first, find the largest bit of all the integers by the algorithm below

```
for(int i = 0; i < arraySize; i++)
{
    while(a[i] > Math.pow(10, digit))
    {
        digit++;
    }
}
```

since we have got the largest bit of the array, the we got the time that we need to loop. In the loop, first we calculate the frequency of single digit in every integer. and in the next loop, we calculate tens digit…… until we go through the largest bit. as we can see in the following code.

```
int[] bucket = new int[10];
for(int i = 0; i < arraySize; i++)
{
    bucket[(a[i]/flag)%10]++;
}
```

the bucket is used to calcular the frequency of each digit..

After this, we calculate the begin index for the element in the bucket, and then we can got an array which is ordered by the single digit. with the it, we make the sort by tens digit, when it's to the end of the loop, we get a anew array which is ordered.

the following code shows how calculate the begin index for the element in the bucket and place it in a new array by the index we have just calculated just now.

```
int[] beginIndex =  new int[10];  for(int i = 1; i < 10; i++)
{
    beginIndex[i] = beginIndex[i-1] + bucket[i -1];
}

int[] tmp = new int[arraySize];
for(int i = 0; i < arraySize; i++)
{
    int index = (a[i]/flag)%10;
    tmp[beginIndex[index]++] = a[i];
}
```

the bottleneck for it is that it needs extra space.

the loop that ergodic every bit of the number taking the longest in the sort

its Big-Oh notational runtime is O(d(n+r))

It's stable, because for every loop the frequency is a specific, and the loop time is depended on the integer which has the highest bit in the array.

using O(n) extra space provide would make the space complex smaller