Sorting (VI): Quicksort

CSCD 300 – Data Structures

Eastern Washington University

EASTERN WASHINGTON UNIVERSITY
start something big

Goal

We will learn the mechanism of the Quicksort algorithm and then analyze its time complexity in the best as well as in the worst case.

Quicksort has its randomized version and deterministic version. Here we only discuss its deterministic version. The randomized version is out of the scope of this course.

Outline

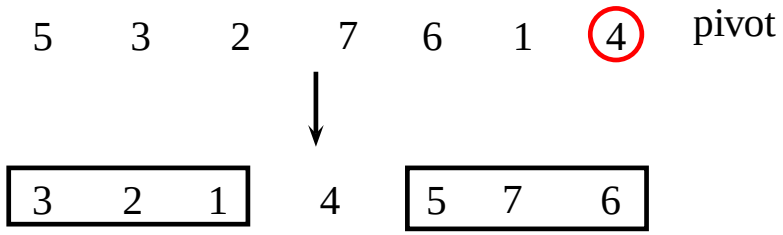
Quicksort

Basic idea

The overall idea is to use recursion:

1. We pick any number in the sequence as the pivot. For convenience, we often pick the rightmost number in the sequence as the pivot.
2. We use that pivot to partition all the numbers in the sequence into three subsequences: the left subsequence, the pivot, and the right subsequence. All the numbers in the left subsequence are smaller than or equal to the pivot; all the numbers in the right subsequence are larger than the pivot.
3. We then apply the same idea from step 1 and 2 to the left and the right subsequences recursively until the size of the working sequence is less than or equal to one.

An example of the first-level partition:



Challenge

How to efficiently partition the sequence into three pieces in the place of the array, so that no extra space is needed.

```
/* Partition A[left...right] using A[right] as the pivot.
   Return: the index of the array location that saves the
           pivot after partition.
*/
partition(A,left,right)
{
   pivot = A[right];
   index = left;
   for(i = left ... right-1)
      if(A[i] <= pivot)
         swap(A[index],A[i]);
         index ++;
    swap(A[index],A[right])
    return index;
}
```

Then, we are ready to give the Quicksort ...

Quicksort

```
/*Sort A[left...right] using quicksort*/
Quicksort(A, left, right)
{
   if(left >= right)
      return;

   pivot_index = partition(A,left,right);
   Quicksort(A, left, pivot_index - 1);
   Quicksort(A, pivot_index + 1, right);
}
```

Function call `Quicksort(A,0,n-1)` sorts the whole array A.

The time complexity: the best case

The best case is when every pivot that we use in each recursive function call partitions the sequence into two (almost) evenly sized subsequences [1].

Then the time cost of Quicksort can be expressed as:

$$T(n) = 2T(n/2) + O(n)$$

where each $T(n/2)$ is the time cost for recursively sorting each of the two subsequences and $O(n)$ captures the time cost for the partitioning. Recall that the time cost function of Mergesort [2] has exactly the same structure, so:

$$T(n) = O(n \log n)$$

---

[1] In fact, it is enough if the ratio of the sizes of the two subsequences is bounded by a constant, but we will not explain why in this class.

[2] Refer to the lecture slides on Mergesort for the derivation of the time cost of Mergesort if you have forgotten it.

The time complexity: the worst case

The worst case is the input sequence is either in ascending or descending order. In that case, after each partition, the whole sequence except the pivot goes to one single subsequence, so the total time cost of Quicksort can be written as:

$$
\begin{aligned}
T(n) &= n + T(n-1) \\
&= n + (n-1) + T(n-2) \\
&\vdots \\
&= n + (n-1) + \ldots + 1 \\
&= \frac{n(n+1)}{2} \\
&= O(n^2)
\end{aligned}
$$

Question

How do you use the Quicksort if the data sequence is saved in a singly linked list or doubly linked list ?