-Contributors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Username | Email | Role | Contributions |
| Danny A. Hasen | Danny Hasen | dahasen@cpp.edu | Programmer | * Implemented Difference Method * Tested A-B vs B-A * Added Javadoc Comments |
| Jacob Carrasco | Gary Fox | garycarrasco@cpp.edu | Project manager | * Created repository * Added BagInterface * Started Efficiency Document * Added Intersection implementation * Implemented ArrayBag * Bug Fixes * Contributed to Javadoc comments * Organized meetings * Created BagDriver |
| Joseluis Ramierz | Pizel29-wq | Joseluisr1@cpp.edu | Programmer | * Created the union method * Attempted to implement Union method |

# -Repository Link

https://github.com/Gary-Fox/BagsProject

# -Table 1. The time complexities of this assignment

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Resizable**  **ArrayBag Union** | **Resizable ArrayBag**  **Intersection** | **Resizable ArrayBag**  **Difference** | **LinkedBag**  **Union** | **LinkedBag**  **Intersection** | **LinkedBag**  **Difference** |
| Best Case | n both of the bags are empty, Union will be **O(1)** for both cases because there is nothing to unify. | One or both bags are empty, in which case an empty bag interface is returned. **O(1)** | when both of the bags are empty, the Difference method will be **O(1).** This is because there is no elements to compare or remove it to. Instead of comparing or removing elements it just returns it instantly. | n both of the bags are empty, Union will be **O(1)** for both cases because there is nothing to unify. | One or both bags are empty, in which case an empty bag interface is returned. **O(1)** | when both of the bags are empty, the Difference method will be **O(1).** This is because there is no elements to compare or remove it to. Instead of comparing or removing elements it just returns it instantly. |
| Worst case | **O(n)** for both cases because we would go through each individual node or index | We traverse only one of the bags (O(n) work), the frequency of the item is considered when updating both resultFrequency and the bags themselves (O(1\*resultFrequency))+(O1\*resultFrequency). The result frequency is always <= n, so absolute worst case we’re dealing with O(n + n + n), Therefore, our total time complexity is O(3n), or **O(n)** | **O(n × m).** This is because for each element in one bag, the algorithm might have to search through all of the element in the other bag to find and remove matches. | **O(n)** for both cases because we would go through each individual node or index | We traverse every entry in one bag for every entry in another bag, giving us **O(n2)** | **O(n × m).** This is because for each element in one bag, the algorithm might have to search through all of the element in the other bag to find and remove matches. |