Data Structures and Algorithms - Lab 1: Dynamic Array

Implement in C++ the given **container** (ADT) using a given representation and a **dynamic array** as a data structure. You are not allowed to use the *vector* from STL or from any other library.

Obs:

- Since your implementation will use dynamic allocation, it is a good practice to implement a destructor, copy constructor and assignment operator as well (even if they are not on the interface).
- You are not allowed to use the functions *memcpy* and *realloc*, because it is not safe to use *memcpy* and *realloc* on memory that was allocated with *new*. Also, if the memory location contains objects, undefined behavior can occur. The implementation might still work with these functions, but it is not a good practice to use them.
- If you need auxiliary functions, fell free to add them to the interface of the ADT, but make them private.

1. **ADT Matrix** – represented as a sparse matrix, using a dynamic array of triples <line, column, value> (value ≠ 0), ordered lexicographically considering the <line, column> of every element.
2. **ADT Matrix** – represented as a sparse matrix, compressed sparse column representation using dynamic arrays.
3. **ADT Matrix** – represented as a sparse matrix, compressed sparse line representation using dynamic arrays.
4. **ADT Bag** – represented using a dynamic array of <element, frequency> pairs (or two dynamic arrays).
   For example, the bag [5, 10, -1, 2, 3, 10, 5, 5, -5] will be represented as [(5,3), (10, 2), (-1, 1), (2, 1), (3, 1), (-5, 1)]
5. **ADT Bag** – represented as a dynamic array of frequencies.
   For example, the bag [5, 10, -1, 2, 3, 10, 5, 5, -5] will be represented as [1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 3, 0, 0, 0, 0, 2], built in the following way:
   - the interval of values [-5, 10] is translated into the interval [0, 16] of positions
   - at position 0 we have the frequency of -5 (minimum element), on position 1 we have the frequency of -4, …, on position 15 we have the frequency of 10 (maximum element).
6. **ADT Bag** – represented as a dynamic array of unique elements (U) and a dynamic array of positions (P) in U of the elements from the Bag.
   For example, the bag [5, 10, -1, 2, 3, 10, 5, 5, -5] will be represented as:
   U = [5, 10, -1, 2, 3, -5]
   P = [0, 1, 2, 3, 4, 1, 0, 0, 5]

7. **ADT SortedBag** – having elements of type **TComp**, represented using a dynamic array of <element, frequency> pairs (or two dynamic arrays), sorted using a relation on the elements.
8. **ADT SortedBag** – having elements of type **TComp**, sorted using a relation on the elements and stored in a dynamic array.
9. **ADT SortedSet** – having elements of type **TComp**, sorted using a relation on the elements and stored in a dynamic array.
10. **ADT Set** – represented as a dynamic array of elements.
11. **ADT Set** – represented as a dynamic array of Boolean values. For example the set {5, 1, -4, 0, 8} can be represented as an array of 13 elements where position 0 corresponds to element -4, position 1 corresponds to element -3, …, position 12 corresponds to element 8: [true, false, false, false, true, true, false, false, false, true, false, false, true].
12. **ADT Queue** – represented on a circular dynamic array + **ADT Stack** – represented on a dynamic array.
13. **ADT Map** – represented as a dynamic array of <key, value> pairs.
14. **ADT Sorted Map** – represented as a dynamic array of <key, value> pairs, sorted using a relation over the keys.