HW 6: Mergesort

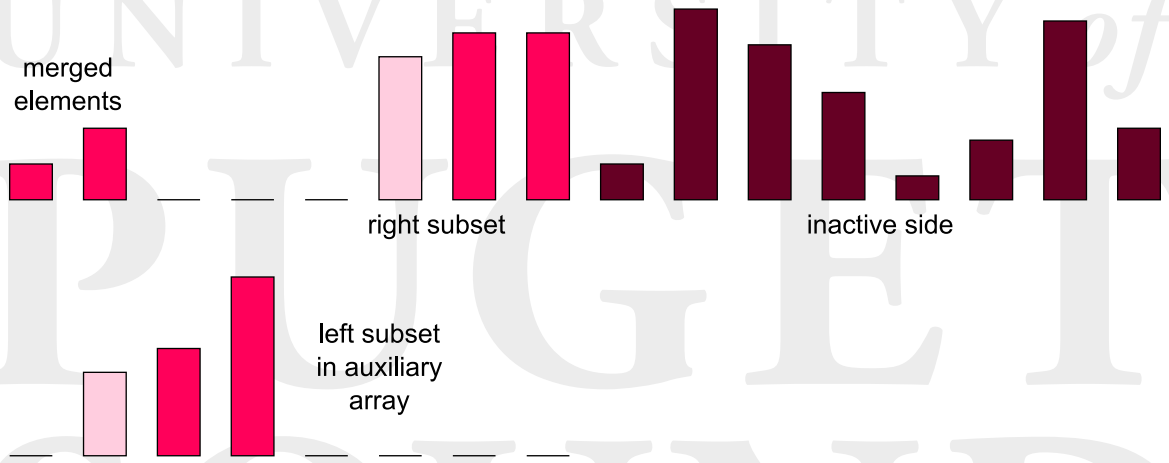
You must implement a class which sorts using mergesort. It should inherit from the abstract Sorter class, and be called `MergeSorter`.

The `main()` method should be nearly the same as the one from previous sorting assignments. It will ask for a number, and then time how long (in ms) it takes to sort an array of that length:

```
Testing mergesort.
Please enter the array size:  50000
Array of size 50000 took 91 ms to sort.
```

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.



Since you must extend the abstract `Sorter class`, the public `sort()` method must take only one argument: the array to be sorted. However, because this algorithm recurses on subregions of the array, it makes your task easier if you make a private `sort()` function that takes four arguments. These include the array itself, and the left and right bounds on which to act. The fourth argument should be an auxiliary array that is half the size of the full array. The public `sort()` method will thus allocate the new array, and call the private method. You should avoid the impulse to make the auxiliary array into a field for the class. Not only will this promote cleaner code, but it will make it possible for two parallel threads to call your method at the same time.

The merge step is the one that requires the auxiliary array. Copy the sorted left half into this array, and then compare the first left element with the first right element. The lesser

one will be copied into the proper location in the full array. Continue this until one of the halves is exhausted.

Mergesort's main fault is that its space complexity is linear, due to the auxiliary array. (Recursion adds a bit to its memory requirements, but this is dwarfed by the auxiliary array.) However, mergesort's time complexity is linearithmic. The merge step is linear. Proving that the algorithm is at worst quadratic is easy, since the merge will be done a linear number of times. However, successive merges are smaller and smaller, such that the amount of calculation done by all the merges is $O(n \log n)$.

As before, take care that the end user sees no exceptions. Further, you should suppress compiler warnings that are printed during compilation.