Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

**Лабораторная работа №1**

Выполнили:

Мануковская Д. М.

Сакулин И. М.

Сафронов И. С.

Проверил:

Мусаев А. А.

Санкт-Петербург,

2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc178097445)

[Задание 1 4](#_Toc178097446)

[Задание 2 6](#_Toc178097447)

[Задание 3 10](#_Toc178097448)

[ЗАКЛЮЧЕНИЕ 11](#_Toc178097449)

ВВЕДЕНИЕ

В данной лабораторной работе необходимо построить зависимость между количеством элементов и количеством шагов для 4-х алгоритмов со сложностями О(1), O(logn), O(n^2), O(2^n), реализовать программу пузырьковой сортировки и сравнить её с методом sort(), а также написать 5 алгоритмов, имеющих соответсвенные сложности O(3n), O(nlogn), O(n!), O(n3), O(3log(n)).

Цель работы: знакомство с вычислительной сложностью алгоритмов.

Задание 1

Построим зависимости количества элементов и количества шагов алгоритмов в виде графиков.

В таблице 1 указаны значения количества элементов и количества шагов для алгоритма со сложностью О(1).

Таблица 1 – Расчетные данные для графика алгоритма сложностью О(1)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| O(1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Построим график 1 по этим значениям. Горизонтальная ось будет соответствовать значениям n, вертикальная – О(1).

График 1 – Зависимость для алгоритма сложностью O(1)

График имеет вид прямой параллельной горизонтальной оси, т. е. сложность алгоритма не зависит от количества элементов и является константной.

Составим таблицу 2 для значений количества элементов и количества шагов для алгоритма сложностью O(logn).

Таблица 2 – Данные для дальнейшего построения графика алгоритма со сложностью O(logn)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| O(logn) | 3 | 4 | 4,58 | 5 | 5,32 | 5,58 | 5,8 | 6 | 6,17 | 6,32 |

По этим значениям выстроим график 2. На вертикальной оси будут значения O(logn), а на горизонтальной – n.

График 2 – Зависимость для алгоритма со сложностью O(logn)

График имеет вид возрастающей логарифмической кривой.

Построим таблицу 3 для значений количества элементов и количества шагов для алгоритма со сложностью О(2^n).

Таблица 3 – Расчетные данные для графика алгоритма сложностью О(2^n)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| O(2^n) | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |

На графике 3, составленном по соответствующей таблице, представлена зависимость количества элементов и шагов для данного алгоритма.

График 3 – Зависимость для алгоритма со сложностью О(2^n)

По графику видно, что зависимость экспоненциальная.

Составим таблицу 4 для количества элементов и шагов алгоритма сложностью O(n^2).

Таблица 4 – Расчетные данные для графика алгоритма сложностью О(n^2)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| O(n^2) | 1 | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 | 100 |

По этим значениям составим график 4. Горизонтальная ось – это значения n, вертикальная – O(n^2).

График 4 – Зависимость для алгоритма со сложностью О(n^2)

График 4 имеет вид одной ветви параболы, направленной вверх.

По этим 4м зависимостям можно сделать вывод, что наименьшей вычислительной сложностью обладают алгоритмы с верхней оценкой трудоемкости O(1), O(logn), а наибольшей – О(2^n), O(n^2). Это значит, что при реализации алгоритмов стоит стремится к тому, чтобы вычислительная сложность была O(1) или O(logn).

Задание 2

Для начала необходимо написать сам алгоритм сортировки рисунок 1:

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Рисунок 1- Сортировка пузырьком

Основная идея в том, что проходясь по списку, мы сравниваем два соседних элемента, размещая правее тот, что больше. Сложность данном алгоритма – O(n2).

Встроенный в Python метод sort() использует способ сортировки слиянием.

Сравнение алгоритмов (график 5):

График 5 – Сравнение сортировок

По графику видно, что оптимальнее использовать сортировку слиянием, то есть используя встроенный метод sort().

Задание 3

Придумать и реализовать алгоритмы, имеющие сложность O(3n), O(nlogn), O(n!), O(n3), O(3log(n)).

Ввиду того, что константа перед переменной обычно упускается, а идеальную сложность сделать очень трудно, алгоритмы сложностью O(3n) и O(3log(n)) были написаны приближенно, но не точно к константе.

1. O(n) – Нахождение длины самой большой возрастающей последовательности.

Это переборный алгоритм (функция find\_up на рисунке 2), потому что он проходит по каждому из n элементов массива ровно один раз, отсюда и получается линейная сложность. На каждой итерации 3 операции: так получается приближение к константе.

На вход подаётся массив чисел, на выходе искомый результат. Для начала функция создаёт 2 переменные для хранения текущего (length) и максимального (max\_length) результатов поиска. Затем, итерируя по списку, при выполнении условия возрастания текущий результат увеличивается, при невыполнении – сбрасывается до единицы. Совместно с этим переменная max\_length сохраняет максимальное из всех значений, принимаемых length, которое и будет результатом работы алгоритма.

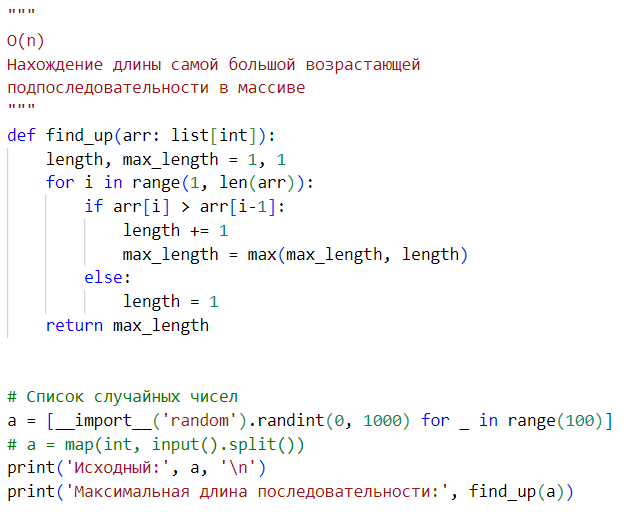


Рисунок 2 − Алгоритм O(n)

1. O(n\*log(n)) – Быстрая сортировка

Со сложностью O(n\*log(n)) обычно ассоциируются оптимальные алгоритмы сортировки. Один из них представлен на рисунке 3. На вход функции quick\_sort неотсортированный список, а её результатом является отсортированный.

Функция рекурсивна. Сначала она выбирает один из элементов, затем отбирает все элементы больше и меньше его в 2 массива: один рекурсивно сортирует и добавляет слева от опорного элемента, другой справа. Базовым случаем является список из 0 или 1 элемента, так как их сортировать не нужно.

Сложность является таковой, потому что сначала происходит log(n) итераций в глубину, так как список каждый раз делиться на 2 части, а на каждом слое итерации происходит полный перебор всех элементов, что является n.

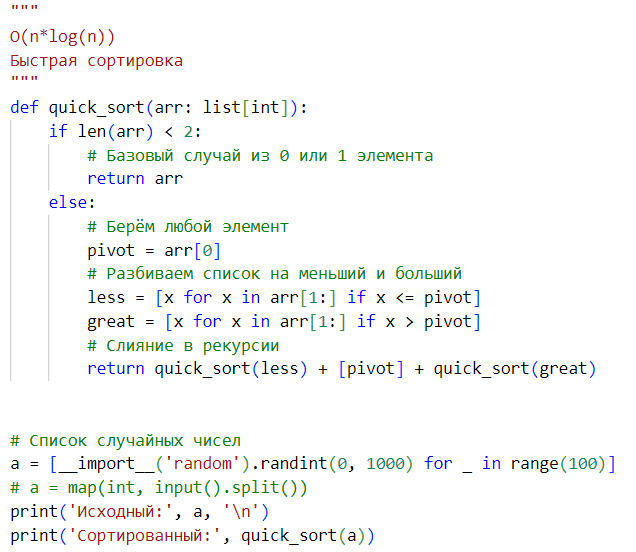


Рисунок 3 – Алгоритм O(n\*log(n))

1. O(n!) – Рекурсивное нахождение всех перестановок

Количество перестановок любых n элементов ровно «n!», поэтому алгоритм, который составляет все такие перестановки будет иметь такую же сложность O(n!).

На вход функции permutations, представленной на рисунке 4, подаётся массив чисел и сразу инициализируемые общие переменные: для хранения создаваемой перестановки (res), всех перестановок (ret), а также для ограничения глубины рекурсии (border). В качестве базового случая выбираем тот, где мы достигли необходимой глубины, здесь сохраняем перестановку. Рекурсивно же мы заходим с каждым по порядку элементом.

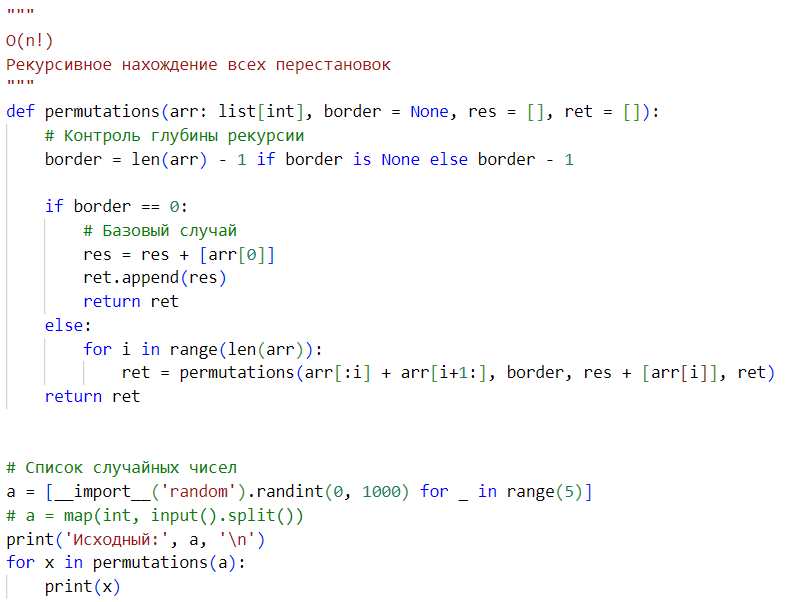


Рисунок 4 – Алгоритм O(n!)

1. O(n3) – Умножение матриц

На рисунке 5 представлен реализованный алгоритм умножения матриц, который работает за n\*n\*n операций, потому что мы создаём матрицу n\*n за такое же количество операций, а затем в каждом таком случае проходим ещё одну итерацию в n элементов.

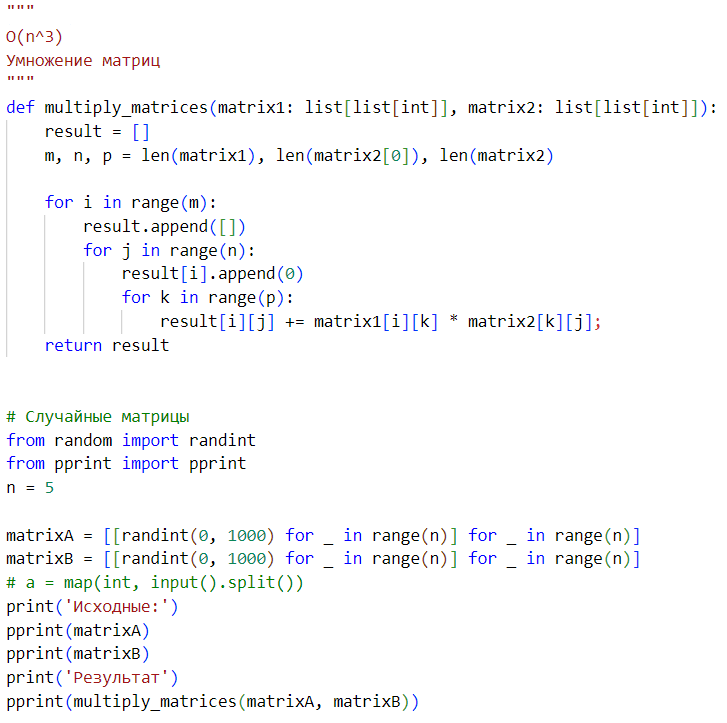


Рисунок 5 − Алгоритм O(n3)

1. O(log(n)) – Бинарный поиск

Бинарный поиск работает за log(n) итераций, в каждой из которых примерно по 3 команды, что позволяет приблизиться к константе.

На входе: отсортированный массив и искомый элемент, на выходе: индекс искомого элемента.

Алгоритм представлен на рисунке 6.

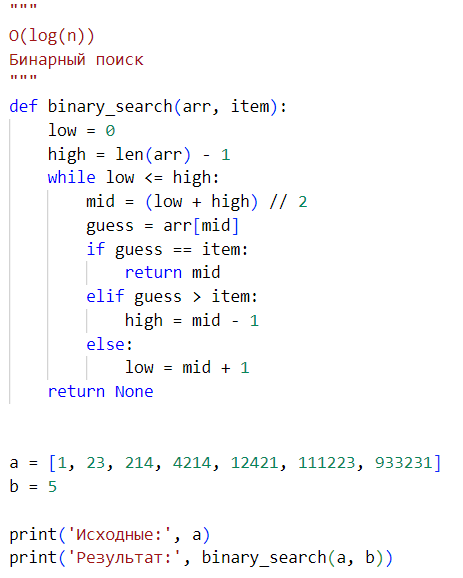


Рисунок 6 – Алгоритм O(log(n))

ЗАКЛЮЧЕНИЕ

Цель работы достигнута. Были построены 4 зависимости-графики для алгоритмов с различными вычислительными сложностями.

2 задание

А также были реализованы 5 графиков с разными асимптотическими сложностями.