Tutorial #6. Sorting

Theoretical part

Sorting allows us to do multiple tasks:

- Sorted elements can be found in O(log(n)) for search using binary/interpolation search (or even a place they should be inserted)
- They allow us to select linked areas of data in O(n) time (SELECT * FROM T where C1 > V1).
- They allow us to make cool math (Principal Component Analysis include sorts, create a polygon of points, compression with loss algorithms involve sorting)
- They allow us to represent data for humans.

Faster sorts

There's a class of sorts that allow us to do it in **T(n*log(n))** time: **quick sort**, that involves recursive strategy, **heap sort** and **merge sort**. Some of them provide O(n*log(n)), some – Ω(n*log(n)).

O(n)-sorts

Unlikely bubble and insertion sort, there's a class of algorithms that does not involve comparison. All O(n) sorts are non-comparative, because it is proven, that comparative sorts cannot perform better then Ω(n*log(n)). We already know **counting sort**, it takes O(n+k). There are also two well-known non-comparative sorts:

- **Bucket sort (Explain)**. Idea is to
    - Split the range of values into buckets [1-50, 51-100, …].
    - Then **Scatter**: place items in the buckets according to the value (value / $N_{buckets}$).
    - Sort each non-empty bucket.
    - Then **Gather**: collect values back into array
        - Worst case: **$O(n^2)$**,
        - best/average case **E(n+k) (use count sort inside the bucket)**
    - Study this sort and try to implement it for integers.
- **Radix sort (Just mention)**. Similar idea: use digits as buckets.

Practical part (for Students)

1) Implement **insertion sort** using **Comparator** object for case insensitive string sorting.
2) Embed your implementation into **MyFramework.sort**(…) method, reuse the *Comparable.compareTo(…)* in comparator object.
3) Run tests in **MyFramework** class. See the output.
4) Uncomment line 22 in **MyFrameworkTests.** Measure **select** operation speed before and after this operation.
5) *Try to implement O(n) sort