Assignment 6
Jovan Shandro

Problem 6.1

//note: in all pseudo codes below the array starts with index 0 and ends with length-1

c)The preprocessing part includes a loop with k iterations and one with n iterations so the pre-processing time is $\Theta(n+k)$ and in the end count[k] will contain how many integers are there from 0 to k, so to calculate how any are in between a and b, we calculate count[b] – count[a-1]. This takes $\Theta(1)$ time.

//the pre-processing part

```
Pre-processing (A, k)
     for i=0 to k-1 inclusive
          count[i] = o;
     end for

     for i=0 to A.length-1 inclusive
          count[A[i]]+=1
     end for

     for i=1 to k-1 inclusive
          count[i] += count[i-1]
     end for
  return count
end function

Linear_search_range(count, a, b)
     return (count[b] = count[a-1])
end function
```

e) In the worst case, all entries of the arrays will end up in the same bucket. In this case the complexity becomes equal to the complexity of the algorithm used to solve each bucket, which in my case, (the implementation in point b) , runs in O(n*lg(n)) complexity. So the implementation in point b of bucket sort has a worst case time complexity of O(n*lg(n)). An example will be the case <0.1, 0.19, 0.12, 0.11, 0.18, 0.13, 0.17, 0.15, 0.16, 0.14>. In this case all the entries will end up in the second bucket (the first one will keep track of those with 0 after decimal point and the second one the numbers with 1 as the first digit after the decimal point) as they all have 1 as their tenth digit and in this case n = 10.

f) //since there is no bound in asymptotic time complexity I will be using bubble sort
//let point be a class with 2 properties , x and y , both doubles

```
distance(point a, point b)
     return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2))
```

```
Sort(point[] A)
     n = A.length-2
     exchanged = true
     while(exchanged)
          exchanged = false
          for i=0 to n inclusive
               if(dist(A[i],{0,0}) > dist(A[i+1],{0,0}))
                    swap(A[i], A[i+1])
                    exchanged = true
               end if
          end for
          n--;
     end while
end function
```

Problem 6.2

a) Please check the implementation in java of both msd and lsd versions of the radix sort.
b) The time complexity of my implementation of the msd version of the radix sort depends on the entries that will end in each bucket. In all cases, best, worst ,and average, even if all entries are split into all the buckets the number of operations will be equal to the number of operations when all the entries will be in a single bucket as this is how the algorithm was implemented as the sum of the elements of all buckets is the always the same, so the time complexity depends on the numbers of digits of the largest number. Since the algorithm performs the same asymptotically, to make the analysis easier take the case when all entries have k digits and they differ from each other only from the least significant digit. In this case all the entries will be part of a single bucket until the last iteration, so since apart from the recursive call the other operations are carried out in $\Theta(n)$, since there will be k splits into buckets the time complexity will be $\Theta(kn)$. This mean the time complexity is still linear. Now considering the space complexity. Every time the function is called, its uses and arraylist or arraylists as auxiliary storage and since the first has size n and the other has size 10 (on every iteration all lists created are either empty or used to take the entries of the array). So the space complexity is linear $\Theta(n+k)$.
c) Please find an implementation of the algorithm the java file that contains the radix sort. The basic idea, is expressing the numbers of the array in base n, and using counting sort to sort the first, second, and third digit which allows to sort in linear time numbers up to n cubed minus 1.

References:
https://javarevisited.blogspot.com/2017/01/bucket-sort-in-java-with-example.html
https://en.wikipedia.org/wiki/Radix_sort#Most_significant_digit_radix_sorts
algorithms, 4th edition