```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <pthread.h>


struct Params
{
   int *start;
   size_t len;
   int depth;
};

// only used for synchronizing stdout from overlap.
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

// forward declare our thread proc
void *merge_sort_thread(void *pv);


// a simple merge algorithm. there are *several* more efficient ways
//  of doing this, but the purpose of this exercise is to establish
//  merge-threading, so we stick with simple for now.
void merge(int *start, int *mid, int *end)
{
   int *res = malloc((end - start)*sizeof(*res));
   int *lhs = start, *rhs = mid, *dst = res;

   while (lhs != mid && rhs != end)
      *dst++ = (*lhs <= *rhs) ? *lhs++ : *rhs++;
```

```
    while (lhs != mid)
        *dst++ = *lhs++;

    while (rhs != end)
        *dst++ = *rhs++;

    // copy results
    memcpy(start, res, (end - start)*sizeof(*res));
    free(res);
}

// our multi-threaded entry point.
void merge_sort_mt(int *start, size_t len, int depth)
{
    if (len < 2)
        return;

    if (depth <= 0 || len < 4)
    {
        merge_sort_mt(start, len/2, 0);
        merge_sort_mt(start+len/2, len-len/2, 0);
    }
    else
    {
        struct Params params = { start, len/2, depth/2 };
        pthread_t thrd;

        pthread_mutex_lock(&mtx);
        printf("Starting subthread...\n");
        pthread_mutex_unlock(&mtx);
```

```
        // create our thread
        pthread_create(&thrd, NULL, merge_sort_thread, &params);

        // recurse into our top-end parition
        merge_sort_mt(start+len/2, len-len/2, depth/2);

        // join on the launched thread
        pthread_join(thrd, NULL);

        pthread_mutex_lock(&mtx);
        printf("Finished subthread.\n");
        pthread_mutex_unlock(&mtx);
    }

    // merge the paritions.
    merge(start, start+len/2, start+len);
}

// our thread-proc that invokes merge_sort. this just passes the
//  given parameters off to our merge_sort algorithm
void *merge_sort_thread(void *pv)
{
    struct Params *params = pv;
    merge_sort_mt(params->start, params->len, params->depth);
    return pv;
}

// public-facing api
void merge_sort(int *start, size_t len)
{
```

```
    merge_sort_mt(start, len, 4); // 4 is a nice number, will use 7 threads.
}

int main()
{
    static const unsigned int N = 2048;
    int *data = malloc(N * sizeof(*data));
    unsigned int i;

    srand((unsigned)time(0));
    for (i=0; i<N; ++i)
    {
        data[i] = rand() % 1024;
        printf("%4d ", data[i]);
        if ((i+1)%8 == 0)
            printf("\n");
    }
    printf("\n");

    // invoke our multi-threaded merge-sort
    merge_sort(data, N);
    for (i=0; i<N; ++i)
    {
        printf("%4d ", data[i]);
        if ((i+1)%8 == 0)
            printf("\n");
    }
    printf("\n");

    free(data);
```

```
    return 0;


}
```