```
#include <iostream>
using namespace std;

int findMax(int a[], int arraySize){

  int max = a[arraySize-1];

  for (int i = arraySize-2; i>=0; i--){  This causes linear runtime, but is not the bottleneck.

    if(a[i] > max){
      max = a[i];
    }
  }

  return max;

}


int findCount(int a[], int arraySize){

  int count = 0;

  int max2 = findMax(a , arraySize);




  for(int i = 0; max2>= 1; i++){

    max2 /= 10;

   count = i+1;


  }


  return count;

}

void Print(int a[], int arraySize){
  for(int i = 0; i<arraySize; i++){
    cout<<a[i]<<" ";
  }
}

void digitSort(int a[], int arraySize, int x){

  int digitcopy[10] = {0};

  int sorted[arraySize];

  for (int i = 0; i<arraySize; i++){  This causes linear run time, and will be part of the bottleneck.

     digitcopy[(a[i]/x)%10]++;
  }

  for (int i = 1; i<10; i++){
    digitcopy[i] += digitcopy[i-1];
  }
  for(int i = arraySize-1; i>= 0; i--){  Linear run time once again, and causes part of the bottleneck.
```

```
    sorted[digitcopy[(a[i]/x)%10] -1] = a[i];
    digitcopy[(a[i]/x)%10]--;
  }
  for(int i =0; i<arraySize; i++){  Causes linear runtime yet again.
    a[i] = sorted[i];
  }
}

void Problem1Sort(int a[], int arraySize){

  int numReps = findCount(a, arraySize);

  int i = 0;

  for(int x =1; i<= numReps; x*=10){

    digitSort(a,arraySize,x);


    i++;


  }


}
```

The bottlenecking function in my radix sort is the digitsort function, which causes a runtime of O(3N), which is essentially linear run time, as constants/coefficients do not get factored into the simplified runtime.

D. My sort is stable as I use strictly less than, meaning that duplicates will not be sorted with each other.

```
  for (int i = 1; i<10; i++){
    digitcopy[i] += digitcopy[i-1];
  }
  for(int i = arraySize-1; i>= 0; i--){  Linear run time once again, and causes part of the bottleneck.
    sorted[digitcopy[(a[i]/x)%10] -1] = a[i];
    digitcopy[(a[i]/x)%10]--;
  }
  for(int i =0; i<arraySize; i++){  Causes linear runtime yet again.
    a[i] = sorted[i];
  }
}
```

E. I was unable to get my algorithm to work using constant space, unfortunately. However, I know that using constant space would give the massive advantage in that the size of the array I am sorting wouldn't matter to the sort itself, meaning that if I'm sorting an array that takes half my memory, it will actually be sorted, unlike in a case where O(N) space is used.