Arrays & Strings

1.1 All unique chars in a string

Java Lessons :

string.toCharArray() -> As name states
string.length() -> Returns length → Method here
Array.length -> Returns length → Prop here
int[] arr1 = new int[65]; -> Array Init
for(int x:arr1) -> For each loop

Ex 1 : abc -> Yes
Ex 2 : aba -> No
Ex 3 : lxza -> Yes

Assumption : Characters ASCII : 1 Byte -> 128 Chars

Data Struct:
bool[128] hasOccured ; -> Checks occurence of each character
If any char has occurred more than onces exit the main check loop

Code:

```
boolean hasAllUniqueChars (String input){
boolean [] hasOccured = new boolean[128];

for(char c:input.toCharArray()){
  if(hasOccured[c]){
    return false;
    }
  hasOccured[c]=true ;
  }
  return true;
}
```

1.2 Reverse a null terminated string in C/CPP

Find the end using null ; use two pointer and keep swapping till they meet in middle

```
void reverseString (char* initalChar){
Int offset = 0;
char* finalChar;
while(*(intialChar+offset)!=){
offset++;
 }

finalChar= initialChar+offset-1;

while(finalChar>initialChar) {
  char temp = *  intialChar;
  * initalChar = * finalChar ;
  * finalChar = temp;
    intialChar++;
    finalChar--;
  }

}
```

1.3 Check if one String is permutation of other

// If they contain same freq of characters they are permutation of each other

```
boolean checkIfPermutation (String str1,String str2) {

if(str1.length!=str2.length){
          return false;
  }

int[] charCountForString1 = new int[128];
Int[] charCountForString2 = new int[128];

for(char c: str1.toCharArray()){
   charCountForString1[c]++;
  }

for(char c: str2.toCharArray()){
   charCountForString2[c]++;
  }


for(int i=0;i<128;i++){
   if(charCountForString1[i]!=charCountForString2[i]){
          return false;
    }

  }


  return true;
}
```

1.4 Replace all spaces in a string with %20

Assumption : Do it in place assuming has adequate space at the end to accommodate the expansion.

Input : String,Actual Length Without Padding

```
char[] stringReplaceWithHexForSpaceChar(String input,Integer length){

int lastIndex = input.length()-1;
char[] inputAsCharArray = input.toCharArray();

for(int i = length-1; i>0;i--){
    if(inputAsCharArray[i]!= ‘ ‘1){
       inputAsCharArray[lastIndex] =  inputAsCharArray[i];
       lastIndex--;
    }else{
     inputAsCharArray[lastIndex] = ‘0’;
     inputAsCharArray[lastIndex-1] = ‘2’;
     inputAsCharArray[lastIndex-2] = ‘%’;
     lastIndex=lastIndex-3;
   }


  }

  return inputAsCharArray;
}
```

1.5 Compress the repeated chars of a String

Ex: aabcccccaaa -> a2b1c5a3 ; If compressed string not smaller return the original string
Assumption : a-z are the only chars

```
public static char[]  naiveCompression (String inputString) {
   if(inputString.length()<=1){
      return inputString.toCharArray();
   }
   char[] inputStringAsCharArr = inputString.toCharArray();
   char[] compressedArray = new char[inputString.length()*2];

   int writingIndex = 0;
   char lastScanned  = inputStringAsCharArr[0];
   Integer lastScannedCharCount = 1;
   for(int i=1;i<inputStringAsCharArr.length;i++){
      if(inputStringAsCharArr[i]!=lastScanned ){
         compressedArray[writingIndex++] = lastScanned;
         compressedArray =
naiveCompressionHelper(lastScannedCharCount.toString().toCharArray(),compressedArray,writingIn
dex);
         writingIndex = writingIndex + lastScannedCharCount.toString().toCharArray().length;
         lastScanned = inputStringAsCharArr[i];
         lastScannedCharCount = 1;
      } else if(inputStringAsCharArr[i]==lastScanned && i!=inputStringAsCharArr.length-1){
         lastScannedCharCount ++;
      } else if(inputStringAsCharArr[i]==lastScanned && i==inputStringAsCharArr.length-1){
         lastScannedCharCount++;
         compressedArray[writingIndex++] = lastScanned;
         compressedArray =
naiveCompressionHelper(lastScannedCharCount.toString().toCharArray(),compressedArray,writingIn
dex);
         writingIndex = writingIndex + lastScannedCharCount.toString().toCharArray().length;
         compressedArray[writingIndex] = '\0';
      }

   }
   if(writingIndex<inputStringAsCharArr.length)
      return compressedArray;
   else
      return inputStringAsCharArr;
}

private static char[]  naiveCompressionHelper (char[] numberAsCharArray,char[] toCopyTo,int
startingIndex){
   for(char c:numberAsCharArray){
      toCopyTo[startingIndex] = c;
      startingIndex++;
   }
   return toCopyTo;
}
```

1.6 Rotate a matrix of N*N representing a pixel (RGB,Alpha)

```
public class Pixel {
   private short red;
   private short blue;
   private short green;
   private short alpha;

   public Pixel(short red,short blue,short green,short alpha){
       this.red = red;
       this.blue= blue;
       this.green = green;
       this.alpha = alpha;
   }

   public short getAlpha(){
       return alpha;
   }

   public short getRed(){
       return red;
   }

   public short getBlue(){
       return blue;
   }

   public short getGreen(){
       return green;
   }

}


/*

Sample  Input

 00 01 02
 10 11 12
 20 21 22

 Sample Output
 20 10 00   ; R1 -> C3
 21 11 01   ; R2 -> C2
 22 12 02   ; R3 -> C1

 --> R(x) -- C(n-x)
 */
```

```
public static Pixel[][] rotateImageBy90Clockwise (Pixel[][] inputImage){

   Pixel[][] outputImage = new Pixel[inputImage.length][inputImage[0].length];


   print2DArray(inputImage);

   //Rotation Logic as per Comment above
   for(int i=0;i<inputImage.length;i++){
       for(int j=0;j<inputImage[0].length;j++){
           outputImage[j][inputImage.length-i-1] = inputImage[i][j];
       }
   }

   print2DArray(outputImage);


   return outputImage;
}


private static void print2DArray(Pixel[][] array){

   for(int i =0;i<array.length;i++){
       for(int j=0;j<array[0].length;j++){
           System.out.print(array[i][j].getAlpha()+" ");
       }
       System.out.println("\n");
   }
}
```

1.7 Mark all rows and columns zero if 0 element is encountered.

1.8 Chek if s1 is a sub-string of s2 using only isSubstring method.

Linked List

2.1 Remove duplicates from a unsorted linked list ? Level up if temp buffer is not allowed ?

Input : 1->2->2->3->4 Output: 1->2->3->4
Idea: Use a hashmap

```
Class Node {
Int data;
Node next;}
```

```
public static Node removeDuplicate(Node root){
//Node copy
Node current = root;
Node prev = root;
//Early exit on 0,1 element
   if(root==null || root.next==null){
       return root;
   }
   HashSet<Integer> duplicateTracker = new HashSet<>();

   //1-2-2-3-3-4
  while(current!=null){
     if(!duplicateTracker.contains(current.data)){
         duplicateTracker.add(current.data);
         prev = current;
         current = current.next;
     }else{
         prev.next=current.next;
         current = current.next;
     }
  }
   return root;
}
```

Lookahead runner for without extra space. O(n2)

2.2 Kth last Element of a Linked List

Sol 1 : If length n is know , k=1 , Move ptr ahead by n-1 times.
Generalise and slove : k=t ; Move ptr ahead by n-k times.

Sol 2 : If any other DS is allowed.
Populate the stack and pop k-1 times to get to last kth element

Sol 3: Recursive call returning +1 from the end of recursion tree;

Sol 4 : Keep two ptrs and 1 k steps ahead

```
int findKthLastElement(Node head,int k){

Node runner1=head;
Node runner2=head;

for(int i =0;i<k;i++){
   runner1=runner1.next;
  }
}

while(runner1!=null){
 runner1=runner1.next;
 runner2=runner2.next;
}

return runner2.data;
}
```

2.3 Delete the middle element of a linked list, given access to that itself.

Thought process: Start overwriting ,as good as deletion.

2.4 Partition around a given int value

Tought process: Too trivial