Theory: Quick sort in Python

1 hour 0 / 5 problems solved

Quick sort is an efficient comparison-based sorting algorithm: for a list of size n, its average time complexity is $O(n \log n)$. In the worst-case, the algorithm works in $O(n^2)$. However, being well implemented, it works mostly like on average and even outperforms other efficient sorting algorithms such as merge sort or heap sort. In this topic, we will consider how the algorithm can be implemented in Python.

§1. Implementing the partition procedure

The main subroutine of the **quick sort** algorithm is the **partition** procedure. Given a list, it chooses one of its elements as a **pivot** and rearranges the list such that all elements smaller or equal to the pivot appear from the left of it, and the other elements are placed from the right of the pivot. After the procedure is completed, the pivot element appears to be in the position in which it should be placed in a sorted list.

Below is one possible implementation of the procedure in Python:

```
def partition(lst, start, end):
    j = start

    for i in range(start + 1, end + 1):
        if lst[i] <= lst[start]:
            j += 1
            lst[i], lst[j] = lst[j], lst[i]

    lst[start], lst[j] = lst[j], lst[start]

    return j
```

A function named `partition` takes a list `lst`, `start` and `end` positions (both inclusive) of a range where the partition should be performed. In this version, we consider the leftmost element of the range to be a pivot.

First, we create a variable `j` to store the rightmost position (inclusive) of a range, where all elements are smaller or equal to the pivot. Initially, the range includes only the pivot element itself. Next, we start iterating through the elements of the list using the `for` loop. At each iteration, we compare the current element with the pivot and, in case the element is smaller or equal to it, we place this element to the end of the current range and increase its size by one. Finally, we place the pivot to the end of the range getting the desired order of elements.

Below is an example of how the function works:

```
lst = [4, 8, 3, 5, 1, 9, 2, 0, 2, 7]
j = partition(lst, 0, len(lst) - 1)  # j = 5
print(lst)  # [2, 3, 1, 2, 0, 4, 5, 8, 9, 7]
```

After the function is completed, the pivot element, which is 4, is placed on the 5th position. All elements from the left of the pivot are smaller or equal to it, the elements greater than the pivot are placed from the right of it.

§2. Implementing quicksort

Using the partition procedure, the quick sort algorithm can be implemented as follows: first, we apply the partition for an input list. Then, having the pivot element in its proper position, we recursively apply the quick sort for the left and the right part of the resulting list (excluding the pivot element) finally getting the whole list sorted.

Below is an implementation of the algorithm in Python:

Current topic:

Quick sort in Python

Topic depends on:

- ✓ Quick sort
- ✓ Algorithms in Python
- ✗ Recursion in Python

Table of contents:

↑ Quick sort in Python

§1. Implementing the partition procedure

§2. Implementing quicksort

Feedback & Comments

```
def quick_sort(lst, start, end):
    if start >= end:
        return

    j = partition(lst, start, end)
    quick_sort(lst, start, j - 1)
    quick_sort(lst, j + 1, end)
```

The `quick_sort` function takes a list `lst` and a range of the list to be sorted. Both the `start` and the `end` positions are inclusive.

First, we check whether the range is either empty or contains only one element. If the condition holds, we quit the function, since there is nothing to sort. Otherwise, we perform the partition on the current range and then, recursively sort the left and the right part of the resulting list.

Below is an example of how the function works:

```
dates = [1994, 1998, 1993, 1995, 1991, 1999, 1992, 1990, 1992, 1997]
quick_sort(dates, 0, len(dates) - 1)

print(dates)  # [1990, 1991, 1992, 1992, 1993, 1994, 1995, 1997, 1998, 1999]
```

As expected, the elements of the input list are sorted in ascending order.