Quick Sort

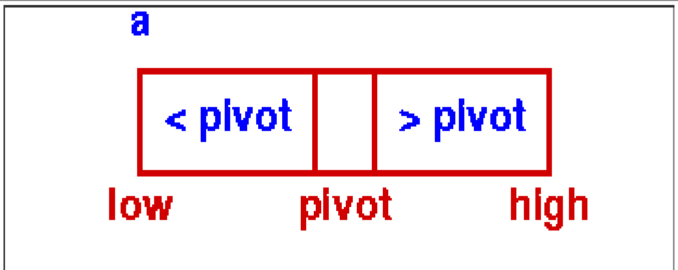
Quicksort is a very efficient sorting algorithm invented by C.A.R. Hoare. It has two phases:

- the partition phase and
- the sort phase.

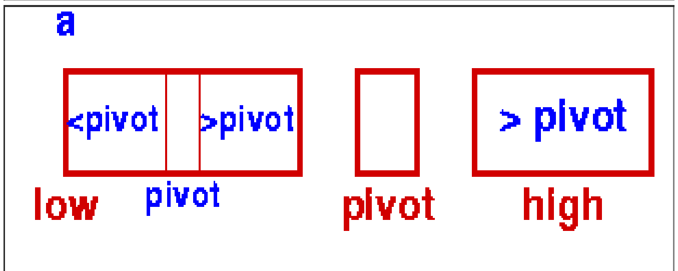
As we will see, most of the work is done in the partition phase - it works out where to divide the work. The sort phase simply sorts the two smaller problems that are generated in the partition phase.

This makes Quicksort a good example of the **divide and conquer** strategy for solving problems. (You've already seen an example of this approach in the binary search procedure.) In quicksort, we divide the array of items to be sorted into two partitions and then call the quicksort procedure recursively to sort the two partitions, *ie* we *divide* the problem into two smaller ones and *conquer* by solving the smaller ones. The *conquer* part of the quicksort routine looks like this:

```
quicksort( void *a, int low, int high )
  {
  int pivot;
  /* Termination condition! */
  if ( high > low )
    {
    pivot = partition( a, low, high );
    quicksort( a, low, pivot-1 );
    quicksort( a, pivot+1, high );
    }
  }
```



Initial step: Partition data



Sort Left partition in the same way

For the strategy to be effective, the *partition* phase must ensure that the pivot is greater than all the items in one part (the lower part) and less than all those in the other (upper) part.

To do this, we choose a *pivot* element and arrange that all the items in the lower part are less than the pivot and all those in the upper part greater than it. In the most general case, we don't know anything about the items to be sorted, so that any choice of the pivot element will do - the first element is a convenient one.

If several items are the same as the pivot, these items can be grouped with the pivot in a third (middle) partition or left in the lower part: change "less than" in the description above to "less than or equal".

CODES(C)

```
#include<stdio.h>

void swap (int a[], int left, int right)
{
 int temp;
 temp=a[left];
 a[left]=a[right];
 a[right]=temp;
}//end swap

void quicksort( int a[], int low, int high )
{
 int pivot;
 // Termination condition!
 if ( high > low )
 {
  pivot = partition( a, low, high );
  quicksort( a, low, pivot-1 );
  quicksort( a, pivot+1, high );
 }
} //end quicksort

int partition( int a[], int low, int high )
{
 int left, right;
 int pivot_item;
 int pivot = left = low;
 pivot_item = a[low];
 right = high;
 while ( left < right )
 {
  // Move left while item < pivot
  while( a[left] <= pivot_item )
   left++;
  // Move right while item > pivot
  while( a[right] > pivot_item )
   right--;
  if ( left < right )
   swap(a,left,right);
 }
 // right is final position for the pivot
 a[low] = a[right];
 a[right] = pivot_item;
 return right;
}//end partition

// void quicksort(int a[], int, int);
void printarray(int a[], int);

int main()
{
 int a[50], i, n;
 printf("\nEnter no. of elements: ");
 scanf("%d", &n);
 printf("\nEnter the elements: \n");
 for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
 printf("\nUnsorted elements: \n");
 printarray(a,n);
 quicksort(a,0,n-1);
 printf("\nSorted elements: \n");
 printarray(a,n);

}//end main


void printarray(int a[], int n)
{
 int i;
 for (i=0; i<n; i++)
  printf(" %d ", a[i]);
 printf("\n");
}//end printarray
```

CODES(JAVA)

```
import java.io.*;
import java.util.*;

public class QuickSort
{
  public static void swap (int A[], int x, int y)
  {
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
  }

  // Reorganizes the given list so all elements less than the first are
  // before it and all greater elements are after it.
  public static int partition(int A[], int f, int l)
  {
    int pivot = A[f];
    while (f < l)
    {
      if (A[f] == pivot || A[l] == pivot)
      {
        System.out.println("Only distinct integers allowed - C321");
        System.out.println("students should ignore this if statement");
        System.out.exit(0);
      }
      while (A[f] < pivot) f++;
      while (A[l] > pivot) l--;
      swap (A, f, l);
    }
    return f;
  }

  public static void Quicksort(int A[], int f, int l)
  {
    if (f >= l) return;
    int pivot_index = partition(A, f, l);
    Quicksort(A, f, pivot_index);
    Quicksort(A, pivot_index+1, l);
  }

  // Usage: java QuickSort [integer] ...
  // All integers must be distinct
  public static void main(String argv[])
  {
    int A[] = new int[argv.length];
    for (int i=0 ; i < argv.length ; i++)
      A[i] = Integer.parseInt(argv[i]);
```

```
        Quicksort(A, 0, argv.length-1);

        for (int i=0 ; i < argv.length ; i++) System.out.print(A[i] + " ");
        System.out.println();
    }
}
```