Method	Array	Node	Explanation
Add	O(1)/O(n) If the array doesn't need to grow, it is a simple operation to just place a value at an index. If the array does need to grow, the array needs to be run through once along its length in order to copy the values over to the new array.	O(1) Adding to the node list is a matter of simply attaching a new node to the end of the list with just a couple operations. It performs the same no matter the length of the node list.	The node list is better here because the node list doesn't need to grow. It is always O(1) rather than most of the time being O(1).
Insert	O(n) The method runs along the length of the array until it reaches the input index. It then continues along the length of the array.	O(n) The method runs along the length of the linked list until it reaches the input index. It then does a simple operation to insert the desired input value.	Both methods are O(n). The array method runs along the length of the entire list. The node list runs along the list until it reaches the input index. The node list is slightly better here, although it depends on the value of the index.
Clear	O(1) The method just creates a new array.	O(1) The method just clears the link on the header node	Both methods are very simple, and are equal in complexity.
Contains	O(n) This method is just a call to the indexOf() method.	O(n) This method is just a call to the indexOf() method.	See indexOf()
Get	O(1) It is very simple with an array to just get the value at a desired index.	O(n) The method must run along the entire linked list until it reaches the desired index.	The array method is better here because it is a simple call to get a value at an index. The node list does not have such functionality.
indexOf	O(n) The method runs through the length of the array until it finds the an object equal to the input object	O(n) The method runs through the length of the array until it finds the an object equal to the input object	Both methods are about equal in complexity as they run along the length of the list until they reach the object.
Isempty	O(1) The method just checks the size of the array.	O(1) The method just checks whether the header	Both methods are about equal in complexity as it is a

		node is linked to anything or not.	simple check of a value.
Remove	O(n) The method sets a value to null and then runs through the entire array, copying values in to close up the gap left	O(n) The method runs through the list until it reaches the node at the index.	Both methods are O(n). The array method runs along the length of the entire list. The node list runs along the list until it reaches the input index. The node list is slightly better here, although it depends on the value of the index.
remove	O(n) The method sets a value to null and then runs through the entire array, copying values in to close up the gap left	O(n) The method runs through the list until it reaches the node containing the input value	Both methods are O(n). The array method runs along the length of the entire list. The node list runs along the list until it reaches the input value. The node list is slightly better here, although it depends on the index of the object.
Set	O(1) The method just sets a value at an index to the input value	O(n) The method must run through the list until it reaches the node that needs to be changed.	The array method is less complex because an array is easy to just change values at an index. Node lists must run through the list until it reaches the index.
Size	O(1) The method checks a premade size variable that changes dynamically.	O(1) The method checks a premade size variable contained in the header node that changes dynamically.	Both methods are about equal in complexity as it is a simple check of a value.