7 (text) Algorithm Analysis [5 points]

For each of the algorithms you wrote for problems 4-6, explain their time complexity and space complexity using Big-O notation. Explain how you arrived at your answer.

Question 4 : Time Complexity O(n); Space Complexity O(n^2)

N is the number of elements that are being hashed to the map. Even though it takes constant time to hash, with the collisions we would need to search through the list to find a spot for the element in the list. The space complexity is n^2 because in the case where the map is full, you need to double the size of the list.

Question 5: Time Complexity O(n*m); Space Complexity O(n)

N is the number of elements from the input and m is the length of the longest element from the list of strings. CountSort will sort the list of strings starting from the rightmost character.

The Space complexity is O(n) because as the list is larger we would need a larger auxiliary list for sorting which increases linearly with the input.

```
public static String[] countSort(String[] input, int length, int pos) {

   String ret[] = new String[length];
   int count[] = new int[256];


   for (int i = 0; i < length; i++) {
       int charPos;
       if (input[i].length()    > pos) {
           charPos = input[i].charAt(pos);
       } else {
           charPos = 0;
       }
       count[charPos]++;
   }

   for(int i = 1; i < 256; i++){
       count[i] += count[i-1];
   }

   for(int i = length - 1; i >= 0; i--){
       int charPos;
       if(input[i].length() > pos){
           charPos = input[i].charAt(pos);
       }else{
           charPos =  0;
       }
       ret[count[charPos] - 1 ] = input[i];
       count[charPos]--;
   }

   for(int i = 0; i < length; i++){
       input[i] = ret[i];
   }
```

```
    return input;
}
```

```
public static String[] radixSort(String[] s) {

    int longestLen = getMaxLength(s) - 1;


    for (int i = longestLen; i >= 0; i--) {
        countSort(s, s.length, i);
    }

    return s;
}
```

Question 6: Time Complexity is O(n * m) Space Complexity is O(n)

N is the size of the pattern and m is the average length of the split strings. The algorithm goes through each char in the pattern with the strings list. To get the split strings we need to go through an average of length of each string before it reaches the delimiter.

The Space complexity is O(n) where n is the size of the maps created from the pattern and split strings. In the worst case, the maps will be of same number of elements.

```
public static boolean wordPattern(String p, char delim, String s){



    String delimiter = "\\" + delim;

    String[] strings = s.split(String.valueOf(delimiter));



    System.out.println("Pattern P: " + p + " " + "Deliminter : " + delim
+
            " " + "String Input: " + s);




    if(strings.length != p.length()){

        return false;
```

```
    }

    HashMap<Character, String> charMapString = new HashMap<>();
    HashMap<String, Character> stringMapCharacter = new HashMap<>();

    for(int i = 0; i < p.length(); i++ ){

        if(charMapString.containsKey(p.charAt(i))){
            if(!charMapString.get(p.charAt(i)).equals(strings[i])){
                return false;
            }
        }
        else {
            charMapString.put(p.charAt(i), strings[i]);
        }

        if(stringMapCharacter.containsKey(strings[i])){
            if(stringMapCharacter.get(strings[i]) != p.charAt(i)){
                return false;
            }
        }else {
            stringMapCharacter.put(strings[i],p.charAt(i));
        }
    }

    return true;
}
```

Hashtable with Collisions:

Index	Element
0	25, 0
1	
2	21
3	33
4	14, 5
5	
6	107
7	9, 24
8	6
9	
10	3, 42
11	
12	7

Result of Hashing:

Items	Hash	
25	0	
14	4	
9	7	
7	12	
5	4	Collision 1
3	10	
0	0	Collision 2
21	2	
6	8	
33	3	
25	0	Already added
42	10	Collision 3
24	7	Collision 4
107	6	Collision 5

Collision 1: Probe = [4, 9]

Item: 5 Reverse(5) = 5
Home Slot: 4 Double Hash = (4 + 1·5) % 13 = 9

Collision 2: Probe = [0, 1]

Item: 0 Reverse(0) = 0 → 1
Home Slot: 0 Double Hash = (0 + 1·1) % 13 = 1

Collision 3: Probe = [10, 11]

Item: 42 Reverse(10) = 1
Home Slot: 10 Double Hash = (10 + 1·1) % 13 = 11

Collision 4: Probes: [7, 3, 33, 7, 6]

Item: 24 Reverse(24) = 42 42 % 13 = 3
Home Slot: 7 Double Hash = (7 + 1·3) % 13 = 10 taken by 3
= (10 + 2·3) % 13 = 3 taken by 33
= (3 + 3·3) % 13 = 12 taken by 7
= (7 + 4·3) % 13 = 6

Collision 5: Probe: [6, 5]

Item 107: Reverse(107): 701 701 % 13 = 12
Home slot: 6 Double Hash: (6 + 1 · (12)) % 13 = 5

Finished Hashtable:

Index	Element
0	25
1	0
2	21
3	33
4	14
5	107
6	24
7	9
8	6
9	5
10	3
11	42
12	7