Applied Computational Genomics Homework 2

Github_repo: https://github.com/syneoxya/Computatiional_Genomics

Question 1

1a)

```python
import argparse
import random
from Bio import SeqIO

def mutate_sequence(seq, mutation_rate, seed):
    random.seed(seed)
    seq = list(seq)  # Convert to list for mutation
    num_mutations = int(len(seq) * mutation_rate)

    print(f"Total bases: {len(seq)}, Mutating {num_mutations} bases ({mutation_rate*100}%)")

    mutation_positions = random.sample(range(len(seq)), num_mutations)
    bases = ['A', 'C', 'G', 'T']

    for pos in mutation_positions:
        original_base = seq[pos].upper()
        if original_base not in bases:
            continue  # skip N or other ambiguous bases
        new_base = random.choice([b for b in bases if b != original_base])
        seq[pos] = new_base

    return ''.join(seq)

def main():
    parser = argparse.ArgumentParser(description="Introduce random substitutions into a FASTA sequence.")
    parser.add_argument('-i', '--input', required=True, help='Input FASTA file')
    parser.add_argument('-o', '--output', required=True, help='Output FASTA file with mutations')
    parser.add_argument('-m', '--mutation_rate', type=float, required=True, help='Mutation rate (e.g., 0.015 for 1.5%)')
    parser.add_argument('-s', '--seed', type=int, required=True, help='Random seed for reproducibility')

    args = parser.parse_args()

    with open(args.input, 'r') as infile, open(args.output, 'w') as outfile:
        for record in SeqIO.parse(infile, 'fasta'):
            mutated_seq = mutate_sequence(str(record.seq), args.mutation_rate, args.seed)
            record.seq = mutated_seq
```

```
        SeqIO.write(record, outfile, 'fasta')

if __name__ == '__main__':
    main()
```

1b)

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.01_s1.fa

NUCMER

```
    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  |  [LEN R]  [LEN Q]  |  [COV R]  [COV Q]  | [TAGS]
=========================================================================================================================
       1  1000001  |       1  1000001  |  1000001  1000001  |   99.00  |  1000001  1000001  |   100.00   100.00  | chr22:20000000-21000000    chr22:20000000-21000000


==> chr22_orig_vs_mut0.01_s2.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.01_s2.fa

NUCMER


    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  |  [LEN R]  [LEN Q]  |  [COV R]  [COV Q]  | [TAGS]
=========================================================================================================================
       1  1000001  |       1  1000001  |  1000001  1000001  |   99.00  |  1000001  1000001  |   100.00   100.00  | chr22:20000000-21000000    chr22:20000000-21000000


==> chr22_orig_vs_mut0.01_s3.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
```

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.01_s3.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 1000001 | 1 1000001 | 1000001 1000001 | 99.00 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.05_s1.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.05_s1.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 1000001 | 1 1000001 | 1000001 1000001 | 95.00 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.05_s2.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.05_s2.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

```
==========================================================================
==========================================================

      1  1000001 |       1  1000001  |  1000001  1000001  |    95.00  |  1000001  1000001
|   100.00   100.00  | chr22:20000000-21000000    chr22:20000000-21000000


==> chr22_orig_vs_mut0.05_s3.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomi
cs/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomi
cs/hw2/chr22_mut0.05_s3.fa

NUCMER


    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  |  [LEN R]  [LEN Q]  |
[COV R]  [COV Q]  | [TAGS]

==========================================================================
==========================================================

      1  1000001 |       1  1000001  |  1000001  1000001  |    95.00  |  1000001  1000001
|   100.00   100.00  | chr22:20000000-21000000    chr22:20000000-21000000


==> chr22_orig_vs_mut0.10_s1.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomi
cs/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomi
cs/hw2/chr22_mut0.10_s1.fa

NUCMER


    [S1]     [E1]  |     [S2]     [E2]  |  [LEN 1]  [LEN 2]  |  [% IDY]  |  [LEN R]  [LEN Q]  |
[COV R]  [COV Q]  | [TAGS]

==========================================================================
==========================================================

      1  1000001 |       1  1000001  |  1000001  1000001  |    90.02  |  1000001  1000001
|   100.00   100.00  | chr22:20000000-21000000    chr22:20000000-21000000


==> chr22_orig_vs_mut0.10_s2.coords <==
```

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.10_s2.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 999996 | 1 999996 | 999996 999996 | 90.02 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.10_s3.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.10_s3.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 999973 | 1 999973 | 999973 999973 | 90.02 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.15_s1.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.15_s1.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 1000001 | 1 1000001 | 1000001 1000001 | 85.09 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.15_s2.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.15_s2.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 999996 | 1 999996 | 999996 999996 | 85.09 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

==> chr22_orig_vs_mut0.15_s3.coords <==

/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_orig.fa
/Users/akshatchauhan/Desktop/JHU/Computational_Genomics/Computatiional_Genomics/hw2/chr22_mut0.15_s3.fa

NUCMER

[S1] [E1] | [S2] [E2] | [LEN 1] [LEN 2] | [% IDY] | [LEN R] [LEN Q] | [COV R] [COV Q] | [TAGS]

==========

1 999973 | 1 999973 | 999973 999973 | 85.09 | 1000001 1000001 | 100.00 100.00 | chr22:20000000-21000000 chr22:20000000-21000000

NUCMER alignments show declining sequence identity (% IDY: 99.00 to 85.09) and slight reductions in aligned lengths as mutation levels increase (0.01 to 0.15). Despite this, alignment coverage remains high (99.99–100%), indicating robust alignment. Random sampling results are consistent, demonstrating reliability. Higher mutations increase divergence, reducing similarity, but alignments remain comprehensive across the genome.

1c)

```
import argparse
import math

def clean_seq(seq):
    seq = seq.upper()
    return ''.join([c if c in "ACGT" else "N" for c in seq])

def read_fasta(path):
    with open(path) as f:
        seq = ''
        for line in f:
            if not line.startswith('>'):
                seq += line.strip()
    return clean_seq(seq)

def get_kmers(seq, k):
    return set(seq[i:i+k] for i in range(len(seq) - k + 1))

def compute_jaccard(A, B):
    intersection = len(A & B)
    union = len(A | B)
    return intersection / union if union else 0.0

def compute_ani(jaccard, k):
    # Exact formula
    if jaccard <= 0.0:
        return 0.0
    return jaccard ** (1 / k)

def compute_ani_approx(jaccard, k):
    # Approximation
    if jaccard <= 0.0:
        return 0.0
    return 1 + (math.log(jaccard) / k)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-a', required=True, help='Reference FASTA')
```

```
    parser.add_argument('-b', required=True, help='Mutated FASTA')
    parser.add_argument('-k', required=True, type=int, help='k-mer size')
    args = parser.parse_args()

    seq_a = read_fasta(args.a)
    seq_b = read_fasta(args.b)
    k = args.k

    kmers_a = get_kmers(seq_a, k)
    kmers_b = get_kmers(seq_b, k)
    jaccard = compute_jaccard(kmers_a, kmers_b)
    ani = compute_ani(jaccard, k)
    ani_approx = compute_ani_approx(jaccard, k)

    print(f'filename_a\tfilename_b\tjaccard\tani_exact\tani_approx')
    print(f'{args.a}\t{args.b}\t{jaccard:.6f}\t{ani:.6f}\t{ani_approx:.6f}')

if __name__ == '__main__':
    main()
```

1d)

Filename	Expected Identity	Jaccard	Exact ANI	Approximate ANI
chr22_mut0.01_s1.fa	0.99	0.677640	0.981640	0.981470
chr22_mut0.01_s2.fa	0.99	0.676857	0.981586	0.981415
chr22_mut0.01_s3.fa	0.99	0.677441	0.981626	0.981456
chr22_mut0.05_s1.fa	0.95	0.208039	0.927963	0.925237
chr22_mut0.05_s2.fa	0.95	0.206381	0.927610	0.924856
chr22_mut0.05_s3.fa	0.95	0.206193	0.927569	0.924812
chr22_mut0.10_s1.fa	0.90	0.061675	0.875762	0.867339
chr22_mut0.10_s2.fa	0.90	0.060300	0.874822	0.866265
chr22_mut0.10_s3.fa	0.90	0.060580	0.875016	0.866487
chr22_mut0.15_s1.fa	0.85	0.018422	0.826793	0.809800
chr22_mut0.15_s2.fa	0.85	0.018211	0.826340	0.809251
chr22_mut0.15_s3.fa	0.85	0.018345	0.826628	0.809599

The table of results shows how the Jaccard coefficient, exact ANI, and approximate ANI change with increasing mutation rate across different random seeds. As the mutation rate rises (from 0.01 to 0.15), the expected identity decreases (from 0.99 to 0.85), and the raw Jaccard similarity drops sharply, which is expected since each mutation affects multiple k-mers. The exact ANI (computed as Jaccard to the power of 1/k) consistently provides a close estimate to the expected sequence identity, even as similarity declines. For example, at a mutation rate of 0.10, the exact ANI values range around 0.875,

which is very near the expected identity of 0.90. The approximate ANI, which is based on a logarithmic formula, also tracks the expected identity reasonably well at low mutation rates, but begins to underestimate as divergence increases: at a mutation rate of 0.15, the approximate ANI is about 0.81 compared to the exact ANI of 0.83 and an expected identity of 0.85. Overall, these results show that the exact ANI transformation is robust and reliably reflects true sequence identity, while the approximate ANI formula is only accurate when the sequences are highly similar. This demonstrates that Jaccard-based ANI calculations are suitable for comparing closely related sequences, but the logarithmic approximation should be used with caution as divergence increases

1e)
```python
#!/usr/bin/env python3


import argparse
import math
import zlib

def clean_seq(seq):
    seq = seq.upper()
    return ''.join([c if c in "ACGT" else "N" for c in seq])

def read_fasta(path):
    with open(path) as f:
        seq = ''
        for line in f:
            if not line.startswith('>'):
                seq += line.strip()
    return clean_seq(seq)

def get_modimizers(seq, k, m):
    mods = set()
    for i in range(len(seq) - k + 1):
        kmer = seq[i:i+k]
        h = zlib.crc32(kmer.encode('utf-8')) & 0xffffffff
        if h % m == 0:
            mods.add(kmer)
    return mods

def compute_jaccard(A, B):
    intersection = len(A & B)
    union = len(A | B)
    return intersection / union if union > 0 else 0.0

def compute_ani(jaccard, k):
    if jaccard <= 0.0:
        return 0.0
    return jaccard ** (1 / k)
```

```python
def compute_ani_approx(jaccard, k):
    if jaccard <= 0.0:
        return 0.0
    return 1 + (math.log(jaccard) / k)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-a', required=True, help='Reference FASTA')
    parser.add_argument('-b', required=True, help='Mutated FASTA')
    parser.add_argument('-k', required=True, type=int, help='k-mer size')
    parser.add_argument('-m', required=True, type=int, help='mod value for modimizer sampling')
    args = parser.parse_args()

    seq_a = read_fasta(args.a)
    seq_b = read_fasta(args.b)
    k = args.k
    m = args.m

    mods_a = get_modimizers(seq_a, k, m)
    mods_b = get_modimizers(seq_b, k, m)

    jaccard = compute_jaccard(mods_a, mods_b)
    ani = compute_ani(jaccard, k)
    ani_approx = compute_ani_approx(jaccard, k)

    print(f'filename_a\tfilename_b\tmod_value\tmodimizers_a\tmodimizers_b\tjaccard\tani_exact\tani_approx')
    print(f'{args.a}\t{args.b}\t{m}\t{len(mods_a)}\t{len(mods_b)}\t{jaccard:.6f}\t{ani:.6f}\t{ani_approx:.6f}')

if __name__ == '__main__':
    main()
```

1f)

Filename	Mod Value	Modimizers (Orig)	Modimizers (Mut)	Jaccard	ANI_Exact	ANI_Approx
chr22_mut0.01_s1.fa	100	9290	9521	0.675067	0.981462	0.981288
chr22_mut0.01_s1.fa	1000	931	957	0.675244	0.981475	0.981301
chr22_mut0.01_s2.fa	100	9290	9464	0.682275	0.981959	0.981794
chr22_mut0.01_s2.fa	1000	931	933	0.668756	0.981023	0.980841
chr22_mut0.01_s3.fa	100	9290	9384	0.679921	0.981797	0.981630

chr22_mut0.01_s3.fa	1000	931	970	0.685284	0.982165	0.982004
chr22_mut0.05_s1.fa	100	9290	9775	0.209324	0.928235	0.925530
chr22_mut0.05_s1.fa	1000	931	922	0.211903	0.928777	0.926113
chr22_mut0.05_s2.fa	100	9290	9912	0.212707	0.928944	0.926293
chr22_mut0.05_s2.fa	1000	931	1052	0.207674	0.927885	0.925153
chr22_mut0.05_s3.fa	100	9290	9839	0.203006	0.926882	0.924071
chr22_mut0.05_s3.fa	1000	931	957	0.209481	0.928268	0.925566
chr22_mut0.10_s1.fa	100	9290	9945	0.063706	0.877114	0.868882
chr22_mut0.10_s1.fa	1000	931	1017	0.066229	0.878738	0.870731
chr22_mut0.10_s2.fa	100	9290	9922	0.061847	0.875878	0.867472
chr22_mut0.10_s2.fa	1000	931	1009	0.068871	0.880376	0.872594
chr22_mut0.10_s3.fa	100	9290	9927	0.063064	0.876691	0.868399
chr22_mut0.10_s3.fa	1000	931	984	0.056843	0.872367	0.863454
chr22_mut0.15_s1.fa	100	9290	9948	0.020205	0.830438	0.814198
chr22_mut0.15_s1.fa	1000	931	977	0.018687	0.827355	0.810479
chr22_mut0.15_s2.fa	100	9290	9916	0.018184	0.826281	0.809180
chr22_mut0.15_s2.fa	1000	931	1033	0.026660	0.841474	0.827400
chr22_mut0.15_s3.fa	100	9290	9971	0.018723	0.827432	0.810572
chr22_mut0.15_s3.fa	1000	931	1040	0.015979	0.821212	0.803026

Using modimizers for ANI calculation, as shown in the table above, allows for dramatic reduction in memory and computational requirements with minimal loss of accuracy. When sampling kmers with a mod value of 100, each sequence yielded about 9300 modimizers, while a mod value of 1000 reduced that to around 930—vastly fewer than the approximately 1 million kmers present in the full-length sequence. Remarkably, the exact ANI and approximate ANI values computed from these modimizers remain very close to those obtained using all kmers, especially at lower mutation rates. For example, with a mutation rate of 0.01, the ANI values derived from modimizers (both m=100 and m=1000) are within 0.001 of the expected value. At higher mutation rates or lower sampling (m=1000), there is slightly more variability, but the estimates still track the expected sequence identity accurately. This demonstrates that modimizer sampling is highly efficient: a tiny fraction of kmers can reliably represent sequence similarity, making it feasible to compute ANI on large genomes quickly and with minimal memory usage, while maintaining nearly the same biological interpretability as full kmer analysis.

Question 2

2a)

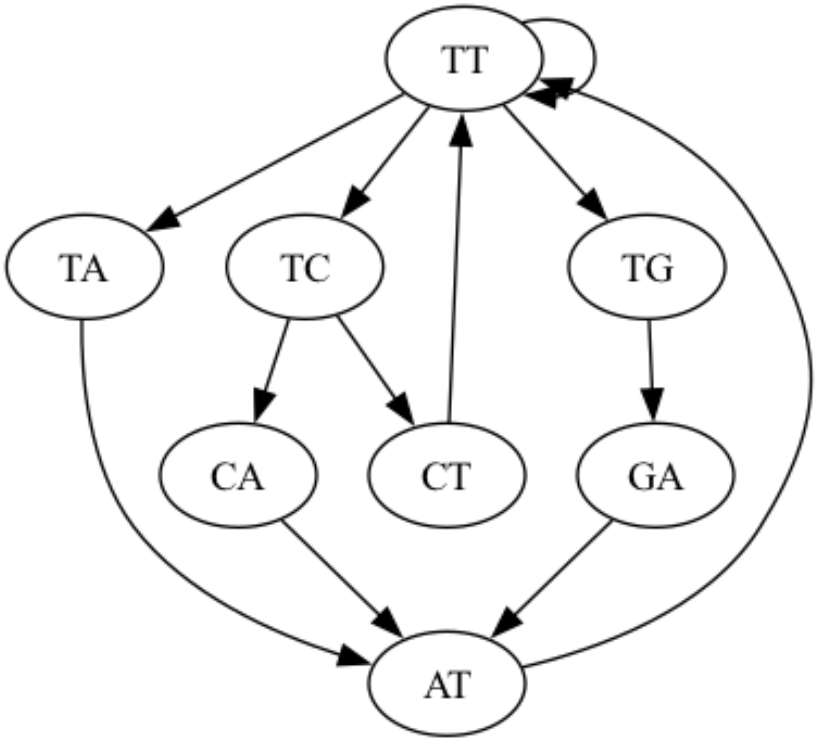
```
reads = [
    "ATTCA",
    "ATTGA",
    "CATTG",
    "CTTAT",
    "GATTG",
    "TATTT",
    "TCATT",
    "TCTTA",
    "TGATT",
    "TTATT",
    "TTCAT",
    "TTCTT",
    "TTGAT"
]

k = 3  # length of kmer
edges = set()
nodes = set()

for read in reads:
    for i in range(len(read) - k + 1):
        kmer = read[i:i+k]
        src = kmer[:k-1]
        dst = kmer[1:]
        nodes.add(src)
        nodes.add(dst)
```

```
        edges.add((src, dst))

# Output in DOT format
print('digraph debruijn {')
for src, dst in edges:
    print(f'    "{src}" -> "{dst}";')
print('}')
```



2b) From the graph above, the full k-mer list is:

ATT, TTT, TTA, TTC, TTG, TCA, TCT, TGA, GAT, TAT, CAT, CTT

From AT → TT(add T) → ATT

TT → TG (add G) → ATTG

TG → GA (add A) → ATTGA

GA → AT (add T) → ATTGAT

AT → TT (add T) → ATTGATT

TT → TC (add C) → ATTGATTC

TC → CA (add A) → ATTGATTCA

CA → AT (add T) → ATTGATTCAT

AT → TT (add T) → ATTGATTCATT

TT → TT (add T) → ATTGATTCATTT

TT → TA (add A) → ATTGATTCATTTA

TA → AT (add T) → ATTGATTCATTTAT

AT → TT (add T) → ATTGATTCATTTATT

TT → TC (add C) → ATTGATTCATTTATTC

TC → CT (add T) → ATTGATTCATTTATTCT

CT → TT (add T) → ATTGATTCATTTATTCTT

So the final genome sequence is:

ATTGATTCATTTATTCTT

2c) To completely solve the genome, you need more information to fill in the gaps and clear up any confusion caused by repeating pieces. This could be longer DNA reads that stretch across tricky areas, or special reads that link distant parts of the genome together. If you had reads that cover the whole genome without missing bits, and with no errors, you could confidently figure out the exact order of all the pieces and fully reconstruct the genome.

Question 3

3a) 932 k-mers occur 50 times

3b) Top 10 most occurring kmers are:
93 AGGTTCAATTCCTGCCGGGCG
91 GCGCCCGGCAGGAATTGAACC
91 CGCGCCCGGCAGGAATTGAAC
90 GGCGCGCCCGGCAGGAATTGA
90 GCGCGCCCGGCAGGAATTGAA
90 GCAGGAATTGAACCTGCGACC
90 GAAGGTCGCAGGTTCAATTCC
90 CCGGCAGGAATTGAACCTGCG
90 CCCGGCAGGAATTGAACCTGC
90 CAGGTTCAATTCCTGCCGGGC

3c) The estimated genome size is **233,492 bp**

3d) The reference genome size is 233,806bp and the estimated genome size is 233,492 bp. The

estimation is pretty close to the genome size.

Question 4

Note: I installed spades according to the use guide, but my assembler finishes in 2.8 seconds even though it says that it should take a few minutes on the homework page. I do not know if my output is incorrect and where I messed up. I have linked the github repo which has the output log under **question4_5/asm/spadeslog** Honestly I was stuck on this for hours. Kindly help me to locate where I messed up.

4a) grep -c '>' asm/contig.fasta
4 contigs were produced

4b) (asn2) akshatchauhan@Akshats-MacBook-Pro bin % samtools faidx asm/contigs.fasta
(asn2) akshatchauhan@Akshats-MacBook-Pro bin % awk '{sum += $2} END {print sum}' asm/contigs.fasta.fai
Total length of contigs = 234569bp

4c) (asn2) akshatchauhan@Akshats-MacBook-Pro bin % sort -nrk 2 asm/contigs.fasta.fai | head -n 1

Length of largest contig: 105834bp

4d) (asn2) akshatchauhan@Akshats-MacBook-Pro bin % awk '{print $2}' asm/contigs.fasta.fai | sort -nr > contig_lengths.txt
(asn2) akshatchauhan@Akshats-MacBook-Pro bin % awk '{sum+=$1; if (sum >= TOTAL/2) {print $1; exit}}' TOTAL=$(awk '{sum += $1} END {print sum}' contig_lengths.txt) contig_lengths.txt

Length of N50 contig: 47849bp

Question 5

5a) asn2) akshatchauhan@Akshats-MacBook-Pro bin % dnadiff ref.fa asm/contigs.fasta
Building alignments
Filtering alignments
Extracting alignment coordinates
Analyzing SNPs
Extracting alignment breakpoints
Generating report file
(asn2) akshatchauhan@Akshats-MacBook-Pro bin % grep -E "Insertion|Deletion" out.report

Reference (REF): 5 insertions.
Query (QRY): 1 insertion.

5b) (asn2) akshatchauhan@Akshats-MacBook-Pro bin % show-coords -rcl out.delta > coords.txt

The insertion is located in the assembly contig NODE_3_length_41445_cov_18.207587, which

has two alignments to the reference genome. The first alignment spans reference coordinates 3–26,789 and query coordinates 41,445–14,659 (reversed), while the second spans reference coordinates 26,788–40,639 and query coordinates 13,852–1 (also reversed). In the query contig, there is a gap of approximately 807 bases between the end of the first alignment (14,659) and the start of the second alignment (13,852), corresponding to the insertion in the assembly. This insertion, which does not appear in the reference genome, maps to around reference position 26,788–26,789, where the two alignments transition.

5c) The **insertion is 807 bases long**

5d) akshatchauhan@Akshats-MacBook-Pro bin % samtools faidx asm/contigs.fasta NODE_3_length_41445_cov_18.207587:13852-14659

DNA Sequence of encoded message is:
CCAGATTGCAATCGGGCCCGCTTCCGTCCCTTACAGCAGATCTGCAGATGATACTGGCCG
CAGCTGGGAATGTGTGAGGTCAATCCGCGATTAAGCGATTCAGGCTGACATCAAGTACGT
GGTTCGGCGCTGAATTTCCGAAGTGATAACTTTCTACAGAGGCTCATTTACGAAGGTTGG
AGTTCGGCAATACCCGACGGACGTAAGGTGGTAGTCACTTCCGTACCGCCTAGGCGACTA
ATAGCATCAAATACGCAAAGAATGTCCGACGTGTGGATTTACAGAAGCCCAGCAGGGGTG
CGGGTGTCACGATCGGAATAGCTTCAGCCTCTGATGGGTTATTAGGTGCTGCGGTTAGGC
GGGGATGCTGTGAGGCCTGACCGCATTAACGCTGTGACGTGCATAAGAATTTAAGTTGGG
CTGACTGAACGACTGTCCGCTGCTAATATAAGACTCCCATCCTTCACTCAGATATGAAGA
CATTTCGGGGTTCGGGTCGGAATCTCTTACTAGGCGCCTGGACGCCGTGTTACCGGGTGA
ACGCTGTTTCTTTGTGCTGCTATCGAGGGTCTGCGGCTCGTCTGTTTTCGTCTATACGTT
CGGCCTGGTTTCCGGGATTCCTTTTTGGAGTAGATTAATGGGAGCAGTTCTACAGGTTTG
CTTACCAGCAGGTAGCACTGTTGGATGCAGGAAATGACATAATACCCTACGCGGCGACTT
TCTCCCTAGTTGCCTGTACGACGCTCTACAGGTCTCCCTATGTGCCGGCGTCAAGCCCTT
ACTGCAATACGAACCGCCGACGTGGAGG

5e) (asn2) akshatchauhan@Akshats-MacBook-Pro bin % samtools faidx asm/contigs.fasta NODE_3_length_41445_cov_18.207587:13848-14650 > message.fa

The decoded message is: **ongratulations to the 2025 JHU Applied Genomics class!!!! Keep on looking for little green aliens...**