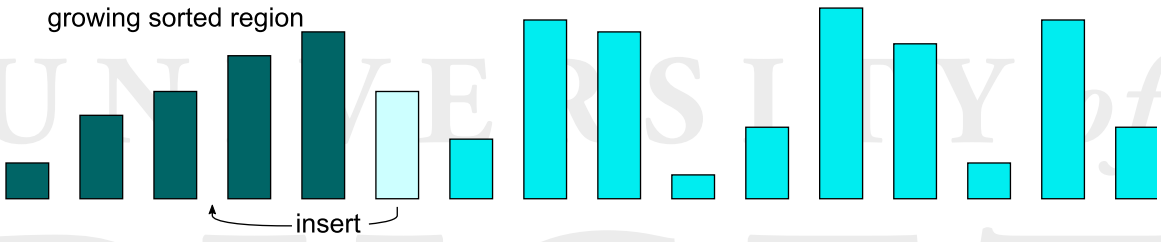
Lab 6: Insertion Sort

You must implement insertion sort, again inheriting from the abstract `Sorter` class. Your class will be called `InsertionSorter`.

The `main()` method of each the class will be nearly the same as for the previous assignments. It will ask the user for a number, and use the `timeSort()` method to count how many milliseconds it took to sort the data:

```
Testing insertion sort.
Please enter the array size: 50000
Array of size 50000 took 9961 ms to sort.
```

Insertion sort works by building up a sorted region at the beginning of the array. Starting with the second element, it finds where the element should be in the sorted region to its left. It then inserts the element into the proper location, moving all of the other elements over. It needs to perform this step $(n - 1)$ times. Since the time taken to do each step is linear, the overall algorithm is quadratic.



The fastest way to find where an element should go in the sorted region is to do a "binary sort". First look at the element halfway through the sorted region, and decide whether the active element should go before or after it. Then, depending on the previous result, do the same comparison with the element at the one-quarter point or the three-quarters point. Continue this until you find the right location. However, for the purposes of the lab, it will suffice if you just compare elements from the end of the sorted region, until you find the right place. (Be sure to place the active element to *after* any tied elements in the sorted region, to make the sort *stable*.)

The best way to move the active element is to store it in a temporary variable, like when you swap. Then move all the necessary elements over, and place the active element in its place. Swapping the elements one at a time is less efficient, and turns the insertion sort into a variation called *gnome sort*.

As before, be careful not to let any exceptions slip out to be seen by the end user. You may think your code is foolproof, but fools are a remarkably resourceful lot.