Sorting Arrays And Lists: Takeaways

by Dataquest Labs, Inc. - All rights reserved © 2021

Syntax

- Sorting a list:

```
list.sort()
```

- Implementing a swap:

```
def swap(array, pos1, pos2):
    store = array[pos1]
    array[pos1] = array[pos2]
    array[pos2] = store
```

- Implementing selection sort:

```
def selection_sort(array):
    for i in range(len(array)):
        lowest_index = i
        for z in range(i, len(array)):
            if array[z] < array[lowest_index]:
                lowest_index = z
        swap(array, lowest_index, i)
```

- Implementing bubble sort:

```
def bubble_sort(array):
    swaps = 1
    while swaps > 0:
        swaps = 0
        for i in range(len(array) - 1):
            if array[i] > array[i+1]:
                swap(array, i, i+1)
                swaps += 1
```

- Implementing insertion sort:

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

Concepts

- The default sorting behavior isn't ideal in the following cases:
    - We have a custom data structure, and we want to sort it. For example, we want to sort a set of JSON files.

    - We're working with data that's too large to fit into memory, but we still want to ensure that everything is sorted. This may require splitting the data across multiple machines to sort, and then combining the sorted results.
    - We want a custom ordering — for example, we want to sort locations based on their proximity to one or more cities. We can't sort by simple distance to the closest city, since we want to take distance to multiple cities into account.
- There are a variety of different sorting techniques that have different time and space complexity trade-offs.
- The basic unifying factor behind most sorting algorithms is the idea of the swap. Most sorting algorithms differ only in which order different items are swapped.
- Pseudocode for making a swap:
    - Select the two elements you want to swap.
    - Copy the first element to an external variable.
    - Replace the first element with the value of the second.
    - Replace the second element with the value of the external variable.
- How each type of sort works:
    - The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.
    - A bubble sort works by making passes across an array and "bubbling" values until the sort is done.
    - The insertion sort works by looping through each element in the array and "inserting" it into a sorted list at the beginning of an array.
- Time complexity for each sorting algorithm:
    - Selection sort has $O(n^2)$ time complexity.
    - A bubble sort has time complexity of $O(n)$ when the array is already sorted but has time complexity of $O(n^2)$ otherwise.
    - An insertion sort has time complexity of $O(n)$ when the array is already sorted but has time complexity of $O(n^2)$ otherwise.
- Space complexity for each sorting algorithm:
    - Selection sort has $O(1)$ space complexity.
    - Bubble sort has $O(1)$ space complexity.
    - Insertion sort also has $O(1)$ space complexity.

Resources

- Selection Sort
- Bubble Sort
- Insertion Sort
- Sorting Terminology

Takeaways by Dataquest Labs, Inc. - All rights reserved © 2021