**PART 1:**

**A**

**Does the algorithm actually determine the mode? If so, explain why and describe how the algorithm would handle a tie for the mode. If not, give an example where the algorithm would return the incorrect answer.**

This algorithm does not find the mode. In any and all lists the mode is only ever set to the first item.

“update the max count to be the current count and the mode to be the first item”

In an algorithm where the mode just happens to be the first item it will be correct, but any other sequence, such as [1,0,0] it will return an incorrect answer.

**What is the worst case Big-Oh complexity for the algorithm, assuming N is the size of the list? Justify your answer.**

This sequence runs on O(N) as it will perform an operation O(1) repeated n times. f(N) = CN

**Do the best and worst case Big-Oh complexities differ for the algorithm? That is, will the algorithm require fewer steps for some lists of size N (depending on the values in the list)? Justify your answer.**

Because the algorithm must run through the entire n items of the list without regard to the value of the list, there is no difference in best or worst case.

**B**

**Does the algorithm actually determine the mode? If so, explain why and describe how the algorithm would handle a tie for the mode. If not, give an example where the algorithm would return the incorrect answer.**

It generates a correct answer. In the event of a tie mode it will only result in the mode first appearing in the list as the algorithm only updates the mode when one of greater frequency is found. Because the algorithm does the work of counting the specific frequency (in an unspecified way that I shall assume is correct) the algorithm should find one single correct mode for any list.

**What is the worst case Big-Oh complexity for the algorithm, assuming N is the size of the list? Justify your answer.**

As the algorithm does not specify checking to see if items in the list have been previously counted, this algorithm has a Big-Oh of N^2

For each item in the list of n items, the algorithms must traverse all n items to count them again and again. This creates a term of N\*N or N^2 within our algorithm.

**Do the best and worst case Big-Oh complexities differ for the algorithm? That is, will the algorithm require fewer steps for some lists of size N (depending on the values in the list)? Justify your answer.**

Were this list to change its frequency of traversal dependent upon the values found it would have separate cases, but this algorithm does not. Once again, having no difference between any list of size n.

**C**

**Does the algorithm actually determine the mode? If so, explain why and describe how the algorithm would handle a tie for the mode. If not, give an example where the algorithm would return the incorrect answer.**

This algorithm will return a single correct mode. As the items have been sorted, counting the items as the size of groups will result in finding the most frequent. It will not find any modes other than the first of the list as it only updates the mode when one of greater frequency is found.

**What is the worst case Big-Oh complexity for the algorithm, assuming N is the size of the list? Justify your answer.**

Beginning with insertion sort O(n^2) and following it with a single pass of the list (O(N)) makes this algorithm O(N^2).

**Do the best and worst case Big-Oh complexities differ for the algorithm? That is, will the algorithm require fewer steps for some lists of size N (depending on the values in the list)? Justify your answer.**

As insertion sort is highly dependent on the sortedness of the list (if the list is already completely or even nearly sorted) some lists will require significantly more steps than others.

[3,6,4,7,9,3] will take more steps than [3,3,4,6,7,9]

**D**

**Does the algorithm actually determine the mode? If so, explain why and describe how the algorithm would handle a tie for the mode. If not, give an example where the algorithm would return the incorrect answer.**

This algorithm will find a correct mode, in the event of a tie the mode with the lowest value will be returned. This algorithm does a good job tracking the frequency of all values through the use of bins and thus will result in a correct mode.

**What is the worst case Big-Oh complexity for the algorithm, assuming N is the size of the list? Justify your answer.**

The worst case Big-Oh remains O(N) as this algorithm f(N) = CN + CN + C + CN + CLthe difference between max and min + C

**Do the best and worst case Big-Oh complexities differ for the algorithm? That is, will the algorithm require fewer steps for some lists of size N (depending on the values in the list)? Justify your answer.**

As the algorithm fully traverses the list several times, there is virtually no difference between lists. The only (incredibly minor) difference would be a list with greater diversity in values will require more steps only during the part of the algorithm where we traverse the “counts of size” array. This array would be significantly larger in a random sequence than one where all the values are the same.

PART 2:

Theoretical-

Big-Oh test 1: O(1) + [O(n) \* O(1)] + O(1) = O(n)

Big-Oh test 2: O(1) + [O(n) \* O(n)] + O(1) = O(n^2)

Big-Oh test 3: O(1) + O(1) + [O(1) \* {O(n)+O(1)}] + O(1) = O(n)

Big-Oh test 4: O(1) + O(1) + [O(1) \* {O(n)+O(1)}] + O(1) = O(n)

Big-Oh test 5: O(n) \* [O(n) + O(n)] = O(n^2)

Big-Oh test 6: O(n) \* [O(1) + O(1)] = O(n)

Experimental-

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | getCounts(ArrayList) | getCounts(LinkedList) | getCountsV2(ArrayList) | getCountsV2(LinkedList) | addRemove(ArrayList) | addRemove(LinkedList) |
| 10000 | 0/1/1 | 0/0/0 | 1/1/1 | 1/1/1 | 17/15/18 | 17/17/17 |
| 20000 | 1/1/1 | 0/1/1 | 3/1/2 | 1/1/1 | 67/60/67 | 67/67/68 |
| 40000 | 3/2/2 | 1/1/2 | 2/2/2 | 2/1/2 | 265/256/263 | 264/259/259 |
| 80000 | 4/3/4 | 1/1/0 | 4/4/4 | 2/1/1 | 1151/1129/1141 | 1138/1140/1135 |
| 160000 | 5/3/4 | 2/2/2 | 5/5/3 | 2/1/2 | 4778/4741/4605 | 4755/4743/4739 |
| Big-Oh | Log n | Log n | Log n | Log n | n^2 | n^2 |