Quicksort [Knuth]

Description

The basic idea of quicksort is to take one record, say $R[1]$, and to move it to the final position that it should occupy in the sorted file, say position s. While determining this final position, we will also rearrange the other records so that there will be none with greater keys to the left of position s, and none with smaller keys to the right. Thus the file will have been partitioned in such a way that the original sorting problem is reduced to two simpler problems, namely to sort $R[1] \ldots R[s-1]$ and (independently) to sort $R[s+1] \ldots R[N]$. We can apply the same technique to each of these subfiles, until the job is done.

We will achieve the above partitioning into left and right subfiles using a scheme to to R. Sedgewick:

- Keep two pointers, i and j, with $i = 2$ and $j = N$ initially.
- If $R[i]$ is eventually supposed to be part of the left-hand subfile after partitioning (we can tell this by comparing $K[i]$ with $K[1]$), increase i by 1, and continue until encountering a record $R[i]$ that belongs to the right-hand subfile.
- Similarly, decrease j by 1 until encountering a record $R[j]$ belonging to the left-hand subfile.
- If $i < j$, exchange $R[i]$ with $R[j]$; then move on to process the next records in the same way, "burning the candle at both ends" until $i \geq j$. At this point exchange $R[j]$ with $R[1]$ to complete the partitioning.

The quicksort procedure was introduced (and named) by C.A.R. Hoare in 1962.

Example

503	**087**	512	061	908	170	897	275	653	426	154	509	612	677	765	**703**
l	i														jr
503	087	**512**	061	908	170	897	275	653	426	**154**	509	612	677	765	703
l		i								j					r
503	087	**154**	061	908	170	897	275	653	426	**512**	509	612	677	765	703
l		i								j					r
503	087	154	061	**908**	170	897	275	653	**426**	512	509	612	677	765	703
l				i					j						r
503	087	154	061	**426**	170	897	275	653	**908**	512	509	612	677	765	703
l				i					j						r
503	087	154	061	426	170	**897**	**275**	653	908	512	509	612	677	765	703
l						i	j								r
503	087	154	061	426	170	**275**	**897**	653	908	512	509	612	677	765	703
l						i	j								r
503	087	154	061	426	170	**275**	**897**	653	908	512	509	612	677	765	703
l						j	i								r
(**275**	087	154	061	426	170)	**503**	(897	653	908	512	509	612	677	765	703)
l	i				jr										

Now proceed as above with $(R[1] \ldots R[6])$ and put $(8, 16)$ on the stack for future processing. (We put the longer subfile on the stack.)

Algorithm

Records $R[1], \ldots, R[N]$ are rearranged in place; after sorting is complete their keys will be in order $K[1] \leq \cdots \leq K[N]$. An auxiliary stack with at most $\lfloor lgN \rfloor$ entries is needed for temporary storage. This algorithm follows the quicksort partitioning procedure described above.

(a) We assume the presence of artificial keys $K[0] = -\infty$ and $K[N+1] = \infty$ such that

$$K[0] \leq K[a] \leq K[N+1] \quad \text{for } 1 \leq a \leq N.$$

(Equality is allowed.)

(b) Records with equal keys are exchanged, although it is not strictly necessary to do so. (This idea, due to R. C. Singleton, keeps the inner loops fast and helps to split subfiles nearly in half when equal elements are present.

Q1. [Initialize.] If $N = 1$, go to step Q9. Otherwise set the stack empty, and set $l = 1$, $r = N$.

Q2. [Begin new stage.] (We now wish to sort the subfile $R[l], \ldots, R[r]$; from the nature of the algorithm, we have $r \geq l + 1$, and $K[l-1] \leq K[a] \leq K[r+1]$ for $l \leq a \leq r$.) Set $i = l$, $j = r + 1$; and set $k = K[l]$.

Q3. [Compare $K[i]$ and k.] (At this point the file has been rearranged so that

$$K[a] \leq k \quad \text{for } l - 1 \leq a \leq i, \qquad k \leq K[a] \quad \text{for } j \leq a \leq r + 1 \tag{1}$$

and $l \leq i < j$.) Increase i by 1; then if $K[i] < k$, repeat this step. (Since $K[j] \geq k$, the iteration must terminate with $i \leq j$.)

Q4. [Compare k and $K[j]$.] Decrease j by 1; then if $k < K[j]$, repeat this step. (Since $k \geq K[i-1]$, the iteration must terminate with $j \geq i - 1$.)

Q5. [Test i and j] (At this point (1) holds except for $a = i$ and $a = j$; also $K[i] \geq k \geq K[j]$, and $r \geq j \geq i - 1 \geq l$.) If $j \leq i$, interchange $R[l]$ and $R[j]$ and go to step Q7.

Q6. [Exchange.] Interchange $R[i]$ and $R[j]$ and go back to step Q3.

Q7. [Put on stack.] (Now the subfile $R[l] \ldots R[j] \ldots R[r]$ has been partitioned so that $K[a] \leq K[j]$ for $i - 1 \leq a \leq j$ and $k[j] \leq K[a]$ for $j \leq a \leq r + 1$. (Each entry (a, b) on the stack is a request to sort the subfile $R[a] \ldots R[b]$ at some future time.)

- If $r - j \geq j - l > 1$, insert $(j + 1, r)$ on top of the stack, set $r = j - 1$, and go to Q2.
- If $j - l > r - j > 1$, insert $(l, j - 1)$ on top of the stack, set $l = j + 1$, and go to Q2.
- Otherwise
  - If $r - j > 1 \geq j - l$, set $l = j + 1$ and go to Q2; or
  - if $j - l > 1 \geq r - j$, set $r = j - 1$ and go to Q2.

Q8. [Take off stack.] If the stack is nonempty, remove its top entry (a, b), set $l = a$, $r = b$, and return to Q2.

By the way, how did we get here at Q8?

Q9. [End.] Return R.