Sort Algorithms

1. BucketSort

Given: a sequence S, formed of *n* pairs (key, value),
keys are integer numbers from an interval ϵ [0, N-1]
Sort S based on the keys.

EX: S: (7, d) (1, c) (3, b) (7, g) (3, a) (7, e)
⇨ (1, c) (3, b) (3, a) (7, d) (7, g) (7, e)

Assume that the sequence is already implemented, and it has the following operations:

- initEmpty(sequence)
- empty (sequence): boolean
- first (sequence): element
- remove First(sequence)
- addLast(sequence, element)

2. Lexicographic Sort

Given: a sequence S of tuples.
Sort S in a lexicographic order.

EX: (7, 4, 6) (5, 1, 5) (2, 4, 6) (2, 1, 4) (3, 2, 4)
⇨ (2, 1, 4) (2, 4, 6) (3, 2, 4) (5, 1, 5) (7, 4, 6)

Assume that we have:

- R_i – a relation that can compare 2 tuples considering the ith dimension.
- *stableSort(S, r)* – a stable sorting algorithm that uses a relation to compare the elements.

3. Radix Sort

- A variant of the lexicographic sort, which uses as a stable sorting algorithm Bucketsort → every element of the tuples has to be a natural number from some interval [0, N-1].
- Complexity: Θ (d * (n + N))

4. Linked-list probles

Write a subalgorithm to merge two sorted singly-linked lists. Analyze the complexity of the operation.

Version A:
Do not destroy the two existing lists: the result is a third list (we have to copy the existing nodes).

Version B:
Version B: Do not keep the two existing lists, the result will contain the existing nodes (but the links are changed)