2.
a)
f1 = n
n = O(n log n)
n = O(n log (n^2))
n = O(n^2.5)
n = O(n sqrt(n))
n = O(n^2 log(n))
f1 is less than all of these functions so they are all big O

f2 = n log n
n log n = Ω(n) because it is larger than function n
n log n = O(n log (n^2))
n log n = O(n^2.5)
n log n = O(n sqrt(n))
n log n = O(n^2 log(n))
The rest are big O because the equations are bigger than n log n

f3 = n log n^2
n log n^2 = Ω(n)
n log n^2 = Ω(n log n)
These are both Ω because f3 is bigger than both functions
n log n^2 = O(n^2.5)
n log n^2 = O(n sqrt(n))
n log n^2 = O(n^2 log(n))
The rest are big O because the equations are bigger than n log n^2

f4 = n^2.5
n^2.5 = Ω(n)
n^2.5 = Ω(n log n)
n^2.5 = Ω(n log n^2)
n^2.5 = Ω(n sqrt n)
n^2.5 = Ω(n^2 log n)
These are both Ω because f4 is a high degree than all of the following functions

f5 = n sqrt(n)
n sqrt(n) = O(n)
n sqrt(n) = O(n log n)
These are both big O because f5 is bigger than both functions
n sqrt(n) = Ω(n log n^2)
n sqrt(n) = Ω(n^2.5)
n sqrt(n) = Ω(n^2 log n)
These rest are Ω because they are all less than the following functions

f6 = n^2 log n
n^2 log n = Ω(n)
n^2 log n = Ω(n log n)
n^2 log n = Ω(n log n^2)
n^2 log n = O(n^2.5) because the function has a higher degree than f6 so it is greater than it
n^2 log n = Ω(n sqrt n)
The rest are Ω because f6 is has a higher degree than them

B) 10 is O(n log n)
10n <= n log n (c for left is 10 and c for right is 1)
10 <= log n (divide both sides by n)
10^10 <= n
So it is true for all n >= 10^10

C) The Big Oh running time would be O(n^3). I got this because our outer loop goes through n number of times when we have 2 inner loops, one that loops through n^2 times and one that loops through 4n times. So ideally it would be a runtime of n^3 + 4n^2 but because n^3 is the dominant degree we can just say that it is O(n^3)

D)
The run time of this equation would be O(n^3) because there are three nested for loops that run through each element about n times.

The error that I would fix in this equation would be the for loop in the remove method because it wouldn't make sense to loop through the array backward and by adding j+1 this could possibly cause an out of bounds error. So I fixed it by starting at i+1 and looping through until n-2.

Algorithm: removeDuplicates
    for i=1 **to** n
        if isDuplicate (i)
            remove (i)
        endif
    endfor
return n

Method: isDuplicate (i)
    for j=1 **to** n
        if A[j] = A[i]
            return true
        endif
    endfor
    return false

```
Method: remove (i)
// Removes and performs a left-shift.
   for j=i+1 to n-2
      A[j] = A[j+1]
   endfor
```