* **Use Cases:**

Searching/displaying book by title/Purely displaying books:

* Test group: Inputting book titles as strings

1. Entering in an existing book title
2. Parameter: “The Great Gatsby”
3. Result: Program successfully displays the book’s title, author, isbn number, number of pages, and its availability.
4. Entering in a title that isn’t stored in the book library
5. Parameter: “Michael Jordan and the prisoner of Alcatraz”
6. Result: Program tells the user that the book with the inputted title doesn’t exist in the library.
7. Entering in an existing title with inconsistent casing
8. Parameter: “LoRD OF THe FLIEs”
9. Result: Program successfully shows the user the book’s information.

Issuing books:

* Test group: Entering in the titles of books as strings and inputting a listed integer that may or may not correspond to an existing student.

1. Enter an existing book title, and a number corresponding to an existing student
2. Parameter: “1984” (book title) and option student 2 (David Jones)
3. Result: Program successfully issued the book to the student.
4. Enter a book title of a book that’s not available
5. Parameter: “1984” (book title)
6. Result: Program correctly warns user that the book “1984” has already been checked out, and aborts the check out process.
7. Enter an existing book, but enter an integer that doesn’t correspond to a student
8. Parameter: “Night” (Book title). Student options 0, 8, 9, and 5.
9. Result: Correctly tells user that the first 3 options are out of the range 1 to 7, and then correctly selects student option 5 at the end.

Returning books:

* Test group: Entering in integers representing the listed options of records of issued books and the respective borrowing students.

1. Enter an integer that corresponds to an existing issued book record
2. Parameter: Select option 3 for the issued book records
3. Result: Correctly returns book.
4. Enter an integer that doesn’t correspond to a listed issued book record
5. Parameter: Selects options 0, 4, and 1.
6. Result: Correctly re-prompts user for input for the first two inputs, and then finally returns the book at option 1 since the range is 1-2.
7. Trying to return books when there have been no books issued
8. Parameter: Selecting option 3 (return book) when there are no books issued.
9. Result: Warns the user that they can’t return books since none are issued.

Adding Books:

* Test group: Entering in book titles, authors, and ISBN numbers as strings. Then entering the respective number of pages as integers.

1. Properly entering in a new title to the library. Then entering in the author, isbn, and number of pages
2. Parameter: “A Wrinkle in Time” (book title), “Madeleine L’Engle” (author), “978-0312367541” (ISBN number), “256” (number of pages”.
3. Result::Successfully added the book to the library.
4. Entering in a title of a book already in the library with identical casing
5. Parameter: All parameters remained unchanged from part a.
6. Result: Successfully warns the user that the book couldn’t be added.because it’s a duplicate title.
7. Entering in a book of a book already in the library with different casing
8. Parameter: “a WRINKLE IN TIME” (Book title), all other parameters unchanged.
9. Result: Successfully tells the user that the book couldn’t be added.

Removing Books:

* Test group: Entering in book titles as strings.

1. Deleting an existing book in the library
2. Parameter: “To Kill a Mockingbird” (book title).
3. Result: Successfully removed the existing book from the library.
4. Deleting a book that doesn’t exist in the library
5. Parameter: ”Johannas and the pursuit of knowledge!” (book title).
6. Result: Successfully warns the user that the book couldn’t be removed since it wasn’t a book in the library.
7. Deleting a book that’s currently issued/checked out
8. Parameter: “The things they carried” (book title).
9. Result: Successfully warns the user that the book is currently checked out so it can’t be removed from the library.

Adding Students:

* Test group: Entering first name, last name, and student id as strings

1. Entering a new student
2. Parameter: “Johannas” (first name), “Slick” (last name), “76995” (student id).
3. Result: Successfully added the student.
4. Entering a new student with an existing student id
5. Parameter: “Jenny” (first name), “Corden” (last name, “76995” (student id)
6. Result: Successfully aborts process, telling user a student with the inputted student id already exists in the library.

Removing Students:

* Test group: Entering an integer for a menu choice representing a listed student

1. Deleting an existing student
2. Parameter: Select student option “5”.
3. Result: Successfully removes the student from the library.
4. Deleting a student that isn’t registered in the library.
5. Parameter: Select student options 0 and 7
6. Result: Correct re-prompts input when user selects out of range option (0), and successfully deletes student at the in range option (7).

* **Algorithms:**

Merge Sort:

Time Complexity and Big O Analysis: The merge sort algorithm’s runtime complexity is , and the worse, best, and average cases are all of the same. In terms of the differences in all of the cases, the amount of operations across all cases remains approximately the same and consistent. Of course, the way in which the data is ordered or given to the function will minorly affect the real number of operations that it takes to sort the vector. However, overall as the number of inputs increase, the amount of operations required will continue to be denote by the function .

Empirical Tests:

1. Test Group: Sorting a vector of randomly generated strings with 10 characters.
2. Input: 1000 elements. Runtime is approx. 47 milliseconds.
3. Input: 10000 elements. Runtime is approx. 513 milliseconds.
4. Input: 100000 elements. Runtime is approx. 6753 milliseconds.
5. Test Group: Sorting a vector of randomly generated integers in range [0, 10000].
6. Input: 1000 elements. Runtime is approx. 8 milliseconds.
7. Input: 10000 elements. Runtime is approx. 138 milliseconds.
8. Input: 100000 elements. Runtime is approx. 954 milliseconds.

* **Lessons Learned:**

In this project we successfully created a program that a librarian would use to manage books and students registered in a library. The hardest part of this was implementing the foundational classes such as the hash table that used chaining, and the linked lists for that chaining. Deciding on which methods to define in our linked lists and hash table to make it easier to develop our program, which in a sense emphasizes abstraction. The easier part was making the BookLibrary class since I already did the code for the hash table for it. I think the biggest takeaway was just that it was a good

List of possible improvements and additions:

1. The biggest improvements to this application would be making it so the user could edit books in a clean way.
2. A way to search for books based on the author that created it.
3. Allow the program to be able to contain duplicate titles of books.
4. Having a GUI rather than using a command line interface.
5. Incorporating an API to check if add books via ISBN, or to check whether the user was adding in real books.