Problem 7.1

f)

```
Dist(p1, p2){
    return SquareRoot.((p2.y-p1.y)^2+(p2.x-p1.x)^2)
}

Sort(P, n){
    Bucket[n];
    O= {0,0}
    For i=0 to n                                    //distance between point and origin can
        Insert P[i] into Bucket[(int)(Dist(P[i], O)*n)]     // only be in [0, 1], thus,
                                                            //multiply by n
    For i=0 to n-1
        Insertion_Sort(Bucket[i])
    Concatenate from Bucket[0] to Bucket[n-1] in order
}
```

Problem 7.2

c)

If the range changes as size of n changes, n^3-1 in this case. Radix Sort will perform O(n) as time complexity.

Proof:

Assume n is always the biggest integer for k digits. i.e. 9, 99, 999, 9999

Then, n^3-1 always have 3k digits. Therefore, for numbers less than n, the k will always be smaller or equal to 3k for that number^3-1.

The worst-case time complexity for Radix Sort is O(nk). In this case however, k is always way smaller than n. i.e. for n=99999, k is only 3*5=15.

Therefore, Radix Sort is O(n) in this scenario.