Quicksort

COMP 105 staff

Spring 2016

1 Quicksort on arrays

Quicksort was invented by Tony Hoare around 1960. It was one of the first inherently recursive sorting algorithms, and it is still one of the fastest known algorithms. Although in CS 105 we implement Quicksort on lists, it is more normally done on arrays. This handout, borrowed from Margo Seltzer's staff, illustrates the array algorithm.

1.1 Sketch of the algorithm

1. If the array is already sorted, nothing needs to be done. This is the base case of the recursive sort.

   (Hint: any sufficiently small array is sorted.)

2. If the array is not sorted, it has at least two elements not in order. Therefore it has at least one element. Therefore we can pick an element, which we will call the *pivot*.

   Quicksort is correct with any choice of pivot, but for it to perform well, you need a "good" pivot. In the late 1970s, Bob Sedgewick wrote a very interesting doctoral thesis which covers, among other things, many ways to choose a pivot. You may choose a pivot using any method you like, but we urge you toward simplicity.

3. Given a pivot, it is now possible to rearrange the array into three subarrays.[1]

   - Items whose value is at most the value of the pivot
   - Items whose value is exactly the value of the pivot
   - Items whose value is at least the value of the pivot

   Each one of these arrays is then sorted recursively. At the end of this step, the whole array is sorted.

   It is up to you to make sure the recursion terminates. Termination depends on how you split the array into subarrays. The split can be done in a variety of ways. Be careful; depending on how you do the split and on the contents of the array, any one of these subarrays can be empty.

The running time of Quicksort depends on exactly what happens during the splitting step. Quicksort's expected running time is $O(n \log n)$; the worst case is $O(n^2)$. To get good running time in practice, consult Sedgewick.

[1] In the imperative algorithm on arrays, the rearrangement is done by a sequence of swap operations. Since you are sorting lists, you will have to find another way to do things.

1.2 Example

Here is an array to be sorted:

6	11	4	14	13	3	9	5	8	2

As pivot, we choose the first value. To split and rearrange, we set LEFT and RIGHT pointers:

6	11	4	14	13	3	9	5	8	2
PIVOT	LEFT								RIGHT

Now we swap elements until elements at most the pivot value are to the left all elements at least the pivot value are to the right. This is done by starting off LEFT and RIGHT pointers at the two ends of the array and moving them inward, swapping elements when the LEFT pointer has something greater than the pivot value and the RIGHT pointer has something less than the pivot value.

Swap:

6	2	4	14	13	3	9	5	8	11
PIVOT	LEFT								RIGHT

Next mismatch:

6	2	4	14	13	3	9	5	8	11
PIVOT			LEFT				RIGHT		

Swap:

6	2	4	5	13	3	9	14	8	11
PIVOT			LEFT				RIGHT		

Next mismatch:

6	2	4	5	13	3	9	14	8	11
PIVOT				LEFT	RIGHT				

Swap:

6	2	4	5	3	13	9	14	8	11
PIVOT				LEFT		RIGHT			

Now the pointers cross. We swap the pivot with the LEFT element.

3	2	4	5	6	13	9	14	8	11
PIVOT				LEFT		RIGHT			

Finally we split the array into three subarrays:

3	2	4	5

6

13	9	14	8	11

Each of these subarrays can now be sorted recursively.