MODULE 7 PROJECT: SEARCHING & SORTING

Learning Objectives:

CLO 3. Develop problem solving skills by implementing data structures.

CLO 4. Compare advantages and disadvantages of different data structure implementations.

Function Implementations

Files to Modify: algorithms.py

Directions:

1. Implement the function `linear_search(a, x)` which uses the linear search algorithm to return the index of element `x` if it is found in the `ArrayList` object `a`. If `x`, is not found in `a`, then the function returns -100.
2. Implement the function `binary_search(a, x)` which uses the binary search algorithm to return the index of element `x` if it is found in the *sorted* `ArrayList` object `a`. If `x`, is not found in `a`, then the function returns -100.
3. Implement the function `_merge(a0, a1, a)` which overwrites `ArrayList` object `a` by merging the elements of `ArrayList` object `a0` and `ArrayList` object `a1` in increasing order.
4. Implement the function `merge_sort(a)` which uses the merge-sort algorithm to sort the `ArrayList` object `a`.
5. Implement the helper function `_quick_sort_f(a, start, end)` which uses the quick-sort algorithm with the first element as pivot to sort the `ArrayList` object `a`.
6. Implement the helper function `_quick_sort_r(a, start, end)` which uses the quick-sort algorithm with a random element as pivot to sort the `ArrayList` object `a`.

   NOTE: You may introduce any additional helper functions your quick sort functions might need, as long as you do not change the parameters defined for each function.

Extra-Credit Worth: 5% applied towards projects category grade

I. **Changes to `Book`**

Files to modify: Book.py

Directions:

Change the implementation of the `Book` class so that `Book` instances are compared on the basis of title rather than rank. More specifically, suppose that `book1` and `book2` are `Book` instances. Overload the $<, >, \leq, \geq$ and $=$ operators so that

- `book1` < `book2` returns True if `book1` has a title that comes before the title of `book2`, when listed in alphabetical order Aa - Zz (i.e., regardless of letter case) and False otherwise.
- `book1` > `book2` returns True if `book1` has a title that comes after the title of `book2`, when listed in alphabetical order Aa - Zz (i.e., regardless of letter case) and False otherwise.
- `book1 <= book2` returns True if `book1` has a title that comes before, or is exactly the same as, the title of `book2`, when listed in alphabetical order Aa - Zz (i.e., regardless of letter case) and False otherwise.
- `book1 >= book2` returns True if `book1` has a title that comes after, or is exactly the same as, the title of `book2`, when listed in alphabetical order Aa - Zz (i.e., regardless of letter case) and False otherwise.
- `book1` == `book2` returns True if the titles of the books are the same and their keys are the same. If the keys are not the same, or if the titles are different, it returns False.

II. **Changes to Bookstore System**

File to modify: BookStore.py

Directions:

1. Create a `BookStore` method called `sort_catalog(s)` that sorts the `ArrayList` instance `self.bookCatalog` using the sorting algorithm determined by parameter `s`.
   - If `s` = 1, the method uses the merge-sort algorithm. Returns True.
   - If `s` = 2, the method uses the quick-sort algorithm with first element as pivot. Returns True.
   - If `s` = 3, the method uses the quick-sort algorithm with a randomly chosen element as pivot. Returns True.
   - If any other value is given for `s`, the method returns False.

   Once the book catalog has been sorted, the method must print the message:

   ```
   "Sorted {self.bookCatalog.size()} books in {elapsed_time} seconds."
   ```

2. Create a `BookStore` method called `display_catalog(n)` that displays the first `n` books of the book catalog.

III. Changes to Main

Files to modify: main.py

Directions:

1. Modify `menu_bookstore_system()` so that it includes the following new options:
   - sorting the book catalog
   - displaying the first n books of the catalog

   Your new list of options should read as follows:

   ```
   s FIFO shopping cart
   r Random shopping cart
   1 Load book catalog
   2 Remove a book by index from catalog
   3 Add a book by index to shopping cart
   4 Remove from the shopping cart
   5 Search book by infix
   6 Get cart best-seller
   7 Add a book by key to shopping cart
   8 Add a book by title prefix to shopping cart
   9 Search best-sellers with infix
   10 Sort the catalog
   11 Display the first n books of catalog
   0 Return to main menu
   ```

2. If the user chooses option 10, the system should prompt for an algorithm to use, in the following manner:

   ```
   Choose an algorithm:
       1 - Merge Sort
       2 - Quick Sort (first element pivot)
       3 - Quick Sort (random element pivot)
   Your selection: <Insert integer>
   ```

   If the user's selection is not 1, 2, or 3, the message `"Invalid algorithm"` is displayed.

3. If the user chooses option 11, the system should prompt for a number of books to display in the following manner:

   ```
   Enter the number of books to display: <insert integer>
   ```

SUBMISSION PROCESS

Submit to CodePost

- `algorithms.py`

If you complete the extra-credit, submit the following files to the **extra-credit folder** on CodePost.

- `algorithms.py`
- `Book.py`
- `main.py`