|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ResizeableArrayBag | | | LinkedBag | | |
| union | intersection | difference | union | intersection | difference |
| Time Complexity in the Best Case | O(n) | O() | O() | O(n) | O() | O() |
| Time Complexity in the Worst Case | O(1) | O() | O() | O(1) | O() | O() |

*ResizeableArraybag:*

**Union:**

* Best Case – O(n): Regardless of the size of the array the function calls toArray() which iterates through an array and afterwards goes through another for loop to add the other bag’s entries.
* Worst Case – O(n): Considering the worst case, the time complexity is still only dependent on two consecutive loops, meaning the time complexity is still O(n).

**Intersection:**

* Best Case – O(): In order to get a result from this function it uses one toArray() meaning it invokes a loop and consecutively after that it utilizes a for loop with another for loop nested within it making it’s time complexity equivalent to .
* Worst Case – O(): The function utilizes a nested for loop which is the highest time complexity computation found in the function making it’s big O notation .

**Difference:**

* Best Case – O(): This function uses a nested for loop and a remove() function inside of it, however since we are looking at best case the remove function will not alter the overall time complexity of the function as it can be completed at a constant time.
* Worst Case – O(): The worst case for this function involves the remove function searching the newBag’s array without a known end point. This adds to the time complexity of the overall function as this is a feature that is used within a nested for loop.

*LinkedBag:*

**Union:**

* Best Case – O(n): toArray() is called consecutively and there is one for loop in the function giving it a time complexity of O(n).
* Worst Case – O(n): Three loops are invoked in this function without any changes to time complexity depending on size of object.

**Intersection:**

* Best Case – O(): A nested for loop is invoked in this function leading to a time complexity of O().
* Worst Case – O(): The size of the objects involved in this function does not change the nested for loops found in this function.

**Difference:**

* Best Case – O(): There is a nested for loop and a call to a loop within the function, under the best circumstances this call to remove an entry is equal to the currentNode’s data meaning it will have no effect on the overall time complexity of the function.
* Worst Case – O(): If the entry is not found at the current Node then there is an unknown amount of entries that need to be iterated over thus increasing the time complexity of this process.