Ayaana Patel Sikora

Bi 181: Problem Set 3

Question 1

A. The EXACTMATCH algorithm calculates the range of matrix rows beginning with successively longer suffixes of the query. At each step, the size of the range either shrinks or remains the same. When the algorithm completes, rows beginning with $S_0$, which are the entire query, correspond to exact occurrences of the query in the text. If the range is empty, the text does not contain the query. This is insufficient for short read alignment because alignments may contain mismatches. Thus, when EXACTMATCH tries to align a mismatch, it fails.

B. The Bowtie algorithm conducts a quality-aware, greedy, randomized, depth first search through the space of possible alignments, which makes it an ultrafast-memory efficient alignment program. Bowtie indexes the reference genome using a scheme based on the Burrows Wheeler transform, which is a reversible permutation of the characters in a text. BWT-based indexing allows large texts to be searched efficiently in a small memory footprint. Bowtie does make compromises to achieve a high speed, which can be thought of as minor disadvantages. If one or more exact matches exist for a read, then Bowtie is guaranteed to report one, but if the best match is an inexact one then Bowtie is not guaranteed in all cases to find the highest quality alignment. It may fail to align a small number of reads with valid alignments, if those reads have multiple mismatches. However, Bowtie does support options that increase accuracy at the cost of some performance.

C. The purpose of the BWT algorithm is to allow large texts to be searched efficiently in a small memory footprint. It does this by appending a $ character to the text T, where $ is not in T and is lexicographically less than all characters in T. A matrix is then constructed whose rows comprise all cyclic rotations of T$. These rows are then sorted lexicographically. Because BWT has a small memory footprint, Bowtie exhibits a large performance advantage (in terms of speed and memory) over other alignment algorithms. Furthermore, Bowtie has a pretty good sensitivity in terms of reads aligned and creates a permanent index of the reference that may be re-used across alignment runs.

Question 2

See the attached code in the question2.py file

Question 3

See the attached code in the question3.py file

Question 4

This used the global alignment code used in question 6 (see question6.py)
The sequences passed to it were simply the sequences making up each pair.

**Pair 1:**
X1 = GCTGATATAGCT
X2 = GGGTGATTAGCT

Alignment:
-GCTGATATAGCT
GGGTGAT-TAGCT
Score = 7

**Pair 2:**
X1 = GCTGATATAGCT
X3 = GCTATCGC

Alignment:
GCTGATATAGCT
GC---TATCGC-
Score = 2

**Pair 3:**
X1 = GCTGATATAGCT
X4 = AGCGGAACACCT

Alignment:
-GCTGATATAGCT
AGCGGA-ACACCT
Score = 3

**Pair 4:**
X2 = GGGTGATTAGCT
X3 = GCTATCGC

Alignment:
GGGTGATTAGCT
-GCT-A-TCGC-
Score = 0

**Pair 5:**
X2 = GGGTGATTAGCT
X4 = AGCGGAACACCT

Alignment:
-GGGTGATTAGCT
AGCG-GAACACCT
Score = 1

**Pair 6:**
X3 = GCTATCGC
X4 = AGCGGAACACCT

Alignment:
-G C-TATC-GC-
AGCGGAACACCT
Score = -2

Thus, the best pairwise alignment is the first (the alignment of the first two sequences), because it has the highest score. Thus, we can say our new 4 sequences are:

X1 = -GCTGATATAGCT
X2 = GGGTGATTAGCT
X3 = GCTATCGC
X4 = AGCGGAACACCT

Now, we can find the alignment for x1 and x4, since that was the second best alignment.
Alignment:
-GCTGATATAGCT
AGCGGA-ACACCT
Score = 3

Now, we look at the new alignment for x1 and x2:
Alignment:
-GCTGATATAGCT
GGGTGAT-TAGCT
Score = 7

Now we align x1 and x3
Alignment:
-GCTGATATAGCT
-GC---TATCGC--
Score = 2

***Thus, the final multisequence alignment:***
X1 = -GCTGATATAGCT
X2 = GGGTGAT-TAGCT

X3 = -GC---TATCGC-
X4 = AGCGGA-ACACCT

## Question 5

This used the global alignment code used in question 6 (see question6.py)
The sequences passed to it were simply the sequences making up each pair.

Pair Human and Mouse
Pair Duck and Chicken
Pair Human’ and Chicken’
Pair Mouse’ and Duck’
Pair Human’ and Mouse’
Pair Duck’ and Chicken’

Human X = TATAACAGGCTATCACCGGAT
Mouse Y = ACGTCAGGCTATCGCCGGA
Duck Z = ATAGCCTACCACGTGAG
Chicken W = AATAGGCTATCACCTGTGT

**Pair 1:**
Human and Mouse
Human’: TATAACAGGCTATCACCGGAT
Mouse’: -ACGTCAGGCTATCGCCGGA-
Score = 9

**Pair 2:**
Duck and Chicken
Duck’: -ATAGCCTACCACGTGAG-
Chicken’: AATAGGCTATCACCTGTGT
Score = 7

**Pair 3:**
Human’ and Chicken’
Human’: TATAACAGGCTATCACC-G-GAT
Chicken’: ---AATAGGCTATCACCTGTG-T
Score = 9

**Pair 4:**
Mouse’ and Duck’
Mouse’: -ACGTCAGGCTATCGC-CG-GA--
Duck’: -A--T-AGCCTA-C-CACGTGAG-
Score = 6

**Pair 5:**
Human’ and Mouse’
Human’: TATAACAGGCTATCAC-C-G-GA-T
Mouse’: -ACGTCAGGCTATCGC-C-G-GA--
Score = 13

**Pair 6:**
Duck’ and Chicken’
Duck’: -A----T-AGCCTA-C-CACGTGAG--
Chicken’: ----AAT-AGGCTA--TCACCTGTG-T

***Final Multisequence Alignment:***
TATAACAGGCTATCAC-C-G-GA-T
-ACGTCAGGCTATCGC-C-G-GA--
-A----T-AGCCTA-C-CACGTGAG--
----AAT-AGGCTA--TCACCTGTG-T

Question 6

See code in question6.py

Best pair after all pairwise alignments = Sequence 1 and Sequence 4 with a score of 139
Order of pairing:
1 and 4
2 and 4’
1’ and 2’
2’’ and 3
They were appropriately filled with gaps.

***Final Multi sequence alignment result:***

1. --------------GAGCCACATATCAGGGCA--AAGCAATGG-GCG---AGACCCCC---AG-GCC-CTGGCCAAAGCT-G-TGCAGGTTCACCAGGA-T-ACTCT---A-CGCACCATGTA-CTTCGCTTG-A--A-GG-CAGAA-CGC--TGT-TACC---TCAC-T-GGATAG--AAGAA-AGCTTTCCAAG--CCC----TG--G---G-AGCT---GTA-CC--ACCCAAA-TCCAGA-GGAAG--CA---AGG-CAG---AGGGAGGTGGGGT-C-GGA--AGGAG-TA-TAG-GA--G-----G----
2. --------------G--------------GCAG--AGCAATGG-GCG-G--GA-CC-C-CCAG-G-CCCTGGCCAAAGC-CG-TGCAGGTTCACCAGGA-T-ACTCTG----CGCACCATGTA-CTTCGCTTG-A--A-GG-CAGAA-CGC--TGT-TACC---TCAC-T-GGATAG--AAGAA-AGCTTTCCAAGC-CCC--A-A-------G-AGCT---GT-GCCG--CCCA-AATCCAGA-GGAAG--CA-G--GG--AGG--AGGGAGGTGGGGT-----A-G-GGAGG-A------AT-GC---------
3. ------------------------------------GCAATGGT-CGA---GATCCTCA--AGCGT---TGGCCAAAG--CGGTGCAGATTCACCACGACTC-C-CTGA----GGACCATGTAT-TTTGCCTGAATAAC--A-AAAAGCGCAC-GTCT-CCGGA-CACCTC-G--AGCC-AGAAC-------------CCCTG-G--------GT-

GCTAAA----------CCAG--TCCA-AT-GAAGCCCAC--------------------------------------------A--------
-----

4. --------------G--------------GCAG--AGCAATGG-GCG-G--GA-CC-C-CCAG-G-CCCTGGCCAAAGC-
CG-TGCAGGTTCACCAGGA-T-ACTCTG----CGCACCATGTA-CTTCGCTTG-A--A-GG-CAGAA-
CGC--TGT-TACC---TCAC-T-GGATAG--AAGAA-AGCTTTCCAAGC-CCC--A-A-------G-AGCT---
GT-GCCG--CCCA-AATCCAGA-GGAAG--CA-G--GG--AGG--AGGGAGGTGGGGT-----A-G-
GGAGG-A------AT-GC---------

Question 7

**TTG vs GCT**: Leu vs Ala = nonsynonymous substitution
1. _TG = 1/3 s, 2/3 ns, _CT = 0 s, 1 ns = (1/2)(1/3 s + 2/3 ns) + (1/2)(0 s + 1 ns) =1/6 s and 5/6 ns
2. T_G = 0 s, 1 ns, G_T = 0 s, 1 ns = (1/2)(0 s + 1 ns) + (1/2)(0 s + 1 ns) = 0 s and 1 ns
3. TT_ = 1/3 s, 2/3 ns, GC_ = 1 s, 0 ns = 2/3 s and 1/3 ns

**TCT vs CAT:** Ser vs. His = nonsynonymous substitution
4. TCT: 0 s, 1 ns, CAT: 0 s, 1 ns = 0 s and 1 ns
5. TCT: 0 s, 1 ns, CAT: 0 s, 1 ns = 0 s and 1 ns
6. TCT: 1 s, 0 ns, CAT: 1/3 s, 2/3 ns = 2/3 s and 1/3 ns

**AAT vs CCT:** Asn vs. Pro = ns sub
7. AAT: 0 s, 1 ns, CCT: 0 s, 1 ns = 0 s, 1 ns
8. AAT: 0 s, 1 ns, CCT: 0 s, 1 ns = 0 s, 1 ns
9. AAT: 1/3 s, 2/3 ns, CCT: 1 s, 0 ns = 2/3 s and 1/3 ns

**GTC vs TGC:** Val vs Cys = ns sub
10· GTC: 0 s, 1 ns, TGC: 0 s, 1 ns = 0 s, 1 ns
11. GTC: 0 s, 1 ns, TGC: 0 s, 1 ns = 0 s, 1 ns
12. GT_: 1 s, 0 ns, TG_: 1/3 s, 2/3 ns = 2/3 s, 1/3 ns

**ATT vs CCC**: Ile vs Pro = ns sub
13. _TT: 0 s, 1 ns, _CC: 0 s, 1 ns = 0s, 1 ns
14. A_T: 0 s, 1 ns, C_C: 0 s, 1 ns = 0s, 1 ns
15. AT_: 2/3 s, 1/3 ns, CC_: 1 s, 0 ns = 5/6 s, 1/6 ns

**CTC vs TCC:** Leu vs Ser = ns sub
16. _TC: 0 s, 1 ns, _CC: 0 s, 1 ns = 0s, 1 ns
17. C_C: 0s, 1 ns, T_C: 0s, 1 ns = 0s, 1 ns
18. CT_: 1s, 0 ns, TC_: 1s, 0ns = 1 s, 0 ns

**CTT vs CCC:** Leu vs Pro = ns sub
19. _TT: 0s, 1ns; _CC: 0s, 1ns = 0s, 1ns
20. C_T: 0s, 1ns; C_C: 0s, 1ns = 0s, 1ns
21: CT_: 1s, 0ns; CC_: 1s, 0ns = 1s, 0ns

**TCT vs TTC:** Ser vs. Phe = ns sub
22: _CT: 0s, 1ns; _TC: 0s, 1ns = 0s, 1ns
23: T_T: 0s, 1ns; T_C: 0s, 1ns = 0s, 1ns
24: TC_: 1s, 0ns; TT_: 1/3s, 2/3 ns = 2/3s, 1/3ns

**GTC vs CCC:** Val vs. Pro = ns sub
25: _TC: 0s, 1ns; _CC: 0s, 1ns
26: G_C: 0s, 1ns; C_C: 0s, 1 ns = 0s, 1ns
27: GT_: 1s, 0ns; CC_: 1s, 0ns = 1s, 0ns

**ATT vs CCT**: Lle vs Pro = ns sub
28: _TT: 0s, 1ns; _CT: 0s, 1ns
29: A_T: 0s, 1ns; C_T: 0s, 1ns
30: AT_: 2/3s, 1/3ns; CC_: 1s, 0ns = 5/6 s, 1/6 ns

**CAC vs CCC:** His vs Pro = ns sub
31: _AC: 0s, 1ns; _CC: 0s, 1ns = 0s, 1ns
32: C_C: 0s, 1ns; C_C: 0s, 1ns = 0s, 1ns
33: CA_: 1/3s, 2/3 ns; CC_: 1s, 0ns = 2/3s, 1/3ns

**TTG vs TCG**: Leu vs. Ser = ns sub
34: _TG: 1/3s, 2/3ns; _CG: 0s, 1ns = 1/6s, 5/6 ns
35: T_G: 0s, 1ns; T_G: 0s, 1ns = 0s, 1ns
36: TT_: 1/3s, 2/3ns; TC_: 1s, 0ns = 2/3s, 1/3ns

**CAG vs CAG:** no substitution
37: _AG: 0s, 1ns; _AG: 0s, 1ns = 0s, 1ns
38: C_G: 0s, 1ns; C_G: 0s, 1ns = 0s, 1ns
39: CA_: 1/3s, 2/3 ns; CA_: 1/3s, 2/3 ns = 1/3s, 2/3 ns

RIGHT SIDE

**GTG vs GTT:** Val vs Val = s sub
40. _TG: 0s, 1ns; _TT: 0s, 1ns = 0s, 1ns
41. G_G: 0s, 1ns; G_T: 0s, 1ns = 0s, 1ns
42. GT_: 1s, 0ns; GT_: 1s, 0ns = 1s, 0 ns

**GTC vs GTC**: no substitution
43. _TC: 0s, 1ns = 0s, 1ns
44. G_C: 0s, 1ns = 0s, 1ns
45. GT_: 1s, 0ns = 1s, 0ns

**CGA vs CGG**: Arg vs Arg = s sub

46. _GA: 1/3 s, 2/3 ns; _GG: 1/3s, 2/3 ns = 1/3s, 2/3 ns
47. C_A: 0s, 1ns; C_G: 0s, 1ns = 0s, 1ns
48. CG_: 1s, 0ns = 1s, 0ns

**GTG vs GTC**: Val vs Val = s sub
49. _TG: 1/3s, 2/3ns; _TC: 0s, 1ns = 1/6s, 5/6ns
50. G_G: 0s, 1ns; G_C: 0s, 1ns = 0s, 1ns
51. GT_: 1s, 0ns; GT_: 1s, 0ns = 1s, 0ns

**TGG vs TGG** = no substitution
52. _GG: 1/3s, 2/3ns = 1/3s, 2/3ns
53. T_G: 0s, 1ns = 0s, 1ns
54. TG_: 0s, 1ns = 0s, 1ns

**TTC vs TTC = no substitution**
55. _TC: 0s, 1ns = 0s, 1ns
56. T_C: 0s, 1ns = 0s, 1ns
57. TT_: 1/3s, 2/3ns = 1/3s, 2/3ns

**TGT vs TGC = no substitution**
58. _GT: 0s, 1ns; _GC: 0s, 1ns = 0s, 1ns
59. T_T: 0s, 1ns; T_C: 0s, 1ns = 0s, 1ns
60. TG_: 1/3s, 2/3ns; TG_: 1/3s, 2/3ns = 1/3s, 2/3ns

**AAC vs AAC = no substitution**
61. _AC: 0s, 1ns = 0s, 1ns
62. A_C: 0s, 1ns = 0s, 1ns
63. AA_: 1/3s, 2/3ns = 1/3s, 2/3ns

**CGG vs CGG = no substitution**
64. _GG: 1/3s, 2/3 ns = 1/3s, 2/3ns
65. C_G: 0s, 1ns = 0s, 1ns
66: CG_: 1s, 0ns = 1s, 0ns

**CGC vs CGT**: Arg vs Arg = s sub
67: _GC: 0s, 1ns; _GT: 0s, 1ns = 0s, 1ns
68: C_C: 0s, 1ns; C_T: 0s, 1ns = 0s, 1ns
69: CG_: 1s, 0ns; CG: 1s, 0ns = 1s, 0ns

**CAG vs CAG:** no substitution
70: _AG: 0s, 1ns = 0s, 1ns
71: C_G: 0s, 1ns = 0s, 1ns
72: CA_: 1/3s, 2/3 ns = 1/3s, 2/3 ns

**AAG vs AAA**: Lys vs Lys = s sub
73: _AG: 0s, 1ns; _AA: 0s, 1ns = 0s, 1 ns
74: A_G: 0s, 1 ns; A_A: 0s, 1ns = 0s, 1ns
75: AA_: 1/3s, 2/3 ns = 1/3s, 2/3 ns

**GGC vs GGC:** no substiution
76: _GC: 0s, 1ns = 0s, 1ns
77: G_C: 0s, 1ns = 0s, 1ns
78: GG_: 1s, 0ns = 1s, 0ns

A. Overall dN/dS ratio:

sum of nonsynonymous sites: 58.167
sum of synonymous sites: 19.83

number of nonsynonymous substitutions: 12
number of synonymous substitutions: 6

Therefore, dN = 12/58.167 = 0.20630
and dS = 5/19.83 = 0.302572

Therefore, the ratio is dN/dS = 0.20630/0.302572 = 0.6818

B. Left Half:

sum of ns sites: 29
sum of s sites: 10
number of ns subs = 12
number of s subs = 0

Therefore, dN = 12/29 = 0.41379
and dS = 0/10 = 0

Thus, dN/dS = indefinite

Right Half:

sum of ns sites: 29.167
sum of s sites: 9.83
number of ns subs = 0
number of s subs = 6

Therefore, dN = 0/29.167 = 0
and dS = 6/9.83 = 0.61038

Thus, dN/dS = 0

C. Clearly, the left half and right half when evaluated separately do not give a similar dN/dS ratio to the whole sequence. An indefinite dN/dS ratio for the left side tells us that there were no synonymous substitutions. This suggests positive Darwinian selection, since random mutations that led to new codons were favored. Meanwhile, the 0 for the right half dN/dS ratio tells us that there were no nonsynonymous substitutions on this side. This suggests negative Darwinian selection, which means that mutations that changed the codon were not favored. This suggests that the sequence was already optimized for the species. This tells us that particular sequences in a genome can evolutionarily transform will others in the same genome do not.

Question 8

A. We can first compute for the distance of Human and Horse by adding the path distances between them:

   0.005873 + 0.013037 + 0.013037 + 0.0365 + 0.0365 + 0.015682 + 0.006272 + 0.019763 + 0.0189280 + 0.012398 + 0.007287 + 0.099323 = 0.2846

B. After diverging from Elephant, the species experienced more divergences to get to the Tenrec, which means the ‘path’ to the Tenrec is longer.

C. The tree topology or structure would not change though the branch lengths would. This is because more divergences would occur, so the ‘path’ to the current species would be longer, which means branch lengths would increase.