CS238 – Assignment 1

Name – Lohith Kumar Bhambore
Student ID – 862393065
Net ID – Lbham003

Question 1
Input:

- t DNA sequences each of length n, giving a matrix of dimension txn
- Let each row be referred to as a strings – S_1, S_2, S_3…….S_t, each of size n
- l, which is the length of the motif to be found
- An example is given below, (the underlined and bold sequence is the motif)

N = 69; t = 5; l = 8

S1 = aa**<u>aAgtCcgt</u>**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
S2 = agtactggtgtacatttgat**<u>acgtacgt</u>**acaccggcaacctgaaacaaacgctcagaaccagaagtgc
S3 = cctgatagacgctatctggctatcc**<u>acgtacAt</u>**aggtcctctgtgcgaatctatgcgtttccaaccat
S4 = agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatctt**<u>acgtacgt</u>**ataca
S5 = ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgtta**<u>acgtaGgt</u>**c

Output:
A vector pos with each element being the first position of the motif in the corresponding DNA sequence - <pos_1, pos_2, ….. pos_t>

Using BestScore as an evaluation metric over all strings of that length such that the starting index of the motif in S_i is always greater than the one in the S_{i-1}, where 2 <= i <= t

For the above example, the underlined and bold sequences are the motifs. And the output vector of positions is as follows –

Positions in each sequence is – {3, 21, 26, 56, 60}
Each corresponding to pos1= 3, pos2 = 21, pos3 = 26, pos4 = 56, pos5 = 60

The complexity
Let

- t DNA sequences each of length n, giving a matrix of dimension txn
- Let each row be referred to as a strings – S_1, S_2, S_3…….S_t, each of size n
- l, which is the length of the motif to be found

For the brute force method, as discussed in class, the time complexity is $O(tln^t)$

In this situation, where the start position of the motif in the i^{th} sequence must be greater than the one in $i\text{-}1^{th}$ sequence.

- We have to loop through each of the (n – l + 1) start positions in of the first DNA sequence, and in the next t – 1 DNA sequences, we're looking at most (n – l + 1 – (1)), (n – l + 1 – (2)), ….. (n – l + 1 – (t – 1)) sets of starting positions in each sequence, due to the above mentioned condition
- Given t DNA sequences, to consider all possible combinations we have the product → (n – l + 1)* (n – l + 1 – (1))* (n – l + 1 – (2))* ….. *(n – l + 1 – (t – 1))
- As n is much larger than l and t, we can consider the product to be $O(n^t)$
- For all the sets of possible values (which is t), the scoring function requires l operations, so we need to multiply the above with t*l
- Thus, giving the final time complexity as – $O(tln^t)$, the same as in the brute force method, of course it will be faster than the problem in class, as it has less number of combinations, but the order of growth doesn't change as such.

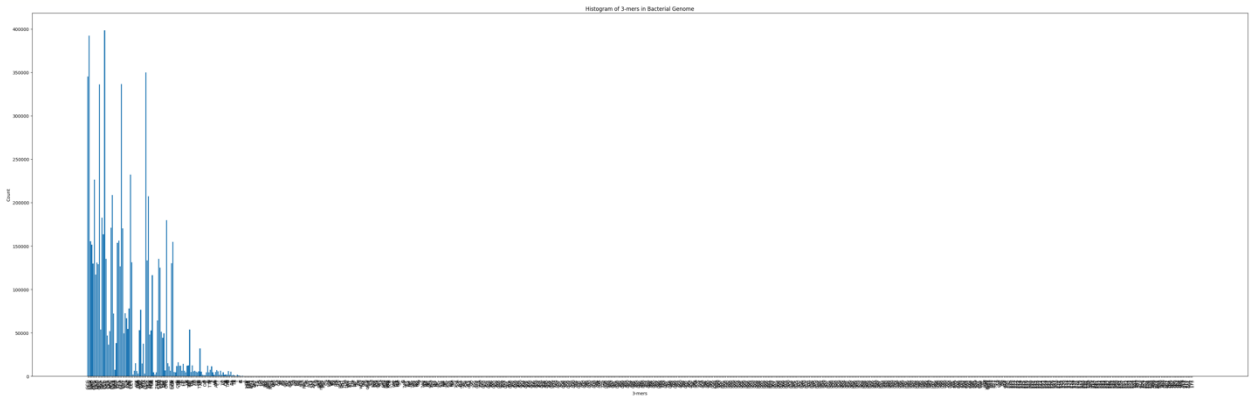
Question 2

For the genome of the bacteria: I used went through the file, and removed all the lines which started with ">" and all the 'N's in the file.
And then found the counts of all the 3-mers using a hashtable. As shown below –

```
# Import the necessary libraries
import matplotlib.pyplot as plt
from collections import Counter

# Read the bacterial genome sequence from a file
with open('genomicDataset.fna', 'r') as f:
  f.readline() # Skip the first line
  for line in f:
    if not line.startswith(">"):
      genome = f.read().replace('N', '') # Remove the N's

# Generate all 3-mers and count their occurrences
kmers = [genome[i:i+3] for i in range(len(genome)-2)]
counts = Counter(kmers)
#top_10_counts = counts[:10]

# Generate a histogram of the counts
plt.figure(figsize=(50, 15))
plt.bar(counts.keys(), counts.values())
plt.xlabel('3-mers')
plt.xticks(rotation=90)
plt.ylabel('Count')
```

```
plt.title('Histogram of 3-mers in Bacterial Genome')
plt.show()
```

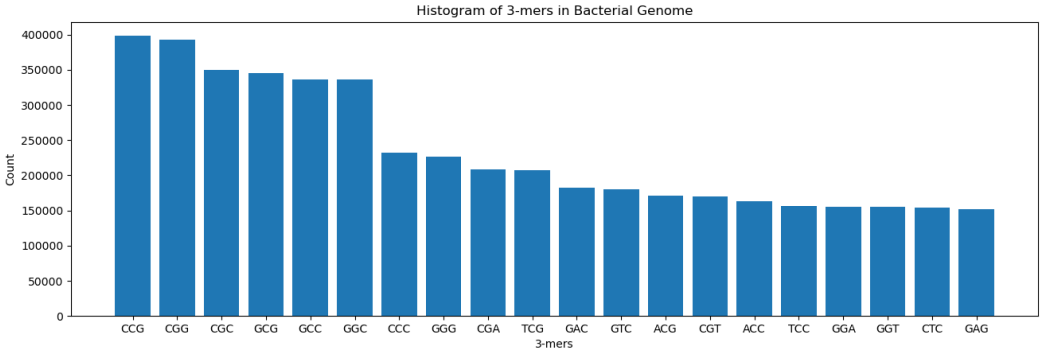
This produces a plot as below (Which is not readable, due to the enormous num of 3 mers)



Hence, I sorted all the frequencies and selected the top 20 most frequent 3-mers to plot a new histogram. The code and the new plot are as shown below –

```
# Import the necessary libraries
import matplotlib.pyplot as plt
from collections import Counter

# Read the bacterial genome sequence from a file
with open('genomicDataset.fna', 'r') as f:
    f.readline() # Skip the first line
    for line in f:
      if not line.startswith(">"):
        genome = f.read().replace('N', '') # Remove the N's

# Generate all 3-mers and count their occurrences
kmers = [genome[i:i+3] for i in range(len(genome)-2)]
counts = Counter(kmers)
sorted_dict = dict(sorted(counts.items(), key=lambda x: x[1], reverse=True))
top_20 = dict(list(sorted_dict.items())[:20])

# Generate a histogram of the counts
plt.bar(top_20.keys(), top_20.values())
plt.xlabel('3-mers')
plt.ylabel('Count')
plt.title('Histogram of 3-mers in Bacterial Genome')
plt.show()
```

This gives the following plot –



Giving the following most frequent or common 3-mers in decreasing order –
'CCG': 398306,
'CGG': 392494,
'CGC': 350059,
'GCG': 345317,
'GCC': 336420,
'GGC': 336150,
'CCC': 232197,
'GGG': 226489,
'CGA': 208807,
'TCG': 207340,
'GAC': 182622,
'GTC': 179614,
'ACG': 170938,
'CGT': 170494,
'ACC': 163403,
'TCC': 156277,
'GGA': 155629,
'GGT': 154784,
'CTC': 153827,
'GAG': 151449

Biological significance–
The most common 3-mers could indicate regions of the genome that are important for different cellular processes like gene regulation, DNA replication, and protein synthesis etc.

(Referenced chatGPT for the following part)
For example, the 3-mer CCG is a rare codon that codes for the amino acid proline. Its frequent occurrence in the genome of Streptomyces_californicus could indicate a high usage of proline in the proteins of this organism. Similarly, CGG and CGC are codons that code for arginine, and their high frequency could indicate a high usage of this amino acid. The frequent occurrence of GCG and GCC may also suggest that the genome of this organism has a high content of alanine and glycine.

It is important to note that the biological meaning of these 3-mers cannot be determined solely based on their frequency and further analysis is required to understand their significance.
For a random genome of same length, I use a random generator to choose from a character set of {A, G, T, C} for the same length as the gnome of the bacteria and then plot the histogram which as expected is a histogram with a uniform distribution –
The code and the graph as shown below –

```
# Import the necessary libraries
import matplotlib.pyplot as plt
from collections import Counter
import random

# set of characters to choose from
char_set = ['a', 'g', 't', 'c']

# generate a new string by choosing characters at random
random = ''.join(random.choice(char_set) for i in range(len(genome)))

kmers = [random[i:i+3] for i in range(len(random)-2)]
counts = Counter(kmers)

# Generate a histogram of the counts
plt.figure(figsize=(50, 15))
plt.bar(counts.keys(), counts.values())
plt.xlabel('3-mers')
plt.xticks(rotation=90)
plt.ylabel('Count')
plt.title('Histogram of 3-mers at Random')
plt.show()
```