```
package com.Jaynish;

public class Main {

  public static void main(String[] args) {
  }

  public static int findMax(int[] a){ //function to determine max number in array

     int max = a[0]; // initialize max to first element in array  O(1)
     for (int i = 1; i < a.length; i++) {  O(N)
        if (a[i] > max)
           max = a[i]; // determine new max O(1) N times
     }
     return max; // return largest number
  }


  public static void Problem1Sort(int[] a){
     int place = 1; // initially in ones digit O(1)

     int[] sorted = new int[10];
     int x = findMax(a); // find the max of the array O(N)
     while(x/place >0) {
        int[] count = new int[10]; // count array
        for (int i = 0; i < a.length; i++) {  O(N)
           count[(a[i]/place)%10]++;// get count for each digit with distinct value O(1) N times
           // place count in appropriate index
        }
        for (int i = 1; i < 10; i++){   O(N)
           count[i] += count[i-1]; // modify count by adding previous counts O(1) 10 Times
        }
        for (int i = a.length-1; i>= 0; i--){ // working backwards O(N)
           sorted[--count[(a[i]/ place)%10]] = a[i]; //place integers in the correct O(1) N times
           // output positions and decrease count by one
        }
        for (int i = 0; i < a.length; i++){ O(N)
           a[i] = sorted[i]; // copy the sorted array onto the input array O(1) N times
        }
        place *=10; // move from ones spot to tens spot to hundreds to etc... O(1)
     }
     for(int i =0; i < a.length; i++){ O(N)
        System.out.print(a[i]+ " ");  O(1) N times
     }

  }
}
```

c.) The algorithm in problem 1 has a time complexity of O(N). The fastest growing functions in the algorithm are single for loops which are looped N times.

```
  for (int i = a.length-1; i>= 0; i--){ // working backwards O(N)
            sorted[--count[(a[i]/ place)%10]] = a[i]; //place integers in the correct O(1) N times
            // output positions and decrease count by one
         }
```

The highlighted snippet of code above causes bottlenecking within my algorithm. When N is small the bottlenecking in this for loop is more visible than in the other for loops. This for loops unlike the others has more than two array accessors.

However, like the rest of the operations this for loop still have an O(N) time complexity.

d)
Yes my algorithm is stable, the following code snippet, which implements a radix sort using a counting sort keeps the algorithm stable:

```
for (int i = 0; i < a.length; i++) {  O(N)
            count[(a[i]/place)%10]++;// get count for each digit with distinct value O(1) N times
            // place count in appropriate index
         }
         for (int i = 1; i < 10; i++){   O(N)
            count[i] += count[i-1]; // modify count by adding previous counts O(1) 10 Times
         }
        for (int i = a.length-1; i>= 0; i--){ // working backwards O(N)
            sorted[--count[(a[i]/ place)%10]] = a[i]; //place integers in the correct O(1) N times
            // output positions and decrease count by one
         }
         for (int i = 0; i < a.length; i++){ O(N)
            a[i] = sorted[i]; // copy the sorted array onto the input array O(1) N times
         }
         place *=10; // move from ones spot to tens spot to hundreds to etc... O(1)
      }
```