Problem Set 2

2.14 First of all, we need to merge-sort the array a1 (O(nlogn) so as to group duplicates. Then we create another array a2 to store the non-duplicates. Finally, we set up a loop:

```
for  i = 1 to a1.length( )
If a1[i] == a1[i+1]
i++

copy a1[i] to a2

return a2;
```

2.15 In order to split the set into 3 subset without allocating new memory, we need two loops to sort the array. The first loop arranges all elements smaller than v at the front of the array. The second loop arranges all elements equal to v right behind the first subset. In this case, the remaining part of the array contains elements larger than v. In each loop, the functionalities of s and i are pretty much the same. s points to the latest position of the subset while i is used for iterating the whole loop to find the corresponding value.

```
An array t is given
Integer s = 1 //used for swap elements
for  i = 1 to n
If t[i] < v
swap t[s] and t[i]
s++

for i = s to n //the second loop is used for arrange the elements equal to v
If t[i] == v
swap t[s] and t[i]
s++
```

3. Approach 1: Merge-sort the array first and the number at the n/2 position is always the majority number because the list is guaranteed to have a number which appears more than n/2 times in the list. In this case, the number indexed at n/2 in a sorted list is always the majority number. And time complexity is O(nlogn).

Approach 2: Creating two loops to count each number in the array and check if the count of each number is larger than n/2.

In this approach, the worst case is that the majority number is all positioned at the second half of the array. Therefore, the time complexity is O(n/2 * n) = O(n^2)

To recap, the first approach is better since it got a lower time complexity.