Quick Sort

Modifications By Mr. Dave Clausen
Updated for Python

Quicksort

There are better sorting algorithms that are O(n log n).

Quicksort is one of the simplest.

The general idea behind quicksort is this:

- *Break an array into two parts*
- *Move elements around so that all the larger values are in one end and all the smaller values are in the other.*
- *Each of the two parts is then subdivided in the same manner, and so on until the subparts contain only a single value, at which point the array is sorted.*

Quicksort

To illustrate the process, suppose an unsorted array, called **a**, looks like:

5	12	3	11	2	7	20	10	8	4	9

Quicksort

Phase 1

1. If the length of the array is less than 2, then done.
2. Locate the value in the middle of the array and call it the pivot. The pivot is 7 in this example.

5	12	3	11	2	7	20	10	8	4	9

3. Tag the elements at the left and right ends of the array as i and j, respectively.

5	12	3	11	2	7	20	10	8	4	9
i										j

Quicksort

4. While a[i] < pivot value, increment i.
   While a[j] >= pivot value, decrement j:

5	12	3	11	2	7	20	10	8	4	9
	i								j	

Quicksort

5. If i > j then

   end the phase

   else

   interchange a[i] and a[j]:

5	4	3	11	2	<u>7</u>	20	10	8	12	9
	i								j	

Quicksort

6. Increment i and decrement j.
   If i > j then end the phase:

5	4	3	11	2	<u>7</u>	20	10	8	12	9
		i						j		

Quicksort

7. Repeat step 4, i.e.,
   While a[i] < pivot value, increment i
   While a[j] >= pivot value, decrement j:

5	4	3	11	2	7	20	10	8	12	9
			i		j					

Quicksort

8. Repeat step 5, i.e.,

   If i > j then

     end the phase

   else

     interchange a[i] and a[j]:

5	4	3	7	2	11	20	10	8	12	9
			i		j					

Quicksort

9. Repeat step 6, i.e.,
   Increment i and decrement j.
   If i < j then end the phase:

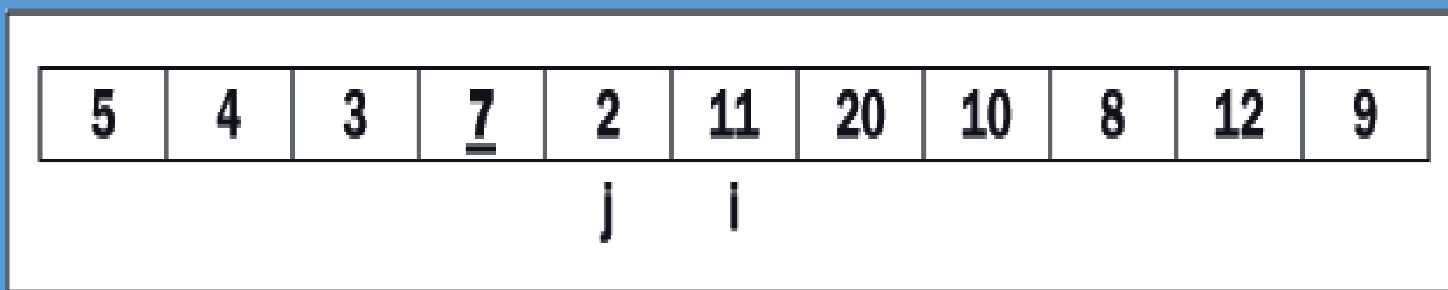
5	4	3	7	2	11	20	10	8	12	9
				ij						

Quicksort

10. Repeat step 4, i.e.,

    While a[i] < pivot value, increment i

    While a[j] >= pivot value, decrement j:



Quicksort

11. Repeat step 5, i.e.,

    If i > j then

    end the phase

    else

    interchange a[i] and a[j].

Quicksort

This ends the phase.

Split the array into the two subarrays a[0..j] and a[i..10].

For clarity, the left subarray is shaded.

Notice that all the elements in the left subarray are less than or equal to the pivot, and those in the right are greater than or equal.

5	4	3	7	2	11	20	10	8	12	9

Quicksort

Phase 2 and Onward

Reapply the process to the left and right subarrays and then divide each subarray in two and so on until the subarrays have lengths of at most one.

Quicksort

Complexity Analysis

During phase 1, i and j moved toward each other.

At each move, either an array element is compared to the pivot or an interchange takes place.

As soon as i and j pass each other, the process stops.

Thus, the amount of work during phase 1 is proportional to *n*, the array's length.

Quicksort

The amount of work in phase 2 is proportional to the left subarray's length plus the right subarray's length, which together yield *n*.

When these subarrays are divided, there are four pieces whose combined length is *n*, so the combined work is proportional to *n* yet again.

At successive phases, the array is divided into more pieces, but the total work remains proportional to *n*.

Quicksort

To complete the analysis, we need to determine how many times the arrays are subdivided.

When we divide an array in half repeatedly, we arrive at a single element in about $\log_2 n$ steps.

Thus the algorithm is $O(n \log n)$ in the best case.

In the worst case, the algorithm is $O(n^2)$.

Quicksort

Implementation

The quicksort algorithm can be coded using either an iterative or a recursive approach.

The iterative approach also requires a data structure called a ***stack***.

The following example implements the quicksort algorithm recursively:

Quicksort Python

Translated from Java Code

```
def quickSort (listCopy, left, right):
#Recursive Version
    if left >= right:
     return

    i = left
    j = right
    pivotValue = listCopy[(left + right) // 2]
    while i < j:
       while listCopy[i] < pivotValue:
         i+=1
       while pivotValue < listCopy[j]:
         j-=1
       if i <= j:
         temp = listCopy[i]
         listCopy[i] = listCopy[j]
         listCopy[j] = temp;
         i+=1
         j-=1
    quickSort (listCopy, left, j)
    quickSort (listCopy, i, right)
```

Big O Notation for Quick Sort

Big O notation for the average case using Quick Sort is $O(n \log n)$