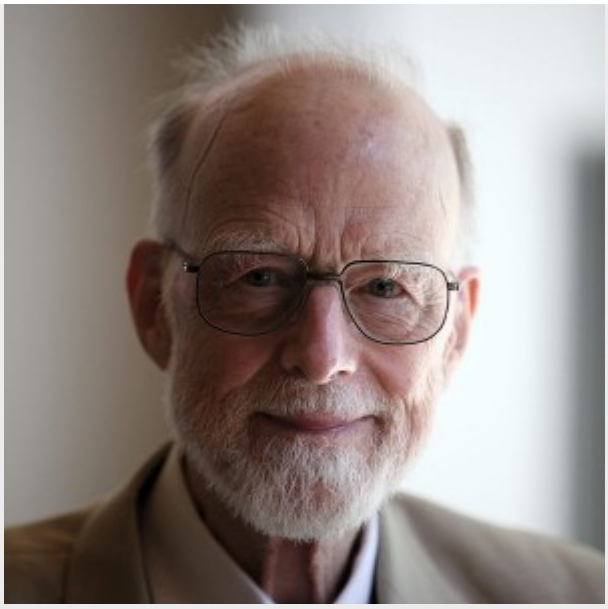
QUICK SORTING ALGORITHM WITH EXAMPLE CODE IN C/CPP/JAVA LANGUAGES

We have seen 3 simple sorting algorithms already **1)** **Bubble Sorting** **2)** **Selection Sorting** and finally **Insertion sorting**. All these algorithms were so simple to understand and were easy to implement as a program in C/C++ or even Java. But at the same time all 3 of them were too inefficient. Their execution time was of the order of n*n, where n is the number of elements to be sorted. In practice, these simple sorting algorithms are seldom used.



Tony Hoare

Quick sort is an improved sorting algorithm developed by **Tony Hoare (C.A.R Hoare)** in **1960**, at the age of 26, while he was working on a machine translation project in Soviet union. You can read more about Tony Hoare and his story of developing Quick sort in Wikipedia – **Quicksort** and **Tony Hoare**

Quick sort is considered as the best general purpose sorting algorithm available till date. The algorithm is developed in a "**divide and conquer**" method which is an important algorithm design paradigm in computer science. Quick sort algorithm divides the array into 2 partitions. To perform this partition, algorithm selects one element in the array as a 'comparand'. Based on this comparand, the algorithm divides all other elements in the array into 2 partitions, one to left side of comparand and other to right side of comparand. Elements on left side partition are all less than comparand and

elements on right side partition are all greater than comparand. Now each of these partitions are sorted separately using the same above process.

Example:- Consider the array was initially **array[5] = {4,3,2,5,1}**; 2 is selected as comparand. After the first pass of the **qsort** function (as defined in the program below), the array would be rearranged as shown below.

array[]= {1, 2, 3, 5, 4};

See, the elements less than comparand ,1 forms first partition (left partition) and elements greater than comparand 3,5,4 forms second partition (right partition). The comprand itself may fit into any one of the partitions as per algorithm. Dont worry about it!

Now look at the bottom lines of the program given below. The same **qsort()** function is called recursively to sort left partition and right partition separately.

To understand the working of quick sort algorithm perfectly, you may observe the below given program carefully. Take a piece of paper and pen, work on the program flow line by line. I have given appropriate comments on each important line of code. If you feel any confusion/doubt – just ask here in comments section.

Example program to implement Quick Sorting in C/C++

```
//PROGRAM TO DEMO QUICK SORTING
#include
#include
void qsort(int *array,int left,int right); //Function qsort declared. Array
to sort is passed to function as a pointer.
void main()
{
int array[5]={4,3,2,5,1}; //Array of 5 elements declared and intialized with
elements.
int left=0,right=4; // right is assigned the index of the last element in the
array.
int x;
qsort(array,left,right); //Fucnction call to perform quicksort.
printf("\n\tThe Sorted array is\n\t");
for(x=0;x<=4;x++)
{
```

```
printf("%d",array[x]);
}
getch();
}
void qsort(int *array,int left,int right) //Function definition of qsort
begins.
{
int i,j,comp,temp; //comp - variable to hold the comparand value.
i=left;
j=right;
comp=array[(left+right)/2]; //comparanad value is determnined as the middle
element in array and is assigned to variable comp.
// Code lines to divide the array into 2 partitions begins. Partitions the
array with elements < comp to left side and elments >comp to right side.
do
{
while((array[i]<comp)&&(i
 {
i++; // If YES, no need to move that element to right side of comp. So i is
incremented
}</comp)&&(i
while((array[j]>comp)&&(j>left)) //Checks if the elements on right side of
comp is already greater than comp.
{
j--; //If YES, no need to move that element to left side of comp. So j is
decremented.
}
if(i<=j) //Code lines to interchange elements to left and right of comp
begins.
{
temp=array[i];
array[i]=array[j];
array[j]=temp;
i++;j--;
}
}while(i<=j); // Continues the process as long as i<=j
// When this do while loop finishes, array will be partitioned into 2 - with
elementscomp on right side.
```

```
// The 2 partitions needs to be sorted seperately. This sorting is achieved
with following lines of codes.
if(left
 {
qsort(array,left,j);
}
if(i
 {
qsort(array,i,right);
}
}
```

Note:- Even if you want into implement quick sort in another programming language like Java, the basic algorithm implementation using loops & recursive calls remains the same. Only the syntax differs.

Selection of the comparand value:-

The selection of comparand value is of some importance in quick sort as it can directly affect the speed of sorting. If the 'comparand' value is selected as the largest value/lowest value in the array, then quick sort will degrade drastically in performance, with speed of execution in the order of n*n. The ideal case is to select the middle value as comparand, which may not be practical unless we have an idea about the nature of data. Generally a comparand is selected inrandom. A much more efficient way is to take the average of 2 or 3 elements and choose the average as comparand.

Source : http://www.circuitstoday.com/quick-sorting-algorithm