Programming Quiz 2 - Algorithm Analysis

Note: This quiz has two questions.

Q1 Algorithm Design

7 Points

DungeonGator is a repository containing *n*m* books, each of which has a title length of at most *t* characters. All the books are stored in a 2-D container or collection, **A[n][m]** which has *n* rows and *m* columns. Each slot in the container contains a random book title. Thus the collection is not sorted and is purely random. For example,

A[n][m]	0	...	m-1
0	pride and prejudice	...	don quixote
.	.	.	.
.	.	.	.
n-1	the great gatsby	...	ulysses

Write a function using pseudocode or C++ code that takes in as input this repository and returns a new 1-D container with **unique** book titles. The returned repository must keep one copy of multiple repeated book titles.

Note: You can assume that the input collection or container is a 2-D array. Other containers like vectors or lists are fine as long as you state what you are using.

```
set<string> getRepo(vector<vector<string>> arr)
{
    set<string> book_repo;

    for(auto row: arr) // O(row) operations, where row is the number of rows in
the array
    {
        for(auto col: row) // O(col) operations, where col is the number of
columns in the array
        {
            book_repo.insert(col); // O(log(n)) operation, where n is the number
of elements in the set
        }
    }

    return book_repo;
}
```

Q2 Algorithm Analysis

3 Points

Describe and justify the worst-case **time** and **space** complexity of your designed algorithm (the one you wrote in Q.1) in Big O notation.

Time Complexity:

The algorithm must iterate across all of n rows and m columns, so using the nested for-loop, the nested for-loop has a time complexity of $O(n*m)$. However, each time the element from the array is inserted into the set, the insertion takes $\log(s)$ time in the worst case, where s is the size of the set, since the set is implemented as a binary search tree in C++.

For this reason, the overall time complexity of this algorithm is $O(n*m*\log(s))$, where n is the number of rows in the 2-D container, m is the number of columns in the container, and s is the number of unique elements in the 2-D container.

Space Complexity:

The algorithm creates auxiliary space for the set, taking up $O(s)$ space, where s is the number of unique elements in the 2-D container. Because the set is implemented as a binary search tree, it will only allocate space for s nodes, sorting them as they are inserted, and not inserting a node if it is already in the set. For this reason, the algorithm has an auxiliary space complexity of $O(s)$.