T.C.

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4065 – Introduction to Computational Genomics

Project 1

GROUP MEMBERS

150117068 - Murathan Akman

150120048 – Yasin Tuğra Taş

1. Input File

First, 10 DNA sequences were created, each 500 characters long. Then, a newly created motif (10-mer) of length 10 was added to each DNA sequence, along with its 4 mutated versions. This motif was inserted into each DNA sequence at a random position.

Original Motif: CGTCAGAGAA

DNA string with mutated motif added:

CGAAGAAGGTTATACCTGATCCGCGATGCTACCTGCGCCGAACTGAAAAGTCTGTTGAGACTCCGACACTTCACTACACTCTATTCTGTAATGATGCCCCGCACGTCTTAGTGTGATGTGTTCCTCTAGGAATTTGAGGCCGTTAGCTGGTGTGGCAGCGAGC
AGGGAACAACAGTGCAAGGGGACACTTCTGCGGAATATTCCTTCGATGATCGGACCCCTCGTCGATGAAGTTGGGCATAGCCATTGTTGCATCTCTCCCAAGCCCGGAGATATATGTACATGAGTGTTGACCGATAGGCCGGGCTCGAGGATCCTTGCAGTG
TTCATCAAGTGATATCTCACCGCTAATGCCACAAAGCATAAGGTGCCACTGCAAATAAGCCTAGTCGGCGACATGTTTCCTCTACCCCGAAGATACCAGGAATAACAGGTCAGTATCTGGTAACCTGATAGCTGACACATCGACCGTTCGTATACACCGTTACG
ACAAACATGCA

AGCCGTCGCGCAGTGGAGACGCAGCCATATTTGCAGTCCTGTCCCACCCGGATTGTAGGTCTATGGTCCCCTAATTTTTTATAACTAACTCCGAATCCATTAATCTGCACCTGAGCTGCCTCTTATAGAACTTACTACATCATGAGAATTAATGATACTCTCCCC
TACCCTCCATATCCGCAGAAAATAAGAAGCCCCGATCCGCGTTGCCACCGCGTTGAAATTTCCGGTGCAGATCTAGGCACGCATGAATGACATCATCGCAACACCAGTCACAGGTCAGGCGATGATGAGTTTAACATTCTTCAGAGCCATTCCATTCCCTGTA
GCTTAACGATAGCTCTTAGGGTAGCGAACGGTAGTGCTAACCCCTCACTGAGGAATAGATCCGGGGCGTGTTGTGAAGACTCCTTAGGTACACCGCCCCACAAAGCAGAGATATCACTGTTAGGTAGCAACAGCCTGGGGGAGCTACACATCTTTCTTAGTG
CGGAGATTCC

AACGTATTTTCTGAAGAACAGCCATCCTCCCGTTACTCGAAAACTCTCCCGGCCAACAAATATAATGTAACTGGCTTGAGAATCCCTCGTCCGGCCTCTGTCGAACACAAGTATGCTAACTACGCGAACAGCCAGCCGCTTGCTAGACCGGTAGCACTGGTGA
CTTATCCCTATTGCTGCGGTTTCCGATCGCATCGCGCGATTAGTTGAAGTTCCAGACTCAATTTCTGTCAGGGATTGCCCTCTCGGGCGAGAAAAACGTCGCTCCGAGGCTTGGAGAGAAGGGCACCATACTAACCTAACGCTTATATAGGCTAACCACACGA
CTGAATAACCCCGCTGGCTTATCTGTTCCCGGCTCGGCGTATGCGAATGCATGGTGTGGCTTCGCTTATCGTTGAACCTCGAATGCGCGAAGATATCTCAGTGGCGTCAATGCGAGCCTGGAGGGCACCAACCTAAGCTGCCCCCCCACAAAGCAGATTATG
CCTGCCTTAAGC

AGAAGTCGCTTCTGGAATACCTAGGTCGAGAGCTCGATAGGCAGAACCCTACAGTAGCAGCCGCGAAAATTTTTACCTCCGGCCCATGCCACACGCTCTATCGAGTATGAATATCGCGGCCACGTTGCCCCCCTGGTCATGGACTAGAAAGCCTGGCGGTTC
ATCCCACATGATAGCCTGACAGGAAATACTGTACCCGGTTACGACAGAGTATAGCGACAACAGGCGATGGTCTAAACCACAGGTCCGAGATCTAAAGTCCACAAAGCAGGATGACCGAACTGCAAGATAAGAAACGCTAATTTATAGATTTTTTGTGGTGCGG
GTAGCATGTGCGGACACAGGGATTCATCTTACCCTCATAACCGCACCCGGTGGCTAAGCATCCTCGGCAGGATAAACTAATACGGGACACGTCCCGTGTCTGGGATAACTTAGACGAGACCATGCAATAAACGCCATCGACTCTCTGTAGCCGCACGCCATC
CAAAGGCTTATGC

TATCCTAGGCGCTGCTTGGATCAAGTAGATCCACAAAGCACGAGATCAACACACTGCAACCTTTGCGAAGTGACAGAGCAGATGTGCATGGCAGCGGCAATGGAATACCGCGTCGCTTGAACCCCAGTTTCACTGGTACACCGGCTTGGGAACTCCAATTGT
ATGTATCAATGATCAGCTCTCGGTACGACTGAACAAGCGACTGCAATTCGTAACAATGCATTGGTTCAACATTTTTACTAGGAACTTTCGTACACATTGCGTGAGACGTAATATACATATGGCCACCACTCGTTTGCTCACACGAAACTCGAAATACCGTCACGC
TCCCTTTGTGCCTGCTCATCATTAGGATGAATGTATGATTGCCGTATTACTATCGACAGGCATTAAGGTAAACCTAGAACCCCGTTGACTGAGCGCAGGCTCCCAAAGGGTGATTGTAGATGGTAGCTGCGTCCTTAGGCAACGACCACGGGCTATCCTCCGC
TTTTATTGGT

GCTTGTCGATGGGTGGGCTTGTACTGGTTCAATATAGAATTCGGCTGCCGGAAAGCCGGAATCGCGATTCGACCACCTTTGGTGACCCAGGAAGACTTTAGACGAACGAACTGAAATTGTAGCAAATCAACAGCCGGCCGTGATCCTAGCTGGTTTCGCTTC
TTCATTTCAATCTAAGGAGCGTGTTCGATGATGCAATAGAACCACAAAGCATTGTTAAGGCATTGACGATTATTAATGAGTCTGGCATCCCACAATAGCTGTCATCAGGCTCAGGAAGTGCACTTGGTAACCGGAGCGCCGAGGCGGCCTTCAATCTTAAGATT
GACGGATCCTTCTTGCAAATGCAGAGATGATAATGCTAGCACCATACCGGATGTATTTTAGGACCGGGATTCCACTCCGGTCAACGTGCCGTGCCTCCAACATAAATCTGACAAGAGAGGCAAGGCCGACAGCATTAGCCATTTGTCGTTTATAGCTTAAGTTT
TACGATAACG

GGAGGTTTACCTCATTCTAACGCCTGTTACGTAGCCCAACGGCCGGTTCGGCCCACAAAGCAAGAGCTTGTGGATCTTTGGACTTTAATTCATTGATACACGCCATACAAGCTGCCGAAATTTGCCAATCGACTCCGCCTGCTGTCGTCACCGAACACAACAA
CGGTACTTCCTTAGCGAGGCAAGTGGGCGCTTCGGAGGGGTGGTCTGTGGCTCATAGGACGTAAATGAGATCCTTCTCCAGGGCCCATTACTACAGGCATCGTCTCATCAGAGAGGTAAGACGCACACCAAAGTACAGTAGAGGAGCGTCACGTCGTTGTG
TAGCGCCCGATTATGTAAGCACGGAGACCACCCAGCCCAAGTTCTTGCCATAGAACAGAACGTTGAATATGTTCTGTCACTCATTCTTGCGATGACTAGGCAGTGCTATAATTGACCACCAGGCAGTGGAGTGGCAGCACATTGCACGCTCAGCATAGAACTC
CGAGTGCGGAAGT

TCTCCGGGTAAGGTTGCGCCTATAGAGGGATTTGCTTCCAATCCGATCAAGATCGATCACTTGTACGGTTGACGTTCTCTGTGGGCAGTCACCTTGGCCCGATACCAATTGCAGGTTATTGTTGAGCAACTCTCCTGTCAACGGAGTGCAGTTTCTCTCGTGA
CCCCACAAAGCAGTCACTGCAGTGTTTTCAGGCAGGGCGACATCTAATCATAGAAATCTCTACTCGTCGACCCTTATTTCATGGCCTGTGGGTGGTAAGCTAGCGATGGACGGCGCCTTCCATAGTCTTAGAGTAATGTATTCGGAATCCTCCATCCTTGAAGT
GGATAGACCTACGTGCCAAGGCCACGTCGAGCAACGGTTATCTAGTCCTAAGTAAATCACTCCAATATGCCTTTCTCACCTAGGTCCGATTTATTAGTGGGAGGTATCTATTGCTAGTGAACGTCAAGGAGCAACAGCACCCTTCAAGTGCCGTCCATGTAGA
CACAAGGAAA

CCTCGGAAAATTGGGTCTGACTATCTCACAAACCGATCCTTCTCTGGGTCAGGACCTGAATCGTCCTCTGGCTTTACCATACTTAAAGCTAGTTTAGCGTTCTGTGACATTATTATCCTTGGTGCGGAGACGTATGATGTACGAGCCCTGATGCGCGTTAACGC
CTGTGACTATGTACCGCGAACGAACATTCGCCTAGGCATGGGGGTATTAGGAGTCGCTATATCAGCGTACCGCAATGGTAGCCTTGTTCTTTCCACAAAGCATGGTGCGCTACTCTCCACATGTGGTCATAAACTACCTCGGGTCGATCATTAGTTAATCGAA
GATTGTGGATAAGGGCCACGTGCGGCGTGCATGGATACGTAACCTTACTATAGAAGCCCGTAAACTTGGCAGGTACACCATCTGAGCGCGTGGCTGCCACACGGTAGCATCGAGACGTCCGCTGGGAAAACGTCCCTCAACTGCATCGGCATTACAACTGT
TCGACAAACACA

ATGTGTCCCCGTAGGCTCGATTTAATAAGTCATAGGCAGCGGCTCAGTGGTCAGGGTGCTACCGAATTCGTCAACCGGAGTCTTTTATTGTATATCCAGGTAAATTCAGATTACCTGCAGCGCTGTTTTCCTTGTTTCACATCCAACTTCTGCGTGTCAACGCA
TTTAAGTCCCTGTTCGTCCTATTCTGGGTTAATCATGGTCGCCGCGGAAACGTGCCTAACTCTTTTACTTGTTGCATTGAGACTCGATTTGCCGGTTAATCAGAAGGTAATGTCCGGTCAGCTCCGTTCTTGGTCGCCGACATAGGTGCGAACACAGCCGCCA
GCAGCGAGCGGTCTATGACGCACTCCATGTTGGATAGTCAATCCCCACAAAGCAAGTCTGCTATCGACTCCCCAAGGACATGTCAATCTGGTTCTGCATCGGGCGGCAGGGTGGAGAGAGCTCCAGCTGAGTTCGTCGCGCCCCTGGCTCTACTTACTACA
TTCGCGTCCCGC

1. Randomized Motif Search

Initially, we chose arbitrary motifs from every line in this method. First, the randomly chosen motifs are the best score and best motif. The profile matrix of this motif was then calculated in a while loop, and from the profile matrix, a new motif matrix was created. We select the motif with the highest probability for every line. We determined the new motif's score. If it is lower than the best motif's score, we updated the best motif's score to the new motif's score, and we carried on with the algorithm. We have completed the procedure if the new motif's score is greater than or equal to the top score.

```python
def randomized_motif_search(dna, k):
    best_motifs = create_random_motifs(dna, k)
    best_score = score_motifs(best_motifs, k)
    score_history = [best_score]

    while True:
        i = random.randint(0, len(dna[0]) - k)
        motifs = [d[i:i+k] for d in dna]
        current_score = score_motifs(motifs, k)

        if current_score < best_score:
            best_motifs, best_score = motifs, current_score
            score_history = [best_score]  # Reset since improvement occurred
        else:
            score_history.append(current_score)

        if len(score_history) > 50 and all(s >= best_score for s in score_history[-50:]): # Stuck
            break

    return best_motifs, best_score
```

RUN:

#1

```
k= 9
----------
Best Score:  {43}
Best Motifs:  ['CGTCTTAGT', 'TCTGCACCT', 'AACACAAGT', 'AGTATGAAT', 'AATACCGCG', 'GAACGAACT', 'CATACAAGC', 'ACCAATTGC', 'GTGACATTA', 'ATTCAGATT']
Consensus:  AATACAAGT
Execution Time:  2.6941299438476562e-05


k= 10
----------
Best Score:  {49}
Best Motifs:  ['CGTCTTAGTG', 'TCTGCACCTG', 'AACACAAGTA', 'AGTATGAATA', 'AATACCGCGT', 'GAACGAACTG', 'CATACAAGCT', 'ACCAATTGCA', 'GTGACATTAT', 'ATTCAGATTA']
Consensus:  AATACAAGTA
Execution Time:  2.9087066650390625e-05


k= 11
----------
Best Score:  {56}
Best Motifs:  ['GCACGTCTTAG', 'TAATCTGCACC', 'TCGAACACAAG', 'TCGAGTATGAA', 'TGGAATACCGC', 'GACGAACGAAC', 'CGCCATACAAG', 'GATACCAATTG', 'TCTGTGACATT', 'TAAATTCAGAT']
Consensus:  TAAAATACAAG
Execution Time:  3.0994415283203125e-05
```

#2

```
k= 9
----------
Best Score:  {46}
Best Motifs:  ['GCACGTCTT', 'TAATCTGCA', 'TCGAACACA', 'TCGAGTATG', 'TGGAATACC', 'GACGAACGA', 'CGCCATACA', 'GATACCAAT', 'TCTGTGACA', 'TAAATTCAG']
Consensus:  TAAAATACA
Execution Time:  2.5033950805664062e-05


k= 10
----------
Best Score:  {52}
Best Motifs:  ['CTGATAGCTG', 'AGGTAGCAAC', 'GGCACCAACC', 'CAATAAACGC', 'GTAGCTGCGT', 'GGCCGACAGC', 'GTGGAGTGGC', 'GTCAAGGAGC', 'TGGGAAAACG', 'CCAGCTGAGT']
Consensus:  GGGGAAGAGC
Execution Time:  2.5987625122070312e-05


k= 11
----------
Best Score:  {55}
Best Motifs:  ['GTCTTAGTGTG', 'CTGCACCTGAG', 'ACACAAGTATG', 'GTATGAATATC', 'ATACCGCGTCG', 'AACGAACTGAA', 'ATACAAGCTGC', 'CCAATTGCAGG', 'TGACATTATTA', 'TTCAGATTACC']
Consensus:  ATACAAGTATG
Execution Time:  2.7179718017578125e-05
```

#3

```
k= 9
----------
Best Score:  {46}
Best Motifs:  ['CAAATAAGC', 'TGAGGAATA', 'ATGGTGTGG', 'GGCTAAGCA', 'CTATCGACA', 'ATGTATTTT', 'GAACAGAAC', 'AAGTAAATC', 'ATAGAAGCC', 'AAGCAAGTC']
Consensus:  AAATAAATC
Execution Time:  3.2901763916015625e-05


k= 10
----------
Best Score:  {50}
Best Motifs:  ['GTCTTAGTGT', 'CTGCACCTGA', 'ACACAAGTAT', 'GTATGAATAT', 'ATACCGCGTC', 'AACGAACTGA', 'ATACAAGCTG', 'CCAATTGCAG', 'TGACATTATT', 'TTCAGATTAC']
Consensus:  ATACAAGTAT
Execution Time:  3.0994415283203125e-05


k= 11
----------
Best Score:  {59}
Best Motifs:  ['CCGCACGTCTT', 'ATTAATCTGCA', 'TGTCGAACACA', 'TATCGAGTATG', 'AATGGAATACC', 'TAGACGAACGA', 'CACGCCATACA', 'CCGATACCAAT', 'GTTCTGTGACA', 'GGTAAATTCAG']
Consensus:  CATAAAATACA
Execution Time:  2.9802322387695312e-05
```

#4

```
k= 9
----------
Best Score:  {44}
Best Motifs:  ['GTCTTAGTG', 'CTGCACCTG', 'ACACAAGTA', 'GTATGAATA', 'ATACCGCGT', 'AACGAACTG', 'ATACAAGCT', 'CCAATTGCA', 'TGACATTAT', 'TTCAGATTA']
Consensus:  ATACAAGTA
Execution Time:  2.384185791015625e-05


k= 10
----------
Best Score:  {53}
Best Motifs:  ['AGCCCGGAGA', 'CAACACCAGT', 'GCTCCGAGGC', 'CAAAGCAGGA', 'TGAGACGTAA', 'TCATCAGGCT', 'CGTCTCATCA', 'TAGCGATGGA', 'GCATGGTGCG', 'TCAGAAGGTA']
Consensus:  TCACACGGGA
Execution Time:  2.574920654296875e-05


k= 11
----------
Best Score:  {56}
Best Motifs:  ['CACGTCTTAGT', 'AATCTGCACCT', 'CGAACACAAGT', 'CGAGTATGAAT', 'GGAATACCGCG', 'ACGAACGAACT', 'GCCATACAAGC', 'ATACCAATTGC', 'CTGTGACATTA', 'AAATTCAGATT']
Consensus:  AAAATACAAGT
Execution Time:  2.6702880859375e-05
```

#5

```
k= 9
----------
Best Score:  {46}
Best Motifs:  ['GATAGCTGA', 'GTAGCAACA', 'CACCAACCT', 'ATAAACGCC', 'AGCTGCGTC', 'CCGACAGCA', 'GGAGTGGCA', 'CAAGGAGCA', 'GGAAAACGT', 'AGCTGAGTT']
Consensus:  GGAAGAGCA
Execution Time:  2.9087066650390625e-05


k= 10
----------
Best Score:  {52}
Best Motifs:  ['CACGTCTTAG', 'AATCTGCACC', 'CGAACACAAG', 'CGAGTATGAA', 'GGAATACCGC', 'ACGAACGAAC', 'GCCATACAAG', 'ATACCAATTG', 'CTGTGACATT', 'AAATTCAGAT']
Consensus:  AAAATACAAG
Execution Time:  3.0994415283203125e-05


k= 11
----------
Best Score:  {56}
Best Motifs:  ['GCACGTCTTAG', 'TAATCTGCACC', 'TCGAACACAAG', 'TCGAGTATGAA', 'TGGAATACCGC', 'GACGAACGAAC', 'CGCCATACAAG', 'GATACCAATTG', 'TCTGTGACATT', 'TAAATTCAGAT']
Consensus:  TAAAATACAAG
Execution Time:  3.528594970703125e-05
```

2. Gibbs Sampler

In this algorithm, we again randomly selected motifs from each row. These motifs are the best motif and the best score at the beginning. In this algorithm we have a counter to count the number of iterations. In a while loop, we choose a random motif at each iteration. We then calculated the probability of each motif based on the profile matrix. Again, we calculate the score of new motifs and perform the same operations as in the random motif search. The end condition of this algorithm is that the algorithm checks the best score every 50 iterations. If the best score is not increased, the algorithm terminates.

```python
# Gibbs Sampler
def gibbs_sampler(dna, k):
    motifs = random_motifs(dna, k)
    best_motifs = motifs[:]
    best_score = score(motifs)
    scores = [best_score]  # Keep track of the scores

    while True:
        start_time = time.time()

        i = random.randint(0, len(dna) - 1)
        motifs_except_i = motifs[:i] + motifs[i+1:]
        profile = profile_with_pseudocounts(motifs_except_i)
        motifs[i] = weighted_random_kmer(dna[i], k, profile)
        current_score = score(motifs)

        if current_score < best_score:
            best_motifs = motifs[:]
            best_score = current_score
            scores.append(best_score)  # Update
        else:
            scores.append(current_score) # Keep

        if len(scores) > 50 and scores[-1] == scores[-51]:
            break  # Stuck

    end_time = time.time()
    execution_time = end_time - start_time

    return best_motifs, best_score, execution_time
```

RUN:

#1

```
k= 9
----------
Best Score:  {32}
Best Motifs:  ['ACGTCTTAG', 'AGATCTAGG', 'CTGGCTTAT', 'AGATTTTTT', 'CGCTTTTAT', 'TGGGCTTGT', 'AGAGCTTGT', 'TCACCTTGG', 'AAAGCATGG', 'CAATCTGGT']
Consensus:  AGATCTTGT
Execution Time: 0.0004248619 seconds


k= 10
----------
Best Score:  {41}
Best Motifs:  ['TCCCAAGCCC', 'TTGCAGTCCT', 'AACCTAAGCT', 'ATCCAAAGGC', 'ACCTAGAACC', 'TCAGGAAGTG', 'CACCAAAGTA', 'CACCTAGGTC', 'AATCGAAGAT', 'CCACAAAGCA']
Consensus:  AACCAAAGCC
Execution Time: 0.0003848076 seconds


k= 11
----------
Best Score:  {5}
Best Motifs:  ['GCCACAAAGCA', 'CCCACAAAGCA', 'CCCACAAAGCA', 'TCCACAAAGCA', 'TCCACAAAGCA', 'ACCACAAAGCA', 'CCCACAAAGCA', 'CCCACAAAGCA', 'TCCACAAAGCA', 'CCCACAAAGCA']
Consensus:  CCCACAAAGCA
Execution Time: 0.0003869534 seconds
```

#2

```
k= 9
----------
Best Score:  {0}
Best Motifs:  ['CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA', 'CACAAAGCA']
Consensus:  CACAAAGCA
Execution Time: 0.0003590584 seconds


k= 10
----------
Best Score:  {38}
Best Motifs:  ['TGTTTCCTCT', 'AGACTCCTTA', 'ACTAACCTAA', 'CCATGCCACA', 'CGTTTGCTCA', 'GGCTGCCGGA', 'AGCTGCCGAA', 'AGCAGTCACT', 'GGATACGTAA', 'ACGTGCCTAA']
Consensus:  AGATGCCTAA
Execution Time: 0.0004088879 seconds


k= 11
----------
Best Score:  {41}
Best Motifs:  ['GTTCATCAAGT', 'AGCCATATTTG', 'CGGCCTCTGTC', 'ATTCATCTTAC', 'CTCCAATTGTA', 'CTTAAGTTTTA', 'ATTCATTGATA', 'CTCCATCCTTG', 'GTTAATCGAAG', 'CTCCATGTTGG']
Consensus:  CTTCATCTTTG
Execution Time: 0.0003840923 seconds
```

#3

```
k= 9
----------
Best Score:  {27}
Best Motifs:  ['ATAAGCCTA', 'CCGCAGAAA', 'CCATACTAA', 'ATAAACTAA', 'ACACTGCAA', 'ATAATGCTA', 'ACAAAGCAA', 'CTATTGCTA', 'ACAAACCGA', 'ACAAAGCAA']
Consensus:  ACAAAGCAA
Execution Time: 0.0003631115 seconds


k= 10
----------
Best Score:  {32}
Best Motifs:  ['CGATGATCGG', 'TAACGATAGC', 'TTCCGATCGC', 'TTACGACAGA', 'TAACAATGCA', 'TTACGATAAC', 'CAACAACGGT', 'TAGCGATGGA', 'GAACATTCGC', 'TAATCATGGT']
Consensus:  TAACGATGGC
Execution Time: 0.0003862381 seconds


k= 11
----------
Best Score:  {40}
Best Motifs:  ['CTGATAGCTGA', 'ATATCCGCAGA', 'CTTATCGTTGA', 'CTGGCGGTTCA', 'ACTTTCGTACA', 'GTGTTCGATGA', 'CAGAACGTTGA', 'GTTATTGTTGA', 'GGGTATTAGGA', 'GCGGTCTATGA']
Consensus:  CTGATCGTTGA
Execution Time: 0.0004138947 seconds
```

#4

```
k= 9
----------
Best Score:  {35}
Best Motifs:  ['TGACACATC', 'CTACACATC', 'CGTCAATGC', 'CTAAGCATC', 'CGTCACGCT', 'CTTCTTGCA', 'CGACTCCGC', 'TTGCTAGTG', 'CCACATGTG', 'GGACATGTC']
Consensus:  CGACACGTC
Execution Time: 0.0003700256 seconds


k= 10
----------
Best Score:  {39}
Best Motifs:  ['AGGCCGGGCT', 'CTTAGTGCGG', 'TTGCTGCGGT', 'GTGCGGACAC', 'CTGCGTCCTT', 'CGGCTGCCGG', 'CTGCTGTCGT', 'GGTTGCGCCT', 'CCTCTGGCTT', 'CTGAGTTCGT']
Consensus:  CTGCGGGCGT
Execution Time: 0.0003497601 seconds


k= 11
----------
Best Score:  {41}
Best Motifs:  ['CTGATCCGCGA', 'ATGATGAGTTT', 'AAGAACAGCCA', 'ACCATGCAATA', 'AACAAGCGACT', 'ATGATGCAATA', 'CTCATCAGAGA', 'ATGTAGACACA', 'ATGGTGCGCTA', 'CACATCCAACT']
Consensus:  ATGATGCGACA
Execution Time: 0.0003719330 seconds
```

#5

```
k= 9
----------
Best Score:  {33}
Best Motifs:  ['GATCCGCGA', 'ACTCCGAAT', 'ACCCCGCTG', 'AAACGCCAT', 'AACTCCAAT', 'AATCTGACA', 'AATTTGCCA', 'AATATGCCT', 'ACTCTCCAC', 'AATCCCCAC']
Consensus:  AATCCGCAT
Execution Time: 0.0003957748 seconds


k= 10
----------
Best Score:  {11}
Best Motifs:  ['TGCCACAAAG', 'CCCCACAAAG', 'CCCCACAAAG', 'GTCCACAAAG', 'ATCCACAAAG', 'AACCACAAAG', 'GCCCACAAAG', 'CCCCACAAAG', 'TTCCACAAAG', 'CCCCACAAAG']
Consensus:  CCCCACAAAG
Execution Time: 0.0003471375 seconds


k= 11
----------
Best Score:  {40}
Best Motifs:  ['TAGCTGACACA', 'CACCAGTCACA', 'TTCCAGACTCA', 'CAGGAAATACT', 'TAGCTGCGTCC', 'ATGCAGAGATG', 'CACCGAACACA', 'TACGTGCCAAG', 'TACCATACTTA', 'TACCGAATTCG']
Consensus:  TACCAGACACA
Execution Time: 0.0003669262 seconds
```

3. Median String

With the Median String problem, we aimed to find the k-mer motif that is closest to all sequences within the DNA sequence. In this problem, we measured the “nearest” term by the total Hamming distance between the selected k-mer and the closest k-mer in each DNA sequence. Median String found the k-mer that minimizes this total distance.

```
# Median String
def median_string(dna, k):
    min_distance = float('inf')
    execution_time = time.time()
    for kmer in all_kmers(k):
        distance = sum(min(hamming_distance(kmer, seq[i:i+k]) for i in range(len(seq) - k + 1)) for seq in dna)
        if distance < min_distance:
            min_distance = distance
            median_kmer = kmer

    execution_time = time.time() - execution_time
    return median_kmer, execution_time
```

RUN:

```
Median String for k=9: CACAAAGCA
Median String for k=10: CCACAAAGCA
```

```
Median String for k=11:  CCCACAAAGCA
```

3. Outputs

# of RUN	Randomized Motif Search Score	k value
1	43	9
	49	10
	56	11
2	46	9
	52	10
	55	11
3	46	9
	50	10
	59	11
4	44	9
	53	10
	56	11
5	46	9
	52	10
	56	11

Average Score	k value
45	9
51	10
56	11

# of RUN	Gibbs Sampler Score	k value
1	32	9
	41	10
	5	11
2	0	9
	38	10
	41	11
3	27	9
	32	10
	40	11
4	35	9
	39	10
	41	11
5	33	9
	11	10
	40	11

Average Score	k value
25	9
32	10
33	11

When we compare these 3 algorithms, we can make the following conclusions:

- No significant difference was observed between the Gibbs Sampler and Randomized Motif Search algorithms in terms of execution time.

- It is understood from the scores in the tables above that Gibbs Sampler has better flexibility in avoiding local optima than the Randomized Motif Search algorithm for this project.

- It has been observed that Gibbs Sampler and Randomized Motif Search algorithms follow a parallel performance as the k value for k-mer increases.

- It is necessary to open separate parentheses for the Median String algorithm. Since this algorithm works with brute force logic, it can directly reach the perfect result and global optimum. However, as k value increases for k-mer, it reaches an unacceptable execution time. The execution time issue makes this algorithm useless even though it gives perfect results.