NAME

quickselect − multiple selection of order statistics and sorting

SYNOPSIS

#include <quickselect.h>

void quickselect(void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *), void (*swapf)(void *, void *, size_t), size_t *pk, size_t nk, unsigned int options);

errno_t quickselect_s(void *base, rsize_t nmemb, rsize_t size, int (*compar)(const void *, const void *, void *), void *context, void (*swapf)(void *, void *, size_t), size_t *pk, size_t nk, unsigned int options);

void QSORT_FUNCTION_NAME(void *base, size_t nmemb, size_t size, int (*compar)(const void , const void *));

errno_t QSORT_S_FUNCTION_NAME(void *base, rsize_t nmemb, rsize_t size, int (*compar)(const void *, const void *, void *), void *context);

DESCRIPTION

The **quickselect** function implements multiple selection of order statistics. Given an array **pk** of **nk** *size_t* elements representing 0−based order statistic ranks, **quickselect** partially orders the array **base** (having **nmemb** elements of size **size**) such that the specified order statistics are in−place in the array. If **pk** is *NULL* or **nk** is 0UL, a full sort of the array is performed.

Function **compar** is provided by the caller, and should return a value less than zero, equal to zero, or greater than zero when the array element pointed to by the first argument is less than, equal to, or greater than the array element pointed to by the second argument.

The **swapf** pointer points to a caller−supplied function which swaps two array elements. If **swapf** is *NULL*, a suitable default function is provided.

The **options** argument affects operation by bitwise or'ing any of the following components:

QUICKSELECT_STABLE
: causes sorting or selection to preserve partial order present in the input. There is a substantial performance penalty; the default operation does not guarantee preservation of partial order.

QUICKSELECT_OPTIMIZE_COMPARISONS
: uses minimum-comparison methods and is suitable if the comparison function **compar** is known to be relatively expensive. The default operation attempts to minimize run−time for simple comparisons.

QUICKSELECT_RESTRICT_RANK
: is used internally and should not be set by the caller.

bits in QUICKSELECT_NETWORK_MASK
: specify the ability to use sorting networks for arrays of size 3 through 12 elements, corresponding to bit 0x01 << size. A sorting network is always used to sort sub−arrays of size 2. Sorting networks for arrays of size 7 or larger are not applicable when **QUICKSELECT_STABLE** is set, and are silently ignored. When **QUICKSELECT_OPTIMIZE_COMPARISONS** is set, only the size 2 sorting network is used; all others are silently ignored. Sorting networks are fast due to low overhead, but are unable to take advantage of pre−existing order in the input (e.g. already-sorted input).

RETURN VALUES

none for quickselect and QSORT_FUNCTION_NAME. If **__STDC_WANT_LIB_EXT1__** is defined with non-zero value when *quickselect.h* is included, quickselect_s and QSORT_S_FUNCTION_NAME are provided, which return zero on normal exection and non-zero if there is an argument error. Arguments **nmemb** and **size** are compared to **RSIZE_MAX**, and **compar** is compared to **NULL**. In addition, the comparison function **compar** is expected to take a third argument, which is provided by the **contect** argument.

ERRORS

If **base** is *NULL*, **nmemb** is 0UL, **size** is 0UL, or **compar** is *NULL*, the global variable *errno* is set to **EINVAL**.

EXAMPLES

```
size_t karray[2];
karray[0] = (nmemb-1UL)/2UL;
karray[1] = nmemb/2UL;
quickselect(base, nmemb, size, compar, NULL, karray, 2UL, 0x07F8U);
```

places the median (**nmemb** odd) or medians (**nmemb** even) in the middle element(s) of the array pointed to by **base.** Refer to the BUGS and CAVEATS section regarding duplicated order statistic ranks.

```
quickselect(base, nmemb, size, compar, NULL, NULL, 0UL, 0x07F8U);
```

sorts the array, and is equivalent to

```
QSORT_FUNCTION_NAME(base, nmemb, size, compar);
```

APPLICATION USAGE

If the macro **QSORT_FUNCTION_NAME** is defined before *quickselect.h* is included when compiling the *quickselect.c* source, a sorting function with the same semantics as **qsort** is generated, using the specified name. A library implementation of **qsort** may be generated by defining **QSORT_FUNCTION_NAME** as **qsort**.

RATIONALE

While many libraries include a standard **qsort** function, those **qsort** implementations may tend to quadratic performance on adverse inputs. Many implementations exhibit poor performance for some types of structured input sequences, such as reverse-sorted or rotated sequences. Most **qsort** implementations provide no guarantee of stablilty (in the sense of preservation of partial order), and there is no means of providing an optimized element swapping function or means to adjust the algorithm to compensate for expensive comparisons. Few libraries provide a function for selection of order statistics. Those libraries that do provide a selection function usually only permit selection of a single order statistic per function call.

BUGS and CAVEATS

Array **pk** may be sorted by **quickselect** and therefore initially unsorted order statistic ranks may be permuted by a call to **quickselect**. It is recommended (but not required) that the order statistics array **pk** be supplied in sorted order.

If array **pk** contains duplicated ranks, those duplicates will be ignored during processing and will be grouped together by sorting after processing. This may be expensive if a large number of order statistics are specified and there is at least one duplicate. It is recommended (but not required) that the order statistics array **pk** contain no duplicates.

quickselect has expected and worst−case linear complexity for finding a single order statistic. Worst−case non-stable selection of multiple order statistics is linearithmic. **quickselect** has expected and worst−case linearithmic complexity for non-stable sorting.

When stable sorting or selection is specified by setting **QUICKSELECT_STABLE**, selection becomes linearithmic and sorting becomes $O(N \log^2 N)$ due to additional data movement (the complexity of comparisons is unchanged). Sorting and selection remain in−place; no additional size−related memory is required. However, if stable sorting is required and additional memory is available, an alternative means of sorting may be faster.

Compiled library code might have been built with **QUICKSELECT_STABLE** and/or **QUICKSELECT_NETWORK_MASK** set to values other than those which appear in *quickselect.h*. If the library was built without the option to sort or select while maintaining partial order stability, specifying **QUICKSELECT_STABLE** in *options* will not be effective. Consult local documentation if available or examine the build options used to create the library object files using *what* or *ident*. Similarly, support for sorting network sizes may have been limited when the library object files were built. Specifying bits from

QUICKSELECT_NETWORK_MASK in *options* which are not supported in the library object code will result in the use of insertion sort for small sub−arrays of those sizes.

FUTURE DIRECTIONS

none

SEE ALSO

qsort, what, ident

CHANGE HISTORY

Function implementation initial version June 2016. Implementation backward−compatible updates through November 2017.

Manual page initial version January 2017. Latest manual page update November 2017.

AUTHOR

Bruce Lilly <bruce.lilly@gmail.com>

LICENSE

The following license covers this software, including makefiles and documentation:

You may send bug reports to bruce.lilly@gmail.com with subject "quickselect".