1 задание

Для тестирования алгоритмов были созданы вспомогательные функции (рисунок N). Первая – «generate\_lst» принимает границы диапазона и длину, а возвращает список случайных чисел заданной длины. Генерация случайных значений происходит с помощью функции «randint» модуля «random». По умолчанию заданы маленькие параметры для тестов.

Вторая функция «test». На вход к ней подаётся объект функции тестируемой сортировки, а также, опционально, сортируемый список. Если он не задан, то генерируется автоматически. Список сортируется с помощью переданной функции и с помощью встроенной в python «sorted», затем результаты сравниваются. После в терминал выводятся: название алгоритма, корректность его выполнения, изначальный, отсортированный через «sorted» и через переданную функцию сортировки списки.

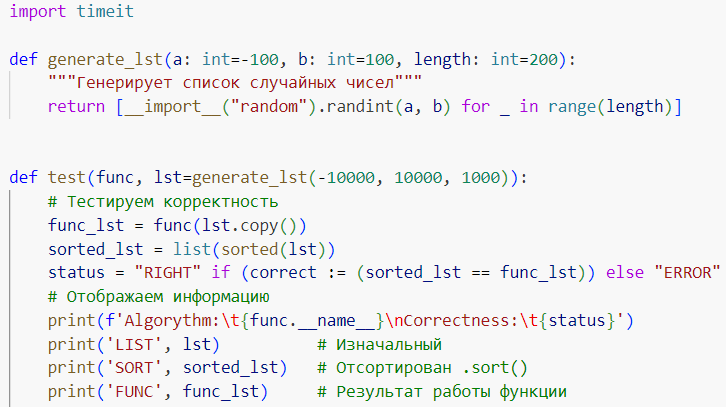


Рисунок N – Генерация списков и тестирование

Во второй части задания необходимо протестировать время исполнения алгоритмов через модуль «timeit». Так как тестирующая функция принимает на вход строку с кодом, для удобства была написан отдельный код, он тестирует все переданные функции через случайные списки, заданного размера и выводит время в терминал (рисунок N+1).



Рисунок N+1 – Тесты с помощью timeit

Быстрая сортировка является одной из самых быстрых сортировок, применяемых на практике, она относится к типу «разделяй и властвуй», когда мы делим большую задачу на маленькие подзадачи, тем самым решая большую задачу. Видоизменённые версии такой сортировки можно обнаружить во многих языках программирования в качестве встроенной.

Основной принцип её работы такой:

1. Проверяется базовый случай, когда список пустой или содержит всего 1 элемент;
2. Поиск индекса опорного элемента;
3. Все элементы делятся на 3 части: меньшие, равные и большие опорному;
4. Элементы меньшие и большие опорного сортируются рекурсивно;
5. Возвращается список, составленный из отсортированных элементов меньше опорного, равных и больше опорного.

Выбор опорного элемента – важный шаг, от которого зависит эффективность алгоритма. Оптимальным может оказаться любой элемент с равной вероятностью. Можно взять медианный от нескольких выбранных элементов, или просто по какому-то индексу, или первый, или применить прочие специальные методы. Если, например, брать первый при отсортированном массиве или такой, что список делиться на пустой и из всех остальных, то сложность достигнет максимальной O(n2).

Сложность:

Средней и оптимальной сложностью является O(n\*logn), потому что алгоритм делит на 2 части список log(n) итераций и на каждом из уровней рекурсии итерируется по n элементам.

В самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и n-1𝑛−1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз [1]. Сложность в таком случае O(n2).

Достоинства:

* Самая быстрая на практике
* Простой принцип действия
* Допускается естественное распараллеливание процессов

Недостатки:

* При неудачных вводных данных сильно падает по скорости
* Зависимость от выбора опорного элемента

Реализация представлена на рисунке N+2.

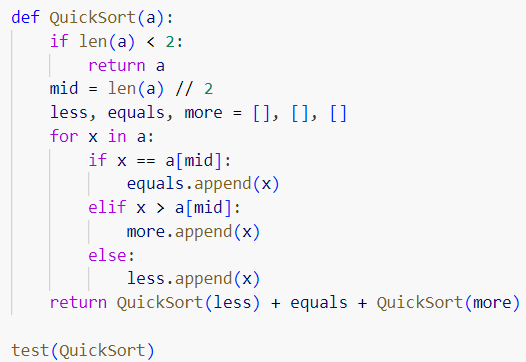


Рисунок N+2 – Быстрая сортировка

Сортировка расчёской – алгоритм, основанный на сравнении двух элементов и смене их мест. Основная идея в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и затем сужать это расстояние вплоть до 1.

Алгоритм делает следующие шаги:

1. Берёт максимальный шаг – расстояние между сравниваемыми элементами – как размер списка
2. Сокращает шаг на фактор уменьшения, оптимальная величина которого примерно 1.25
3. Сравнивает все элементы, находящиеся на расстоянии шага, если больший стоит раньше меньшего, меняет их местами
4. Если изменений за итерацию не было, заканчивает сортировку, иначе переходит к шагу 2.

Сложность:

В худшем случае сортировка произойдёт за O(n^2) операций, в лучшем за O(n\*logn), в среднем же она O(n^2/2^p).

Преимущества:

* простой код;
* стабильная сортировка [2].

Недостатки:

* медленней, чем алгоритмы со средней сложностью O(n\*logn)

На рисунке N+3 представлена реализация этого алгоритма.

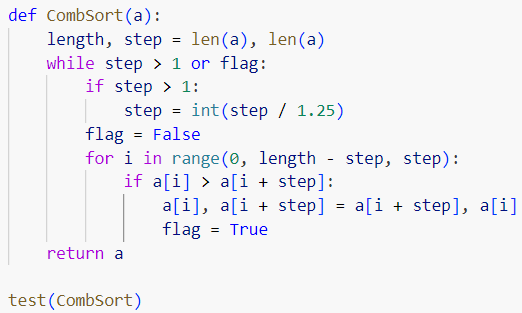


Рисунок N+3 – Сортировка расчёской

Быстрая сортировка и сортировка расчёской были отправлены в timeit, со следующими характеристиками (рисунок N+4). Параметр repeat отвечает за количество повторений: чем больше это значение, тем дольше исполняются тесты, но значения сильнее усредняются. Второй аргумент length отвечает за размер входных данных – количество элементов в сортируемом списке.

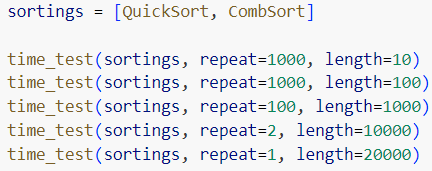


Рисунок N+4 – Запуск тестов

В результате работы тестов были получены замеры времени выполнения (рисунок N+5). На основе этих данных можно сделать выводы о том, что на малых данных скорости разняться не очень сильно, а на вот на больших массивах, разница ощутима заметна. Это были ожидаемые результаты, ведь оценка сложности прогнозировала то же самое.

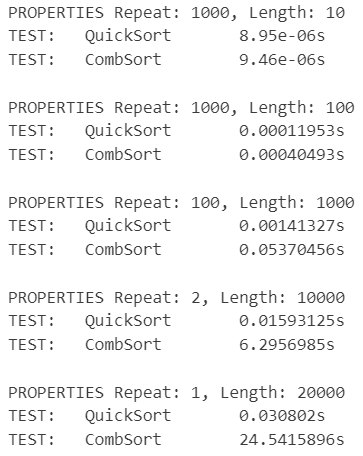


Рисунок N+5 – Результаты тестов

ИСТОЧНИКИ

1. <https://ru.wikipedia.org/wiki/Быстрая_сортировка>
2. <https://sci-article.ru/stat.php?i=1555662521>