Лекция №14.

Тема: **Организация сортировки данных. Часть 1.**

Понятие сортировки данных. Цель сортировки. Факторы, которые необходимо учитывать в процессе сортировки. Критерии эффективности алгоритмов сортировки. Классификации методов сортировки. Простые методы сортировки данных в массиве. Сложные методы сортировки данных в массиве.

Ключевые слова: сортировка, сортировка простыми обменами, сортировка выбором, сортировка вставками.

Keywords: sort, bubble sort, selection sort, insertion sort.

1 Понятие сортировки данных

Сортировка, как стандартная операция, предполагает расположение элементов, которые хранятся в общем случае в произвольном порядке, в вполне определенном порядке: возрастания, неубывания, убывания, невозрастания.

Если для элементов массива A выполняется условие

$$A[i-1] < A[i], \text{ для } 0 <= i < N,$$

то говорят, что элементы массива A упорядочены строго по возрастанию (совпадающих по значению элементов в массиве нет). Аналогично формулируется строгое убывание элементов. Наличие операций отношения «меньше либо равно» и «больше либо равно» говорит о неубывании и невозрастании соответственно (совпадающие по значению элементы в массиве могут быть).

В широком смысле слова «сортировка» – это процесс разделения данных на некоторые группы или сорта, но в программировании чаще сортировка рассматривается в более узком смысле – как процесс размещения данных в определенном порядке.

В классическом труде Дональда Э. Кнута «Искусство программирования. Том 3. Сортировка и поиск» (первое издание вышло в 1972 году), Кнут описывает основные области применения сортировки:

1. Решение задачи группирования – использование сортировки по невозрастанию или по неубыванию фактически решает задачу последовательного размещения равных по значению элементов.
2. Поиск общих элементов в двух или более массивах - для двух или более массивов, уже отсортированных в нужном порядке, можно выполнить за один последовательный просмотр всех массивов (без возвратов).
3. Поиск информации по значениям ключей – поиск информации в отсортированной последовательности данных (массиве, например) эффективнее, чем поиск в данных, которые расположены в произвольном порядке.

2 Цель сортировки

Целью алгоритмов сортировки, является изменение существующего порядка расположения данных в соответствии с некоторым, заранее оговоренным порядком. Алгоритмы сортировки, в отличие от алгоритмов поиска, являются мутационными алгоритмами, то есть алгоритмами, которые изменяют исходное расположение данных. Часто это требует дополнительных объемов памяти.

3 Факторы, которые необходимо учитывать в процессе сортировки

На процесс сортировки оказывают влияние, практически те же факторы, которые влияют на процесс поиска данных. Любая дополнительная информация о исходных данных позволяет сделать сортировку более эффективной для данной предметной области (области деятельности человека, из которой получены данные).

Перечислим факторы, влияющие на выбор метода сортировки.

1. Структура данных – сортировка может выполняться для одномерного целочисленного массива, или некоторых подстрок в тексте, или данных типа «запись», как элементов «сильно ветвящегося дерева», то есть над данными простых типов или в сложно организованных базах данных. Кроме этого, существующий порядок расположения данных может изначально частично или полностью соответствовать нужному порядку сортировки, а учет такого состояния данных сокращает избыточные операции.
2. Объем данных – размер данных оказывает влияние на выбор алгоритма сортировки. Обычно простые алгоритмы сортировки применяют к небольшим по объему данным. Сложные алгоритмы сортировки учитывают дополнительную информацию, затраты на получение которой окупаются на фоне еще больших затрат на выполнение сортировки в больших объемах данных.
3. Способ хранения данных – данные могут храниться в оперативной памяти или в долговременной памяти, причем файлы данных могут быть прямого или последовательного доступа. Способ хранения существенно влияет на выбор алгоритма сортировки. Выделяют два класса алгоритмов сортировки – внутренняя сортировка (в оперативной памяти) и внешняя сортировка (в файлах). Большинство алгоритмов внешней сортировки построены на идее слияния уже отсортированных в оперативной памяти наборов данных (внутренняя сортировка с последующим внешним слиянием).
4. Оценки эффективности алгоритма поиска – выбор алгоритма сортировки для конкретной прикладной задачи определяется по критерию минимизации затрат по времени и памяти для реализации и выполнения алгоритма сортировки.

4 Классификация методов сортировки

Метод сортировки называется устойчивым, если относительный порядок элементов с одинаковыми ключами не меняется при сортировке.

Метод сортировки обладает естественным поведением, если временные затраты у метода больше в случае, когда исходные данные отсортированы в обратном порядке, чем в случае, когда исходные данные отсортированы в нужном порядке.

Метод сортировки называется простым, если в своей реализации метод не использует дополнительной информации о данных, кроме знания о незначительном размере данных (для массива в 20-30 элементов).

Метод сортировки называется сложным, если в своей реализации метод использует дополнительную информацию о данных, включая знание о значительном размере данных (для массива из более чем 30 элементов).

5 Критерии эффективности алгоритмов сортировки

Важными оценками алгоритма сортировки являются время поиска, количество операций перестановки элементов и объем дополнительной памяти, необходимой для реализации алгоритма. Анализ элементарных операций при сортировке показал, что более эффективным (по времени) алгоритмом сортировки является алгоритм, который использует меньшее количество операций сравнения и меньшее количество операций перестановки. Программный подсчет количества сравнений и перестановок позволяет оценить временную сложность алгоритма сортировки, а подсчет количества сравнений и перестановок при разном объеме исходных данных позволяет оценить временную экспоненциальную сложность алгоритма сортировки.

Кроме этого, временные затраты алгоритма сортировки можно получить использованием функции timeit.timeit (аналогично расчетам при поиске).

6 Описание простых алгоритмов сортировки

В данной лекции рассматриваются алгоритмы, наиболее часто применяемые для сортировки одномерных массивов.

6.1 Сортировка простыми обменами (bubble sort)

Сортировка простыми обменами (пузырьковая сортировка, см. https://en.wikipedia.org/wiki/Bubble_sort) – простейший алгоритм сортировки, который основан на принципе сравнения и обмена пары соседних элементов до тех пор, пока не будут рассортированы все элементы массива.

Для реализации алгоритма используются два вложенных цикла. Внешний цикл обеспечивает N-1 проход по массиву. На каждом проходе внутренний цикл обеспечивает последовательное сравнение и обмен (если порядок неверный) пары соседних элементов. Если элементы массива представить пузырьками в резервуаре с водой, обладающими соответствующими весами, то после каждого прохода как минимум один пузырек «всплывает» на соответствующий его весу уровень.

Рассмотрим реализацию алгоритма пузырьковой сортировки на языке Python.

```
# Объявление и инициализация массива A.

...

N = len(A)
for i in range(1, N):
    for j in range(N - 1, i - 1, -1):
        # Сортировка по возрастанию (неубыванию)- знак ">".
        # Сортировка по убыванию (невозрастанию)- знак "<".
        if (A[j - 1] > A[j]):
            # Перестановка элементов массива A.
            tmp = A[j]
            A[j] = A[j - 1]
            A[j - 1] = tmp
```

Сортировка должна быть остановлена, если после выполнения внутреннего цикла не произошло ни одного обмена. Добавим в условие внешнего цикла флажок и ускорим процесс сортировки массива

```
# Объявление и инициализация массива A.

...

N = len(A)
i = 0
flag = True
while flag:
    flag = False
    for j in range(N - i - 1):
        # Сортировка по возрастанию (неубыванию)- знак ">".
        # Сортировка по убыванию (невозрастанию)- знак "<".
        if (A[j] > A[j + 1]):
            # Перестановка элементов массива A.
            tmp = A[j]
            A[j] = A[j + 1]
            A[j + 1] = tmp
            flag = True
    i += 1
```

Данный алгоритм является простейшим для понимания и реализации, но эффективен на массивах небольшой длины.

6.2 Сортировка выбором (selection sort)

Сортировка выбором (selection sort, см. https://en.wikipedia.org/wiki/Selection_sort) – простой алгоритм сортировки, который на каждом i-ом проходе по массиву ($0 \le i \le N-2$) выполняет поиск минимального (максимального) элемента среди последних $N - i$ неупорядоченных элементов и последующий обмен найденного элемента с i-ым элементом массива.

Рассмотрим реализацию алгоритма сортировки выбором на языке Python.

```
# Объявление и инициализация массива A.

...

N = len(A)
for i in range(N - 1):
    min = i
    for j in range(i + 1, N):
        # Сортировка по возрастанию (неубыванию)- знак "<".
        # Сортировка по убыванию (невозрастанию)- знак ">".
        if A[j] < A[min]:
            min = j
    # Перестановка элементов массива A.
    tmp = A[i]
    A[i] = A[min]
    A[min] = tmp
```

Время работы данного алгоритма незначительно зависит от степени упорядоченности исходного массива. Поиск минимального (максимального) элемента за один проход не дает дополнительной информации об исходном массиве, что не позволяет повысить эффективность алгоритма.

6.3 Сортировка вставками (insertion sort)

Сортировка простыми вставками (insertion sort, см. https://en.wikipedia.org/wiki/Insertion_sort) – простой алгоритм сортировки, который заключается в том, что отдельно анализируется каждый конкретный элемент, который затем помещается в отсортированную часть массива. Место вставки в отсортированную часть массива определяется методом линейного поиска. Для выполнения собственно вставки элемента в массив необходимо выполнить перезапись всех элементов массива, начиная с места вставки и до конца массива.

Рассмотрим реализацию алгоритма сортировки простыми вставками на языке Python.

```
# Объявление и инициализация массива A.

...

N = len(A)
for i in range(1, N):
    j = i - 1
    key = A[i]
    # Сортировка по возрастанию (неубыванию)- A[j] > key.
    # Сортировка по убыванию (невозрастанию)- A[j] < key.
    while j >= 0 and A[j] > key:
        A[j + 1] = A[j]
        j -= 1
    A[j + 1] = key
```

Основные преимущества алгоритма:

- прост в реализации;
- эффективен на небольших наборах данных;
- эффективен на наборах данных, которые уже частично отсортированы;
- устойчивый алгоритм сортировки;
- может сортировать список по мере его получения.

Основной недостаток алгоритма: большие временные затраты на перезапись целой последовательности элементов на одну позицию при включении элемента.

Быстродействие алгоритма сортировки простыми вставками можно увеличить, если вместо линейного поиска позиции вставки элемента в отсортированную часть массива использовать бинарный поиск.

Рассмотрим реализацию алгоритма сортировки бинарными вставками на языке Python.

```
# Объявление и инициализация массива A.

...

N = len(A)
for i in range(1, N):
    x = A[i]
    L = 0
    R = i - 1
```

```
        while L <= R:
            m = (L + R) // 2
            if x < A[m]:
                R = m - 1
            else:
                L = m + 1
        # Перестановка элементов массива A.
        for j in range(i - 1, L - 1, -1):
            A[j + 1] = A[j]
        A[L] = x
```

По сравнению с алгоритмом сортировки простыми вставками, количество операций сравнения меньше, однако количество операций перемещений все также много. Причем, если исходный массив уже отсортирован, то результаты у данного метода хуже, чем у алгоритма сортировки простыми вставками.

Сравнительный анализ простых алгоритмов сортировки представлен в таблице 1.

Таблица 1 – Сравнительная характеристика алгоритмов сортировки

Название	Время			Память	Устойчивость	Количество обменов (в среднем)
	Лучшее	Среднее	Худшее			
Пузырьковая сортировка	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да	$O(N^2)$
Сортировка выбором	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(1)$	Нет	$O(N)$
Сортировка вставками	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$	Да	$O(N^2)$