Квадратичные сортировки

1 Алгоритм сортировки

Дан массив элементов, расположенных в произвольном порядке. Необходимо упорядочить эти элементы по какому-то признаку. Решение этой задачи – сортировка элементов массива. Соответственно, алгоритм сортировки – алгоритм упорядочения элементов по какому-то признаку.

Есть большое множество алгоритмов, решающих эту задачу. Оценивать алгоритмы мы будем по времени, по дополнительно используемой памяти и по устойчивости. Устойчивой называется сортировка, которая не меняет относительного порядка элементов с равным значением сравниваемого параметра. Параметр элемента, по которому производится сортировка, называется ключом. Усточивость сортировок важна, когда в задаче необходимо сохранить исходный порядок равных элементов. Например, алгоритм BWT, являющийся частью архиватора bzip2, использует внутри устойчивую сортировку.

По большей части рассматриваемые далее алгоритмы сортировки будут иметь работать за $O(n^2)$ в среднем. Такие сортировки называются квадратичными. Однако рассмотрение алгоритмов сортировки начнем с нескольких непрактичных алгоритмов сортировки.

2 Сортировка обезьяны

Данный алгоритм имеет несколько названий: сортировки обезьяны, случайная сортировка, Bogosort. Принцип работы этой сортировки довольно просто. Мы проверяем, является ли массив отсортированным. Если да, то алгоритм завершен. Иначе мы случайно перемешиваем массив и снова проверяем. И так, пока мы не отсортируем его. Время работы в среднем составляет $O(n \cdot n!)$, дополнительной памяти не требуется. Эта сортировка настолько не практичная, что ее вообще не следует когда-либо использовать.

```
import random


def shuffle(a):
    n = len(a)
    for i in range(n):
        j = random.randint(0, n - 1)
        a[i], a[j] = a[j], a[i]
```

```
def is_sorted(a):
    for i in range(len(a) - 1):
        if a[i] > a[i + 1]:
            return False
    return True


def random_sort(a):
    while not is_sorted(a):
        shuffle(a)
```

3 Глупая сортировка

Алгоритм этой сортировки заключается в следующих шагах. Просматриваем массив слева-направо и по пути сравниваем соседей. Если мы встретим пару взаимно неотсортированных элементов, то меняем их местами и возвращаемся в начало массива. И так далее, пока массив не будет отсортирован. Такой алгоритм потребует $O(n^3)$ времени.

```
def stupid_sort(a, b):
    n = len(a)
    for k in range(n):
        for i in range(n):
            for j in range(n - i + 1):
                if a[j] > a[j + 1]:
                    a[j], a[j + 1] = a[j + 1], a[j]
                    break
```

4 Сортировка пузырьком

Попробуем улучшить алгоритм глупой сортировки. Теперь, когда мы встретили пару взаимно неотсортированных элементов и поменяли их местами, мы не возвращаемся в начало, а продолжаем двигаться по массиву. Таким образом выполнив один проход по массиву, мы вытолкнем в его конец максимальный элемент. После второго прохода мы вытолкнем второй по величине элемент и так далее. Всего нам надо будет сделать $O(n^2)$ операций. Данная сортировка является устойчивой.

```
def bubble_sort(a, b):
    n = len(a)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
```

Далее мы слегка модифицируем внешний цикл. Теперь мы будем выполнять этот цикл не $n-1$ раз, а до тех пор, пока происходит хотя бы один обмен. Как было показано ранее, более $n-1$ итераций внешнего цикла нам не понадобится. Однако, мы можем получить отсортированный массив быстрее. И тогда нам не надо будет делать пустые проходы по массиву. Тогда в случае уже отсортированного массива мы завершим работу после первой итерации, что даст нам время $O(n)$ в лучшем случае. Среднее время остается тем же.

```
def bubble_sort_improved(a):
    n = len(a)
    f = True
    i = 0
    while f:
        f = False
        for j in range(n - i - 1):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                f = True
        i += 1
```

5 Сортировка выбором

Сортировка выбором работает по следующему принципу. Сначала из всего массива выбирается минимальный элемент и меняется с нулевым элементом. Далее мы выбираем второй по величине элемент и меняем его с первым элементом. После этого проделываем похожую операцию со второй ячейкой массива и т.д. Таким образом мы получим отсортированный массив. Выбор элемента для очередной ячейки массива выполняется за $O(n)$, всего таких выборов $n-1$. Итого, время работы алгоритма – $O(n^2)$. Сортировка выбором является неустойчивой.

```
def selection_sort(a):
    n = len(a)
    for i in range(n - 1):
        k = iselection_sort
        for j in range(i + 1, n):
            if a[k] > a[j]:
                k = j
        a[i], a[k] = a[k], a[i]
```

6 Сортировка вставками

Сортировка вставками очень похожа на сортировку пузырьком, однако между ними различие есть, и не стоит их путать. На каждой итерации

алгоритма будет выполняться сортировка только первых i элементов. Пусть уже первые $i-1$ элементов отсортированы, и мы добавили в конец новый элемент. Этот новый элемент нужно передвинуть на правильное место, чтобы снова получить отсортированный массив. Будем просто пытаться обменивать его местами с соседом слева, если этот сосед больше нового элемента. Эта сортировка – устойчивая, время работы – $O(n^2)$.

```
def insertion_sort(a):
    n = len(a)
    for i in range(n):
        j = i - 1
        while j >= 0 and a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]
            j -= 1
```

Список литературы

Кормен, Томас и др. (2013). *Алгоритмы. Построение и анализ. Третье издание*. Издательский дом «Вильямс».