

Quick Sort

The *Quick* sort starts by partitioning the array around a "pivot" value so that all values that are less than or equal to the pivot come before the pivot. For example, assume that the array initially contains the following values:

4 6 7 0 3 9 3 6

If 4 is chosen as the pivot value, then after partitioning, the array might look like this:

3 0 3 4 7 9 6 6

After partitioning the array, the *Quick* sort makes two recursive calls. The first call sorts the portion of the array that contains values less than or equal to the pivot; the second call sorts the rest of the array. The recursion ends when there is just one value to be sorted.

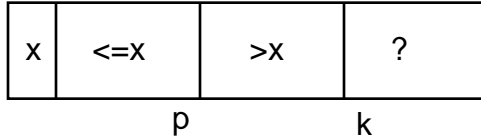
The efficiency of the *Quick* sort depends on the choice of the pivot value. It is least efficient when the value chosen as the pivot is either the smallest or the largest value in the array. In that case, *Quick* sort requires time proportional to n^2 (i.e., it is no more efficient than the Selection or Insertion sort). However, if the number of values that are less than or equal to the pivot is about the same as the number of values that are greater than the pivot, the *Quick* sort is much more efficient: $O(n \log n)$.

The usually way the *Quick* sort is implement is using two partitions

```
void Quick2(apvector<int> & a, int left, int right)
// Standard version of quick sort using one traversing pointer
{
    Swap(a[left] ,a[Median(a,left,right)]); // Chose best pivot value
    int p = left; // Pivot point between <= and > fields

    for (int k = left + 1; k <= right; k++) {
        if (a[k] <= a[left]) { // Go thru unknown field placing
            p++; // values less than pivot value
            Swap(a[p], a[k]); // at the rear of the <= field
        }
    }
    Swap(a[left], a[p]); // Put pivot value in correct spot

    if (left < p-1) {
        Quick2(a, left, p-1);
    }
    if (p+1 < right) {
        Quick2(a, p+1, right);
    }
}
```



x is the pivot value store in the left element of the field. p points to the last element in the <= field, k points to some unknown element. If a[k] is less than the pivot value, then increment the pivot point and swap a[k] with a[p]. This works because a[p] is a still in the greater than field, just now at the back instead on the beginning. The <= grows by one element, which is marked by p.

Here are the benchmarks using this technique:

| | | | |
|----------------|-----------------|------------------|-----------|
| Range 0-30000 | Range 0-300 | Range 0-30 | Range 0-3 |
| Size 5000 | Size 5000 | Size 5000 | Size 5000 |
| Time 0.0833333 | Time 0.13333333 | Time 0.666666666 | Time 4.65 |

Why does the sort degrade?