Lab 06 - Implementing a Custom List Sort

Objective: This assignment challenges you to implement a custom sorting algorithm specifically designed for a std::list of Person objects.

Instructions:

1. **Define the Person Class:**
    - Create a class named Person with the following public members:
        - string name: To store the name of the person.
        - int age: To store the age of the person.
    - Overload the output stream operator (operator<<) for the Person class to print the name and age in a readable format (e.g., "Name: Homer Simpson, Age: 39").
    - Overload the less-than operator (operator<) for the Person class to compare two Person objects based on their name in ascending alphabetical order. This will be helpful for comparison within your sorting algorithm.

2. **Implement the Custom Sort Function:**
    - Write a function named sortListByName(std::list<Person>& personList) that takes a std::list<Person> as a reference.
    - Implement the following sorting algorithm within this function:
        - **Iteratively find the lexicographically largest name** in the unsorted portion of the list.
        - **Move the node containing the largest name to the end** of the currently unsorted portion of the list.
        - **Repeat this process**, considering a progressively smaller unsorted portion (from the beginning up to the element just before the previously placed largest element), until the entire list is sorted in ascending order by name.
    - **Hint:** To move a node within a std::list, you will need to use iterators to locate the node and potentially use the splice() member function to efficiently move the node to the desired position. A less efficient approach would consider erasing the node and inserting it at the last position of the unsorted region.

3. **Test Your Solution:**
    - In your main function, create a std::list<Person> named simpsons and initialize it with the following data:

```
std::list<Person> simpsons = {
   {"Homer Simpson", 39},
   {"Marge Simpson", 36},
   {"Bart Simpson", 10},
   {"Lisa Simpson", 8},
   {"Maggie Simpson", 1}
};
```

    - Print the simpsons list before sorting.
    - Call your sortListByName function to sort the simpsons list.
    - Print the simpsons list after sorting to verify that it is sorted in ascending order by name.

Hints:

- You will need to iterate through the unsorted portion of the list in each step to find the largest name.
- Keep track of the boundary between the sorted and unsorted portions of the list.
- Be mindful of how splice() affects iterators. You might need to get new iterators after a splice operation.
- Consider edge cases like an empty list or a list with only one element.

Note

The **splice** function transfers elements from one list to another (or within the same list). In the case when only one element is transferred from source to destination, the syntax for **splice** is:

```
void splice(iterator destPosition, list& sourceLst, iterator sourcePosition);
```

where

destPosition: The position in the destination list where the elements should be inserted.

sourceLst: The source list from which elements are to be transferred.

sourcePosition: The iterator pointing to the source element to be transferred.