

Сортировка

Куда, как, зачем и почему

Алгоритмы сортировки: что, зачем и почему?

- Алгоритм = пошаговая инструкция, которая дополняет код, прописывается для объяснения значения последнего и произведения компьютером определенного действия. Чаще всего применяются алгоритмы сортировки данных. Последние помогают компьютеру должным образом подойти к работе с информацией.
- Сортировка данных – это то, что будет преследовать программиста от начала учебы и до... Но так как она постоянно нужна и в повседневной жизни, эту подкатегорию алгоритмов следует бояться меньше всего.

- Нам часто требуется что-либо сортировать согласно определенным признакам, но в программировании не все так просто. Для сортировки применяются десятки вариантов алгоритмов и используются они специально для определенных команд. Это единственное в чем легко можно запутаться новичкам, да и более опытные программисты при не частом использовании алгоритмической сортировки могут дать неправильный ответ на собеседовании или просто ошибиться с выбором.

- Самые популярные алгоритмы сортировки:

1. Пузырьковая.
2. Перемешиванием.
3. Вставками.
4. Быстрая.
5. Расчёской.
6. Пирамидальная.
7. Выбором.

- Каждый из них идеален для своей задачи: один – для обработки крупных массивов, другие – для изучения алгоритмических принципов, а третьи – для оптимизации по числу циклов и другим признакам.


Почему без знания алгоритмов не пройти собеседование?

- Рекрутер или руководитель, который проводит собеседование почти всегда дает задачу по алгоритмам на собеседовании. Это необходимо для оценки знаний базиса. Чаще всего она имеет условие, где:
 1. Специально допущена ошибка, например, предлагается решить задачу сложным и времязатратным способом, что необходимо для оценки умения отстаивать свое мнение и внимательно подходить к задачам.

2. Просят использовать один определенный алгоритм, что важно для оценки качества написания кода.
 3. Требуется выбрать любой алгоритм для решения задачи, что позволяет оценить уровень понимания специфики алгоритмов.
- При неправильном ответе вы возможно и пройдете собеседование, но только если покажете, что у вас есть общие знания алгоритмов и к другим навыкам нет вопросов.

- По факту для сортировки по возрастанию достаточно вызвать функцию сортировки Python `sorted()`, которая вернёт новый отсортированный список:


```
1 | >>> sorted([5, 2, 3, 1, 4])
2 | [1, 2, 3, 4, 5]
```



- Также можно использовать метод списка `list.sort()`, который изменяет исходный список (и возвращает `None` во избежание путаницы). Обычно это не так удобно, как использование `sorted()`, но если вам не нужен исходный список, то так будет намного эффективнее.

- Пример:

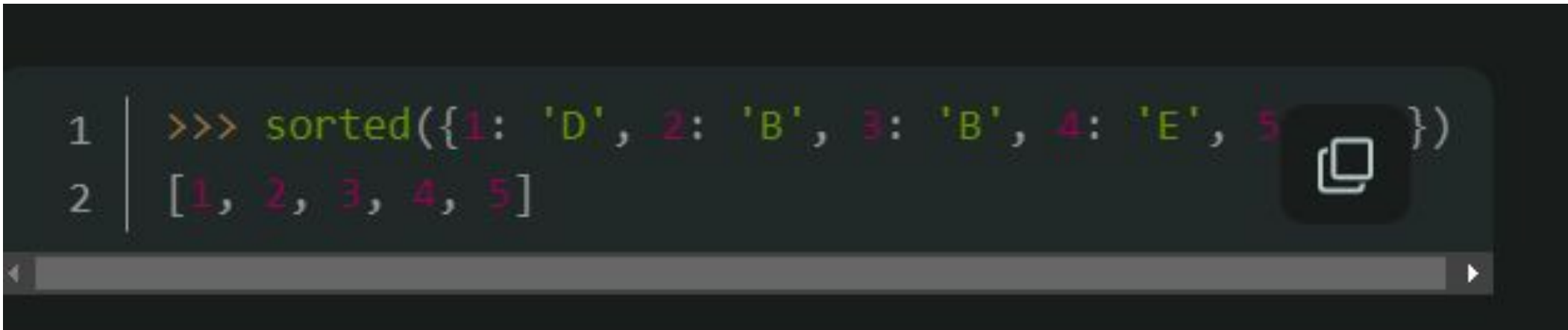
```
1 | >>> a = [5, 2, 3, 1, 4]
2 | >>> a.sort()
3 | >>> a
4 | [1, 2, 3, 4, 5]
```



- В Python вернуть None и не вернуть ничего — одно и то же.
- Ещё одно отличие заключается в том, что метод `list.sort()` определён только для списков, в то время как `sorted()` работает со всеми итерируемыми объектами:

- Пример:

```
1 | >>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})  
2 | [1, 2, 3, 4, 5]
```



- Сортировка пузырьком - это метод сортировки массивов и списков путем последовательного сравнения и обмена соседних элементов, если предшествующий оказывается больше последующего.
- В процессе выполнения данного алгоритма элементы с большими значениями оказываются в конце списка, а элементы с меньшими значениями постепенно перемещаются по направлению к началу списка. Образно говоря, тяжелые элементы падают на дно, а легкие медленно всплывают подобно пузырькам воздуха.
- В сортировке методом пузырька количество итераций внешнего цикла определяется длиной списка минус единица, так как когда второй элемент становится на свое место, то первый уже однозначно минимальный и находится на своем месте.

- Количество итераций внутреннего цикла зависит от номера итерации внешнего цикла, так как конец списка уже отсортирован, и выполнять проход по этим элементам смысла нет.
- Пусть имеется список [6, 12, 4, 3, 8].
- За первую итерацию внешнего цикла число 12 переместится в конец. Для этого потребуется 4 сравнения во внутреннем цикле:

- $6 > 12$? Нет
- $12 > 4$? Да. Меняем местами
- $12 > 3$? Да. Меняем местами
- $12 > 8$? Да. Меняем местами

- Результат:

Результат: [6, 4, 3, 8, 12]

- За вторую итерацию внешнего цикла число 8 переместиться на предпоследнее место. Для этого потребуются 3 сравнения:

- $6 > 4$? Да. Меняем местами
- $6 > 3$? Да. Меняем местами
- $6 > 8$? Нет

Результат: [4, 3, 6, 8, 12]

- На третьей итерации внешнего цикла исключаются два последних элемента. Количество итераций внутреннего цикла равно двум:

- $4 > 3$? Да. Меняем местами
- $4 > 6$? Нет

Результат: [3, 4, 6, 8, 12]

- На четвертой итерации внешнего цикла осталось сравнить только первые два элемента, поэтому количество итераций внутреннего равно единице:

- Финалочка:

- $3 > 4$? Нет

Результат: [3, 4, 6, 8, 12]

- Псевдокод пузырьковой сортировки:

```
1 # Bubble sort
2
3 function bubble_sort( A : list of sortable items)
4
5     while True
6         swapped = false
7
8         for (i=1, i<length(A), i++)
9             if A[i-1] > A[i]
10                swap( A[i-1] , A[i] )
11                swapped = true
12            end if
13        end for
14
15        if not swapped:
16            end while
17    end function
```


- Описание:
- Итак, в псевдокоде мы видим, что мы объявляем функцию с именем `bubble_sort`, которая передала единственный объект списка `A`.
- Затем мы входим в цикл `while`, где мы сначала устанавливаем переменной `swapped = false`.
- Мы будем использовать эту переменную в конце каждой итерации цикла `while`, чтобы посмотреть и проверить – выполнили ли мы какие-либо замены внутри нашего цикла `for`. Если нет – мы выполнили сортировку.
- Цикл `for` сам увеличивает переменную длины входного списка `A`. На каждой итерации цикла `for` мы проверяем и смотрим, не отсортированы ли элементы с индексом `i` и `i-1`.

- Если это так, мы меняем их местами, и устанавливаем в нашей переменной `swapped` значение `True`. В противном случае мы переходим к следующей итерации цикла `for`.
- В первой итерации цикла `for` мы проверяем элемент с нулевым индексом на элемент с индексом `-1`. Мы меняем их местами, если элемент с нулевым индексом больше, чем элемент с индексом `-1`.
- После цикла `for` мы проверяем – выполняем ли мы какие-нибудь подкачки. (`swapped = True`).

- Программная реализация нормальной пузырьковой сортировки.

```
107 def create_array(size=10, max=50):  
108     from random import randint  
109     return [randint(0, max) for _ in range(size)]  
110
```

```
111  
112 def bubble_sort(arr):  
113     swapped = True  
114     while swapped:  
115         swapped = False  
116         for i in range(1, len(arr)):  
117             if arr[i-1] > arr[i]:  
118                 arr[i], arr[i-1] = arr[i-1], arr[i]  
119                 swapped = True  
120     return arr  
121  
122 a = create_array()  
123 print(a)  
124 a = bubble_sort(a)  
125 print(a)
```