HARINANDAN H

AM.EN.U4AIE22120

Problem 1

Generate a Random DNA Sequence Description: Create a random DNA string with letters from the whole alphabet A, C, G, and T. First make a list of random letters and then join all those letters to a string. Also write another function to count the number of bases in the random sequence and measure the CPU time for large such DNA strings. (Hint : use import random, import time)

```
import random
import time

bases =['A','T','G','C']

dna_seq_len=10000
dna_seq=''.join(random.choice(bases)for i in range(dna_seq_len))
print(f'DNA_sequence:\n{dna_seq}')
print(f"\nA: {dna_seq.count('A')}")
print(f"T: {dna_seq.count('T')}")
print(f"G: {dna_seq.count('G')}")
print(f"C: {dna_seq.count('C')}")

start_time=time.process_time()
end_time=time.process_time()
cpu_time=end_time-start_time

print(f'\nCPU time: {cpu_time:.6f}seconds')
```

```
DNA_sequence:
TCCTCCCGCTTTTGCGAATGGATTTAGGAGGGGATGCGGCCAATAAGCGATCTTTGACCATCTCCTAGCACACACGTGAACGTACGCGGCATACAATACACCGCGACGGGCGCGCGGGCGGTCGCTCGCAGGTATA

A: 2442
T: 2492
G: 2507
C: 2559

CPU time: 0.000030seconds
```

Problem 2:

Compute the Hamming Distance Between Two Strings We say that position i in k-mers p1 … pk and q1 … qk is a mismatch if pi ≠ qi . For example, CGAAT and CGGAC have two mismatches. The number of mismatches between strings p and q is called the Hamming distance between these strings and is denoted HammingDistance(p, q).

```
def Hamming_Distance(string_1, string_2):
  d=0
  if len(string_1)==len(string_2):
    for i in range(len(string_1)):
      if string_1[i]!=string_2[i]:
        d+=1
  return d

string_1="ACGTACGTA"
string_2="ACGTACGTT"

hamming_distance=Hamming_Distance(string_1, string_2)
print(hamming_distance)
```

```
1
```

Problem 3: Find Patterns Forming Clumps in a String Given integers L and t, a string Pattern forms an (L, t)-clump inside a (larger) string Genome if there is an interval of Genome of length L in which Pattern appears at least t times. For example, TGCA forms a (25,3)-clump in the following Genome: gatcagcataagggtcccTGCAATGCATGACAAGCCTGCAgttgttttac Clump Finding Problem Find patterns forming clumps in a string. Given: A string Genome, and integers k, L, and t. Return: All distinct k-mers forming (L, t)-clumps

```
def Clump_Finding(L,k, genome, t):
  result_clumps= set()
  for i in range (len(genome)-L+1):
    window=genome[i:i+L]
    kmer_count={}

  for j in range (len(window)-k+1):
    kmer=window[j:j+k]
    kmer_count[kmer]=kmer_count.get(kmer, 0)+1

  for kmer, count in kmer_count.items():
    if count>=t:
      result_clumps.add(kmer)
  return result_clumps

genome="ATCGTACGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCG"
k=5
L=75
t=4

clumps=Clump_Finding(L, k, genome, t)

print("Clumps:",  clumps)
```

```
Clumps: {'TCGAT', 'CGATC', 'ATCGA', 'GATCG'}
```

Problem 4: Find a Position in a Genome Minimizing the Skew Define the skew of a DNA string Genome, denoted Skew(Genome), as the difference between the total number of occurrences of 'G' and 'C' in Genome. Let Prefixi (Genome) denote the prefix (i.e., initial substring) of Genome of length i. For example, the values of Skew(Prefixi ("CATGGGCATCGGCCATACGCCCATGGGCATCGGCCATACGCC ")) are: 0 -1 -1 -1 0 1 2 1 1 1 0 1 2 1 0 0 0 0 -1 0 -1 -2 Minimum Skew Problem Find a position in a genome minimizing the skew. Given: A DNA string Genome. Return: All integer(s) i minimizing Skew(Prefixi (Text)) over all values of i (from 0 to |Genome|).

```
def min_skew_positions(genome):
  skew_values=[0]
  min_skew=float('inf')

  for i in range (1, len(genome)+1):
    if genome[i-1]=='G':
      skew_values.append(skew_values[i-1]+1)
    elif genome[i-1]=='C':
        skew_values.append(skew_values[i-1]-1)
    else:
      skew_values.append(skew_values[i-1])

    min_skew= min(min_skew, skew_values[i])

  min_skew_positions=[i for i, skew in enumerate(skew_values) if skew == min_skew]
  return min_skew_positions

genome="GCTAATACGTGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAG"

result_positions = min_skew_positions(genome)

print("".join(map(str, result_positions)))
```

```
8
```