NAME

qsort, **qsort_b**, **qsort_r**, **heapsort**, **heapsort_b**, **mergesort**, **mergesort_b** - sort functions

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <stdlib.h>

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

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

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

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

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

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

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

#define __STDC_WANT_LIB_EXT1__ 1

errno_t
qsort_s(*void *base, rsize_t nmemb, rsize_t size, int (*compar)(const void *, const void *, void *),*
*void *thunk*);

DESCRIPTION

The **qsort**() function is a modified partition-exchange sort, or quicksort. The **heapsort**() function is a modified selection sort. The **mergesort**() function is a modified merge sort with exponential search

intended for sorting data with pre-existing order.

The **qsort**() and **heapsort**() functions sort an array of *nmemb* objects, the initial member of which is pointed to by *base*. The size of each object is specified by *size*. The **mergesort**() function behaves similarly, but *requires* that *size* be greater than "sizeof(void *) / 2".

The contents of the array *base* are sorted in ascending order according to a comparison function pointed to by *compar*, which requires two arguments pointing to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

The **qsort_r**() function behaves identically to **qsort**(), except that it takes an additional argument, *thunk*, which is passed unchanged as the last argument to function pointed to *compar*. This allows the comparison function to access additional data without using global variables, and thus **qsort_r**() is suitable for use in functions which must be reentrant. The **qsort_b**() function behaves identically to **qsort**(), except that it takes a block, rather than a function pointer.

The algorithms implemented by **qsort**(), **qsort_r**(), and **heapsort**() are *not* stable, that is, if two members compare as equal, their order in the sorted array is undefined. The **heapsort_b**() function behaves identically to **heapsort**(), except that it takes a block, rather than a function pointer. The **mergesort**() algorithm is stable. The **mergesort_b**() function behaves identically to **mergesort**(), except that it takes a block, rather than a function pointer.

The **qsort**() and **qsort_r**() functions are an implementation of C.A.R. Hoare's "quicksort" algorithm, a variant of partition-exchange sorting; in particular, see D.E. Knuth's *Algorithm Q*. **Quicksort** takes O N lg N average time. This implementation uses median selection to avoid its O N**2 worst-case behavior.

The **heapsort**() function is an implementation of J.W.J. William's "heapsort" algorithm, a variant of selection sorting; in particular, see D.E. Knuth's *Algorithm H*. **Heapsort** takes O N lg N worst-case time. Its *only* advantage over **qsort**() is that it uses almost no additional memory; while **qsort**() does not allocate memory, it is implemented using recursion.

The function **mergesort**() requires additional memory of size *nmemb * size* bytes; it should be used only when space is not at a premium. The **mergesort**() function is optimized for data with pre-existing order; its worst case time is O N lg N; its best case is O N.

Normally, **qsort**() is faster than **mergesort**() is faster than **heapsort**(). Memory availability and pre-existing order in the data can make this untrue.

The **qsort_s**() function behaves the same as **qsort_r**(), except that:

- The order of arguments is different
- The order of arguments to *compar* is different
- If *nmemb* or *size* are greater than RSIZE_MAX, or *nmemb* is not zero and *compar* is NULL or *size* is zero, then the runtime-constraint handler is called, and **qsort_s**() returns an error. Note that the handler is called before **qsort_s**() returns the error, and the handler function might not return.

RETURN VALUES

The **qsort**() and **qsort_r**() functions return no value. The **qsort_s**() function returns zero on success, non-zero on error.

The **heapsort**() and **mergesort**() functions return the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

EXAMPLES

A sample program that sorts an array of *int* values in place using **qsort**(), and then prints the sorted array to standard output is:

```
#include <stdio.h>
#include <stdlib.h>

/*
 * Custom comparison function that compares 'int' values through pointers
 * passed by qsort(3).
 */
static int
int_compare(const void *p1, const void *p2)
{
        int left = *(const int *)p1;
        int right = *(const int *)p2;

        return ((left > right) - (left < right));
}

/*
 * Sort an array of 'int' values and print it to standard output.
 */
```

```
int
main(void)
{
        int int_array[] = { 4, 5, 9, 3, 0, 1, 7, 2, 8, 6 };
        size_t array_size = sizeof(int_array) / sizeof(int_array[0]);
        size_t k;

        qsort(&int_array, array_size, sizeof(int_array[0]), int_compare);
        for (k = 0; k < array_size; k++)
                printf(" %d", int_array[k]);
        puts("");
        return (EXIT_SUCCESS);
}
```

COMPATIBILITY

The order of arguments for the comparison function used with **qsort_r**() is different from the one used by **qsort_s**(), and the GNU libc implementation of **qsort_r**(). When porting software written for GNU libc, it is usually possible to replace **qsort_r**() with **qsort_s**() to work around this problem.

qsort_s() is part of the *optional* Annex K portion of ISO/IEC 9899:2011 ("ISO C11") and may not be portable to other standards-conforming platforms.

Previous versions of **qsort**() did not permit the comparison routine itself to call **qsort**(*3*). This is no longer true.

ERRORS

The **heapsort**() and **mergesort**() functions succeed unless:

[EINVAL] The *size* argument is zero, or, the *size* argument to **mergesort**() is less than "sizeof(void *) / 2".

[ENOMEM] The **heapsort**() or **mergesort**() functions were unable to allocate memory.

SEE ALSO

sort(1), radixsort(3)

Hoare, C.A.R., "Quicksort", *The Computer Journal*, 5:1, pp. 10-15, 1962.

Williams, J.W.J, "Heapsort", *Communications of the ACM*, 7:1, pp. 347-348, 1964.

Knuth, D.E., "Sorting and Searching", *The Art of Computer Programming*, Vol. 3, pp. 114-123, 145-149, 1968.

McIlroy, P.M., "Optimistic Sorting and Information Theoretic Complexity", *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1992.

Bentley, J.L. and McIlroy, M.D., "Engineering a Sort Function", *Software--Practice and Experience*, Vol. 23(11), pp. 1249-1265, November 1993.

STANDARDS

The **qsort**() function conforms to ISO/IEC 9899:1990 ("ISO C90"). **qsort_s**() conforms to ISO/IEC 9899:2011 ("ISO C11") K.3.6.3.2.

HISTORY

The variants of these functions that take blocks as arguments first appeared in Mac OS X. This implementation was created by David Chisnall.

In FreeBSD 14.0, the prototype of **qsort_r**() was updated to match POSIX.