Yazan Bawaqna

Homework 3

1)The addBad() and addBetter() methods suffer from inefficiency because they involve frequent copying of elements when resizing the array. In addBad(), every insertion triggers a complete array copy, resulting in a significant time overhead as the list size grows. addBetter() tries to mitigate this by resizing less frequently but still faces the same fundamental issue of expensive array copying during resizing.

In contrast, the final doubling trick employed in the add() method of the MyArrayList class is highly efficient. It begins with a reasonably sized array and only doubles its size when needed. This resizing operation occurs less frequently and involves copying elements once, effectively doubling the capacity. Consequently, the add() method minimizes the number of array resizes and maintains better performance, making it the superior choice for dynamic array management in the MyArrayList class.

2) Since the data in the index is not technically removed but rather assigned the value 0, the data appears to be removed while it still occupies a memory space, hence reducing the efficiency of the program. Because it doesn't actually remove the value for the index, it can return wrong outcomes when we iterate through the data. Also, it doesn't shift the value in the indices to the left, which interrupts the efficiency and the program standard design.

3) Slower for Large Lists: As the list grows larger, the time it takes to remove an element increases linearly. For lists with a significant number of elements, the remove operation becomes slower and less efficient.