	Минусы
Быстрая сортировка	$O(n^2)$	O(n * logn)	-Один из самых быстрых алгоритмов сортировки -Алгоритм прост в реализации и понимании	-Быстро деградирует до худшего времени
Сортировка расчёской	$O(n^2)$	O(n * logn)	-Достаточно мало операций по перестановке элементов. Из-за этого малое время.	-Скорее всего, неустойчива, возможно переполнение стека
Блочная сортировка	$O(n^2)$	O(n)	-Очень быстра на удачных данных	-При небольшом отличии элементов друг от друга, становится очень плохой
Пирамидальная сортировка	O(n * logn)	O(n * logn)	-Имеет очень устойчивую сложность, которая почти не зависит от входных данных	-Из-за устойчивой сложности чересчур долго сортирует почти отсортированные списки -Сортировка неустойчива, при огромных значениях возможно переполнение стека

ЗАКЛЮЧЕНИЕ

Таким образом, в ходе выполнения данной лабораторной работы были достигнуты все поставленные цели. Произошло знакомство с 4 алгоритмами сортировки: быстрая, расчёской, блочная и пирамидальная. Также, были выявлены плюсы и минусы каждого алгоритма, его сложность.

СПИСОК ЛИТЕРАТУРЫ

1) О блочной сортировки в Python// URL: https://www.internet-technologies.ru/articles/blochnaya-sortirovka-v-python.html (дата обращения: 01.12.2022)