Homework 1: Mergesort in C

You must implement mergesort in C. The source code you turn in will be called `mergesort.c`. The function that does the sorting *must* have the following prototype, so that it can be called from external code:

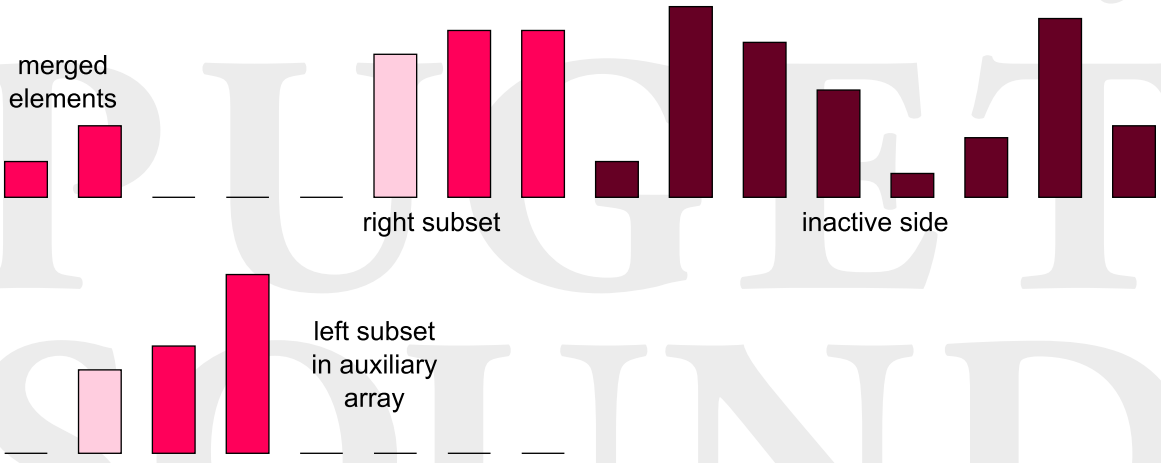
- `void mergeSort(int *array, int size)`

This function should just sort (recursively). It can call any subfunctions that are needed, but it absolutely will not print anything to the screen.

When you run your program independently, the array's size n will be determined by the first (and only) command-line argument. If no argument is included, the program must print a polite error message and terminate. The array will be filled with random numbers r_i such that $0 \leq r_i < n$. If $n \leq 100$, the program must print out the array both before and after sorting. Regardless, the program will print out the number of milliseconds the sort took.

Recall that mergesort is a recursive algorithm. It consists of four steps:

1. Divide the active area into two equal regions: a "left half" and a "right half".
2. Recurse on the left half.
3. Recurse on the right half.
4. Merge the left and right halves together into a big sorted region.



Step #4 is the complicated step. You will need to have an auxiliary array that is half the size of the overall array. The left half of the array is copied to the auxiliary. Then look through both the auxiliary array and the right half of the array, each time moving the first element of one or the other into the next place within the original array. Done correctly, this algorithm's time complexity is $O(n \log n)$, and its space complexity is $O(n)$.

You are responsible for making sure that the algorithm works correctly: it should be fast enough to work with million-element arrays, and all the elements after sorting should be identical to the elements before sorting. (One way to check if the elements are the same is

to exclusive-or all the elements together. If the elements are the same, this final exclusive-or will be identical before and after it is sorted.)

You are also responsible for ensuring that memory allocation works correctly. The proper amount of memory needs to be allocated for both the array itself and the auxiliary scratch space. If you allocate the auxiliary space from the heap, it should only be done once and should be properly freed. Or you may allocate it from the stack.

Please remember to comment your code. There should be comments at the top with your name and information about the code. Every function needs its own comment, and comments should be spaced liberally through the rest of the code. Use as few global variables as possible.

Output might look like this:

```
$ a.out
I'm sorry.  You must include the length of the array.

$ a.out 10
Doing sort on array of size 10...
Before:  [4, 5, 7, 6, 6, 0, 9, 9, 4, 0]
After:  [0, 0, 4, 4, 5, 6, 6, 7, 9, 9]
Time elapsed:  0 ms

$ a.out 1000000
Doing sort on array of size 1000000...
Time elapsed:  273 ms
```