Theory: Quick sort

20 minutes 10 / 10 problems solved

Start practicing

1100 users solved this topic. Latest completion was about 11 hours ago.

Quicksort is an efficient in-place sorting algorithm that is often faster in practice compared to other sorting algorithms. The algorithm is based on the divide-and-conquer paradigm.

Here are the steps of quicksort:

1. Pick some element from the array. We will call that element a **pivot**.
2. Reorder the array so that all values smaller than the pivot are positioned before it and all larger values come after; values equal to the pivot can go either way.
3. Recursively sort the subarrays of smaller and greater elements.

The base case of recursion is arrays the size of zero or one, which is in order by definition, so they never need to be sorted.

Quicksort can be implemented as a recursive or iterative algorithm. Here we will consider only the recursive version.

The time complexity is $O(nlogn)$ in the average case, and $O(n^2)$ in the worst case, but fortunately, it is usually average. We will consider some bad cases later.

Note, there are a lot of modifications that make the algorithm more efficient. The pivot selection and partitioning steps can be implemented in different ways. The choice of a specific implementation strategy greatly affects the algorithm's performance.

§1. Choosing a pivot

Your choice of pivot strongly affects the sorting time. It's quite difficult to determine a good pivot for all arrays.

The best pivot would divide the array into two equal parts, which would halve the problem size. However, this means that the pivot is the median of the elements, and in order to find the median, we would either need to use an already sorted array or a more complex approach to find the median.

Here are some possible methods of choosing the pivot:

- Pick the leftmost or the rightmost element;
- Pick the middle element;
- Pick a random element;
- Take the first, middle and last value of the array and choose the median of these three numbers as the pivot.

In this topic, we will always go for the rightmost element. This approach is called the **Lomuto partition scheme**. Later on, you will learn about more complex strategies for choosing the pivot.

§2. Example

Suppose we need to sort an array of ints using quicksort algorithm. As the pivot, it will always pick the rightmost element.

The input array contains 8 ints with indexes from 0 to 7. The following image illustrates how quicksort works with more explanation provided below:

1) Let's pick the rightmost element 14 as the pivot in the original array and then reorder it so that all values smaller than the pivot – `11, 10, 13` – come before it and all values larger than the pivot – `17, 25, 16, 22` – follow it. After the rearrangement, the pivot has the index 3.

Current topic:

✓ Quick sort

Table of contents:

2) Divide the array into two subarrays: the left one `{ 11, 10, 13 }` and the right one `{ 17, 25, 16, 22 }`. The pivot is excluded from both subarrays.

3) Consider the left subarray. The pivot is 13. After rearranging, we obtain the same order of elements in the array `{ 11, 10, 13 }`.

4) Divide the subarray `{ 11, 10, 13 }` into two smaller subarrays: the left one `{ 11, 10 }` and the empty subarray.

5) Consider the subarray `{ 11, 10 }`. The pivot is 10. After rearranging, we obtain the order `{ 10, 11 }`.

6) Divide the subarray `{ 10, 11 }` into two smaller subarrays: the empty array and `{ 11 }`. Both new arrays are already sorted. The pivot 10 is excluded.

Note, we consider empty and single-element arrays as already sorted and do not process them.

7) We go through the same steps for the right subarray `{ 17, 25, 16, 22 }` of the original array until all subarrays are empty or consist of a single element.

After, the whole array is sorted: `{ 10, 11, 13, 14, 16, 17, 22, 25 }`.

When we have chosen a pivot and rearranged the array, the pivot is in its final position. All left elements are less than the pivot and all right elements are greater than the pivot. But they may be not ordered relative to each other at the current step.

You can take a look at a visualization if you wish.

§3. Problems and possible modifications

1) Unfortunately, the Lomuto partition scheme causes worst-case behavior $O(n^2)$ in already sorted arrays. The problem can be solved by opting for a different pivot-picking strategy: choosing either a random index for the pivot, choosing the middle index of the partition, or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot (as recommended by Sedgewick).

The median-of-three option counters the case of sorted (or reverse-sorted) input and gives a better estimate of the optimal pivot (the true median) than selecting a random element when nothing is known about the ordering of the input.

2) When all the input elements are equal, one subarray is always empty while another has only decreased by one element (the pivot is removed). This causes the worst-case behavior.

To avoid the problem, we can separate the values into three groups: values less than the pivot, values equal to the pivot, and values greater than the pivot. The values equal to the pivot are already sorted, so only the less-than and greater-than partitions need to be recursively sorted.

So, it's possible to improve the worst case of quicksort to make it faster on many data sets.

Report a typo

90 users liked this theory. 3 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion