Problem 1

a) The implementation of the code is in Count.py, and the figure indicates the situation while sorting the given elements.

```
count of 9 is increased
count of 1 is increased
count of 6 is increased
count of 7 is increased
count of 6 is increased
count of 2 is increased
count of 1 is increased
count: [0, 2, 1, 0, 0, 0, 2, 1, 0, 1]
second stage...
count: [0, 2, 3, 3, 3, 3, 5, 6, 6, 7]
store the sorted result
put 9 at index 6
put 1 at index 1
put 6 at index 4
put 7 at index 5
put 6 at index 3
put 2 at index 2
put 1 at index 0
Sorted Result...
[1, 1, 2, 6, 6, 7, 9]
```

b) The implementation of the code is in Bucket.py, and the figure indicates the situation while sorting the given elements.

```
Set empty buckets
[[], [], [], [], [], [], [], [], [], [], [], [], [], []]
search bucket for 0.9
put in bucket 6
current elements in buckets: [[], [], [], [], [], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.1
put in bucket 0
current elements in buckets: [[0.1], [], [], [], [], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.6
put in bucket 4
current elements in buckets: [[0.1], [], [], [], [0.6], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.7
put in bucket 4
current elements in buckets: [[0.1], [], [], [], [0.6, 0.7], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.6
put in bucket 4
current elements in buckets: [[0.1], [], [], [], [0.6, 0.7, 0.6], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.3
put in bucket 2
current elements in buckets: [[0.1], [], [0.3], [], [0.6, 0.7, 0.6], [], [0.9], [], [], [], [], [], [], []]
search bucket for 0.1
put in bucket 0
current elements in buckets: [[0.1, 0.1], [], [0.3], [], [0.6, 0.7, 0.6], [], [0.9], [], [], [], [], [], [], []]
After putting values to buckets...
[[0.1, 0.1], [], [0.3], [], [0.6, 0.7, 0.6], [], [0.9], [], [], [], [], [], [], []]
call insertion sort for [0.1, 0.1]
call insertion sort for []
call insertion sort for [0.3]
call insertion sort for []
call insertion sort for [0.6, 0.7, 0.6]
call insertion sort for []
call insertion sort for [0.9]
[0.1, 0.1, 0.3, 0.6, 0.6, 0.7, 0.9]
```

c) Function F (arr,n,a,b)

B[k] ← array with all 0s

for (i= 0 to n) do

R[arr[i]] = R[arr[i]] + 1

end for

```
        for (i=0 to k) do
            R[i] = R[i] – R[i-1]
        end for
    return R[b] – R[a-1]
end function
```

d) The implementation of the code is in Word.py.

e)
If the elements given are uniformly distributed, but the range of the set is very close to zero, all elements in the set will go into one bucket. So, the elements in this bucket will be sorted by using the insertion sort, whose worst case time complexity is $O(n^2)$. Hence, the time complexity for bucket sort worst case will also be $O(n^2)$.
For example, set of elements [0.1,0.15,0.18,0.13,0.11]
And the options of buckets are 0.0-0.09, 0.1-0.19, and 0.2-0.29.
Then, in this scenario, all elements in the set will be put into the bucket of 0.1-0.19, and they will be sorted by using the insertion sort. Hence, the time complexity will be $O(n^2)$.

f)

```
Function distance (n)
        d = sqrt((n1*n1)+(n2*n2))
        return d

Function Greater(a,b)
        cmp = distance(a) > distance(b)
        return cmp

Function Sort (arr,n)
        swap = 1
        while (swap) do
                swap = 0
                for (i = 0 to n-1) do
                if (Greater (arr[i] , arr[i+1])
                        swap = 1
                        swap (arr[i] , arr[i+1])
                end if
        end while
end function
```

Problem 2

a) The implementation of the code is in Radix.py.

b) The normal radix sort starts from the least significant bit, and iterates to the most significant bit, so it uses d operations on the counting sort routine.

However, in Hollerith's Radix sort, the sorting uses the bucket sort routine and starts form the most significant bit.

In the worst case, every step will divide the input into base cases, and in every depth, we will have n elements to put in and out of buckets, with k, which is the tree depth of the number of digits, k. Hence, the time complexity will be O(k*n) = O(n) .

In the average case, half of the buckets will have size of either 1 or 0, and the rest will have more than 1 element. Hence, the total running time will be O(n+k/2*n) = O(n)

In the best case, all elements will fall into different buckets, and all elements will be sorted, so the time complexity is O(n).

In every recursion, we have to put n elements in and out of buckets, so the space complexity is O(k*n) = O(n). In the average case and worst case, n buckets will be initialized and the size of each bucket will not exceed the size they need since they are dynamic. For each recursive call, new buckets might be created for k (maximum) times, so k*n buckets can be created.

In the best case, the space complexity will be O(n).

c) The radix sort would be the best way to sort integers since the range of integers will become significantly large. We can modify the algorithm in sub-problem a) by setting base to n.

```
Function RadixSort (arr,n,e)
        int o[n], c[n]
        for (i=0 to n-1) do
                c[(arr[i]/e)%n] ++
        for(i=1 to n-1) do
                c[i] = c[i-1]
        for (i=n-1 to 0) do
                o[c[(arr[i]/e)%n]-1] = arr[i]
                c[(arr[i]/e)%n] --
        for(i=0 to n-1) do
                arr[i] = o[i]

Function Sort (arr,n)
        RadixSort(arr,n,1)
        RadixSort(arr,n,n)
        RadixSort(arr,n, n*n)
```

References

B. (2019, January 28). Sort n numbers in range from 0 to n^2 - 1 in linear time. Retrieved from https://www.geeksforgeeks.org/sort-n-numbers-range-0-n2-1-linear-time/
(I used this website to get ideas for Problem 6.2 c)

Filipe, L. (n.d.). Retrieved March 25, 2019, from https://www.quora.com/How-can-I-sort-strings-using-Radix-Sort
(I used this website to get ideas for Problem 6.1 d)

Hsiao, O. (2019, February 04). Bucket Sort. Retrieved March 25, 2019, from https://www.geeksforgeeks.org/bucket-sort-2/
((I used this website to get ideas for Problem 6.1 b)

N. (n.d.). MSD radix string sort in python. Retrieved March 25, 2019, from https://gist.github.com/nlguillemot/3093825
(I used this website to get ideas for Problem 6.2(a))