1. See Code
2. See Code
3. Below
    a.

	LinkedBag1	ArrayBag1	LinkedBag274
Size	1 mil Nodes	1 mil elements	< 1 mil nodes (statistically)
Best insert	O(1)	O(1)	O(1)
Worst insert	O(1)	O(1)	O(N)
Best getFreqOf	O(N) ← N=1 mil	O(N) ← N=1 mil	O(1)
Worst getFreqOf	O(N) ← N=1 mil	O(N) ← N=1 mil	O(N) ← N<1 mil

LinkedBag1 has 1,000,000 Nodes that each have unique values, with some duplicate Nodes. ArrayBag1 has 1,000,000 elements that statistically are not all unique. LinkedBag274 contains less than 1,000,000 Nodes (the number of unique values).

Entering the elements takes the same amount of time for both LinkedBag1 and ArrayBag1 in O(1) time (immediately placing it at the end / next position). For LinkedBag274 it must traverse the chain to see if an element exists already. If it does then it can be put there. This takes on average O(N) time.

getFrequencyOf runs in the same amount of time for both LinkedBag1 and ArrayBag1, as it must traverse the whole array / link chain to count the frequency of each object O(N), however LinkedBag274 only must traverse the chain until it finds the node, taking on average O(N) time (though this should almost always be faster than the others).

    b.

	LinkedBag1	ArrayBag1	LinkedBag274
Size	1,000,000 Nodes	1,000,000 elements	10 Nodes
Best Insert	O(1)	O(1)	O(1)
Worst Insert	O(1)	O(1)	O(N) ← N = 10
Best getFreqOf	O(N) ← N=1 mil	O(N) ← N=1 mil	O(1)
Worst getFreqOf	O(N) ← N=1 mil	O(N) ← N=1 mil	O(N) ← N = 10

LinkedBag1 has 1,000,000 Nodes that each have unique values, with many duplicate Nodes. ArrayBag1 has 1,000,000 elements with many duplicate. LinkedBag274 contains 10 Nodes (assuming all numbers came up at least once).

Entering the elements takes the same amount of time for both LinkedBag1 and ArrayBag1 in O(1) time (immediately placing it at the end / next position). For LinkedBag274 it must traverse the chain to see if an element exists already. If it does then it can be put there. This takes on average O(N) time (where N is 10 iterations).

getFrequencyOf runs in the same amount of time for both LinkedBag1 and ArrayBag1, as it must traverse the whole array / link chain to count the frequency of each object O(N), however LinkedBag274 only must traverse the chain until it finds the node, taking on average O(N) time (where N is 10 iterations).

4. Below
    a. Best Case: ary[0][0] is the element we are seeking. O(1)
    b. Worst Case: ary[N-1][N-1] is the element we are seeking. O(N^2)
        i. The element could be in a position this loop will not reach, for example, the position ary[r][c] where c is less than r will never be checked. This will account for roughly half of the elements in the array. This will result in going through the full loop O(N^2).
    c. Average Case: Assuming the key is actually located in the checked elements, the Big O is O((N^2/2)/2) which is the same as O(N^2).
5. Below
    a. O(N^2).
        i. The first loop is executed N/2 times, while the inner loop is executed N/2-r times. As N grows much larger this grows quadratically.
    b. O(N)
        i. The computation time grows simply with N
    c. O($(2N)^2$) = O(N^2)
        i. Simple double nested loop grows quadratically
    d. O(N)
        i. The functions completed inside each loop are simply O(1) and have no effect on the Big O notation
    e. O(N^2)
        i. The first and second loop happen N times, while the 3rd and 4th loop happen 3 times each, which do not affect the Big O notation.
    f. O(N^4)
        i. Simple quadruple nested loop grows quadratically.