

Dept. of Computer Science

CSCI 301: Introduction to Algorithms and Data Structures

Instructor: Pranava K. Jha

Quicksort

Quicksort was devised by Tony Hoare in 1961. The title is very apt since the algorithm is indeed very quick in most cases. Not surprisingly, it is the most widely used sorting scheme.

Method of attack: Divide the original array into two subarrays, the first of which contains elements that are less than a chosen key called the *pivot*. The second subarray includes elements that are equal to or greater than the pivot. The two subarrays may be sorted separately using recursion. The base case consists of one-element arrays that are already sorted.

A pseudocode follows.

```
quicksort(array[ ])
{
  if(length(array) > 1)
  {
    choose a pivot;
    while(there are elements left in the array)
    {
      if(the element is smaller than pivot)
        include it in the (initially empty) subarray S1;
      else
        include it in the (initially empty) subarray S2;
    }
    quicksort(S1);
    quicksort(S2);
  }
}
```

<i>Data structure:</i>	Array
<i>Average-case time complexity:</i>	$O(n \log n)$
<i>Worst-case time complexity:</i>	$O(n^2)$

The efficiency of the algorithm critically depends upon the choice of the pivot element. While there are several techniques of choosing the pivot, I let it be equal to the larger of the first two distinct elements of the array.

Example: Perform the top-level partition of *quicksort* on the following sequence of integers that is to be sorted into ascending order: 56, 34, 65, 70, 19, 21, 95, 12.

original array								
56	34	65	70	19	21	95	12	Larger of the first two elements is the pivot.
pivot	unknown							
56	34	65	70	19	21	95	12	First unknown element 34 belongs to S ₁ .
pivot	S ₁	unknown						
56	34	65	70	19	21	95	12	65 belongs to S ₂ .
pivot	S ₁	S ₂	unknown					
56	34	65	70	19	21	95	12	70 belongs to S ₂ .
pivot	S ₁	S ₂	unknown					
56	34	65	70	19	21	95	12	19 belongs to S ₁ , so swap 19 and 65.
pivot	S ₁		S ₂	unknown				
56	34	19	70	65	21	95	12	21 belongs to S ₁ , so swap 21 and 70.
pivot	S ₁			S ₂	unknown			
56	34	19	21	65	70	95	12	95 belongs to S ₂ .
pivot	S ₁				S ₂		unkn own	
56	34	19	21	65	70	95	12	12 belongs to S ₁ , so swap 12 and 65
pivot	S ₁					S ₂		
56	34	19	21	12	70	95	65	Place the pivot element between S ₁ and S ₂ .
S ₁				pivot	S ₂			
12	34	19	21	56	70	95	65	

A program in C++ appears next.

```

// Quicksort (Adapted from Carrano, Fifth edition)
#include <iostream>
#include <fstream>
using namespace std;

typedef int DataType;
const int MAX_SIZE = 10;

void choosePivot(DataType theArray[], int first, int last);
/** Chooses a pivot for quicksort's partition algorithm and swaps
 * it with the first item in the array.
 * pre theArray[first..last] is an array; first <= last.
 * post theArray[first] is the pivot.
 * param theArray The given array.
 * param first The first element to consider in theArray.
 * param last The last element to consider in theArray. */

void partition(DataType theArray[],
               int first, int last, int& pivotIndex);
/** Partitions an array for quicksort.
 * pre theArray[first..last] is an array; first <= last.
 * post Partitions theArray[first..last] such that:
 *   S1 = theArray[first..pivotIndex-1] < pivot
 *   theArray[pivotIndex] == pivot
 *   S2 = theArray[pivotIndex+1..last] >= pivot
 * param theArray The given array.
 * param first The first element to consider in theArray.
 * param last The last element to consider in theArray.
 * param pivotIndex The index of the pivot after partitioning. */

void quickSort(DataType theArray[], int first, int last);
/** Sorts the items in an array into ascending order.
 * pre theArray[first..last] is an array.
 * post theArray[first..last] is sorted.
 * param theArray The given array.
 * param first The first element to consider in theArray.
 * param last The last element to consider in theArray. */

void swap(DataType& x, DataType& y); // swaps x and y

int main()
{
    DataType A[MAX_SIZE];
    ifstream inp;
    inp.open("input.dat"); // Input comes from a file.

    int i = 0;
    while(i < MAX_SIZE)
    {
        inp >> A[i];
        i++;
    }
    inp.close();
}

```

```

quickSort(A, 0, MAX_SIZE-1);
ofstream outp; outp.open("output.dat"); // Output goes to a file.

i = 0;
while(i < MAX_SIZE)
{
    outp << A[i] << " ";
    i++;
}
outp << endl;
outp.close();
return 0;
} // end of main

void choosePivot(DataType theArray[], int first, int last)
{
    if(theArray[first] < theArray[first + 1])
        swap(theArray[first], theArray[first + 1]);
} // end of choosePivot

void partition(DataType theArray[],
               int first, int last, int& pivotIndex)
{
    choosePivot(theArray, first, last);
    DataType pivot = theArray[first]; // copy pivot

    // initially, everything but pivot is in unknown
    int lastS1 = first; // index of last item in S1
    int firstUnknown = first + 1; // index of first item in unknown

    // move one item at a time until unknown region is empty
    for (; firstUnknown <= last; ++firstUnknown)
    { // Invariant: theArray[first+1..lastS1] < pivot
      //           theArray[lastS1+1..firstUnknown-1] >= pivot

      // move item from unknown to proper region
      if (theArray[firstUnknown] < pivot)
      { // item from unknown belongs in S1
        ++lastS1;
        swap(theArray[firstUnknown], theArray[lastS1]);
      } // end if

      // else item from unknown belongs in S2
    } // end for

    // place pivot in proper position and mark its location
    swap(theArray[first], theArray[lastS1]);
    pivotIndex = lastS1;
} // end partition

```

```
void quickSort(DataType theArray[], int first, int last)
{
    int pivotIndex;

    if (first < last)
    { // create the partition: S1, pivot, S2
        partition(theArray, first, last, pivotIndex);

        // sort regions S1 and S2
        quickSort(theArray, first, pivotIndex-1);
        quickSort(theArray, pivotIndex+1, last);
    } // end if
} // end quicksort

void swap(DataType& x, DataType& y)
{
    DataType temp = x;
    x = y;
    y = temp;
} // end swap
```