```cpp
#include <iostream>
#include <utility>
#include <Problem1.h>
using namespace::std;
 void PlacementSorting(int arr[], int mySize){
        bool swapped=true;
        //cout<<mySize<<endl;
            for(int modder=10;modder<100000000;modder*=10){
                swapped=true;
                while(swapped){
                swapped=false;
                    for (int i=0,j=1;j<mySize;i++,j++){
                    if(((arr[i]%modder)/(modder/10))>((arr[j]%modder)/(modder/10)))
                    {
                    swap(arr[i],arr[j]);
                    swapped=true;
                    }
                    //for(int k=0;k<mySize;k++){
                    //cout<<arr[k]<<" ";
                    //}
                    //cout<<modder<<endl;


}

}
}
};


int main(){
int arr[]={436,20,8795,2555,9001,16};
int mySize=sizeof(arr)/sizeof(arr[0]);
PlacementSorting(arr,mySize);

for(int k=0;k<mySize;k++){
                    cout<<arr[k]<<" ";
                    }


    return 0;
}
```

My Bottleneck
Unstable due to my contraint up to 100000000

Problem 1

So the way that this problem was approached is simply using mod and bubble sorting. I had to use mod to make sure that only the digit in the same place of each number was being compared and not the entire number itself. Then I needed a way to compare those digits as well as sort them. I picked bubble sorting due to the fact, I knew how to implement that perfectly. As for my bottleneck, that would have to be my outside for loop that is highlighted in the code. This is basically a bottle neck, because even if the array is done sorting, the for loop will continue to run and thus not letting the program run as fast and possible. I did try to fix his using a while loop with the condition that if overall swap does not occur, then exit but that was leading to errors, so I played it safe.

For the part about constant extra space. I believe that my code does an okay job with the extra space portion due to the fact it does not change the size of the array or create new arrays. This is good because it makes sure that there is not issue with the amount of memory need. It will take the array (which already has allocated memory) and just sort it without storing the data anywhere else.

The best-case scenario is that the array that come in is already sorted and the swap function does not have to run. The worst-case scenario is the array is sorted in a manner that makes it so that every time the comparison runs, the swap function will also run.