Q2. String Reconstruction

Original Sequence - GAGGAGGA

Code:-

```python
# Amruth Ashok
# BL.EN.U4AIE20002

import random
def kmers(read, k):
    KList=[]
    num_kmers = len(read) - k + 1
    for i in range(num_kmers):
        kmer = read[i:i+k]
        if i==0:
            KList.append(kmer)
        else:
            KList.append(kmer)
    return KList

a=kmers("GAGGAGGA",3)
print(f"    Kmers      = {a}")

def sortkmer(kmerlist):
    for i in range(len(kmerlist) - 1):
        for j in range(i + 1, len(kmerlist)):
            if kmerlist[i] > kmerlist[j]:
                temp = kmerlist[i]
                kmerlist[i] = kmerlist[j]
                kmerlist[j] = temp
    return kmerlist

sortkmer(a)
print(f"Sorted Kmers   = {a}")

def MyFind(pat,txtList):
    for i in txtList:
        if pat[len(pat)-2:len(pat)]==i[0:2]:
            ordered.append(i)
            txtList.remove(i)
            break
ordered=[]

def arrange(MyList):
    num=random.randint(0, len(MyList)-1)
    firstLink = MyList[num]
#     firstLink = 'GAG'
    ordered.append(firstLink)
    MyList.remove(firstLink)
    for i in range(len(MyList)):
        MyFind(ordered[len(ordered)-1],MyList)

def reconstruct(List,newString):
    j=0
    for i in range(len(ordered)):
        if len(ordered)-1==i:
            new=ordered[j]
            newString=newString+new
            j+=1
        else:
            new=ordered[j]
            newString=newString+new[0:1]
            j+=1
    return newString

arrange(a)

print(f"String matching= {ordered}")
word=reconstruct(ordered,"")
print(" ")
print(f"Reconstructed  = {word}")
```

OUTPUT:-

The first kmer is selected in random(According to the actual algorithm)

Case1:

```
    Kmers       = ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']
 Sorted Kmers   = ['AGG', 'AGG', 'GAG', 'GAG', 'GGA', 'GGA']
 String matching= ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']

 Reconstructed  = GAGGAGGA
```

Case2:

```
    Kmers       = ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']
 Sorted Kmers   = ['AGG', 'AGG', 'GAG', 'GAG', 'GGA', 'GGA']
 String matching= ['AGG', 'GGA', 'GAG', 'AGG', 'GGA', 'GAG']

 Reconstructed  = AGGAGGAG
```

Case3:

```
    Kmers       = ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']
 Sorted Kmers   = ['AGG', 'AGG', 'GAG', 'GAG', 'GGA', 'GGA']
 String matching= ['GGA', 'GAG', 'AGG', 'GGA', 'GAG', 'AGG']

 Reconstructed  = GGAGGAGG
```

In **Case2** The given Original sequence matches the output

TIME Complexity:

n = len(read) - k + 1

len(read) = length of the given sequence

k = length of kmer

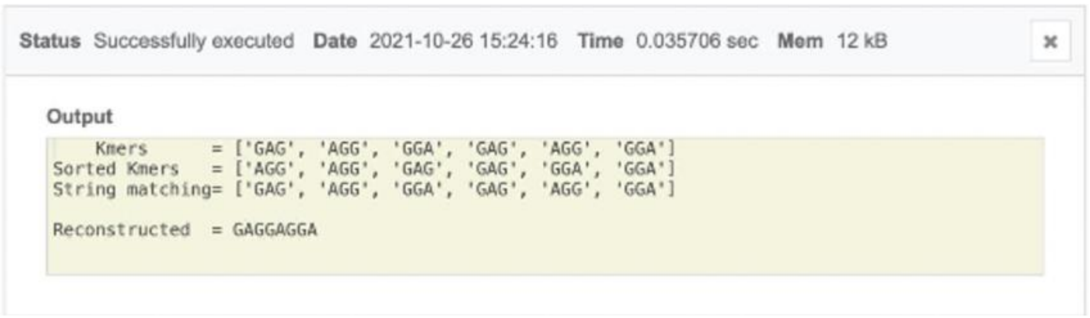
n = number of kmers

Main steps involved are:

1. sorting the kmers – **O(n*log n)**
2. rearranging kmers – **O(n)**
3. constructing String – **O(n)**

Therefore the total TimeComplexity of My Algorithm is **O(n*log n + 2n)**

Total Runtime:



Status Successfully executed Date 2021-10-26 15:24:16 Time 0.035706 sec Mem 12 kB

Output

```
    Kmers       = ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']
 Sorted Kmers   = ['AGG', 'AGG', 'GAG', 'GAG', 'GGA', 'GGA']
 String matching= ['GAG', 'AGG', 'GGA', 'GAG', 'AGG', 'GGA']

 Reconstructed  = GAGGAGGA
```