**Pseudocode of each of the methods**

**Method ReadSequencesFromFile(filename)**

while there are lines in the file {

increment by one numOfLines

}

Create DNAsequence[] myDNAsequences ← new DNAsequence[numOfLines];

for each of the lines {

line← readLine and split(" ");

seqName ← line[0];

seq ← line[1];

Create myDNAsequences[i] ← new DNAsequence(seqName,seq,seq.length());

}

return myDNAsequences;

**Method ReadTargetsFromFile(filename)**

while there are lines in the file {

increment by one numOfLines

}

Create String[]myTargets← new String [numOfLines];

for each of the lines {

myTargets[i] ← textReader.readLine();

}

return myDNAsequences;

**Method PrintSequenceArray (DNAsequence[] A)**

for each of the DNAsequences {

A[i].Print(); //using Print ()method in class file.

}

**Method FindBestMatchSequence(DNAsequence[] A, String[] targets)**

double avgHighest ← 0;

double avg ← 0;

DNAsequence sequenceHighest ← A[0];

double matches ← 0;

double matchesTotal ← 0;

for each of the DNAsequences {

matchesTotal ← 0;

for each of the targets {

if (DNAsequences is not empty) {

matches ← A[i].countSubStringMatch(targets[j]);

matchesTotal ← matchesTotal + matches;

}

avg ← matchesTotal / targets.length;

if (avg is greater than avgHighest) {

avgHighest ← avg;

sequenceHighest ← A[i];

}

}

return sequenceHighest;

**Method SortByBestOccurrenceAverage (DNAsequence[] A, String[] targets)**

Double avg arrat ← new double[DNASequeneces];

double matches ← 0;

double matchesTotal ← 0;

for each of the DNAsequences {

matchesTotal ← 0;

for each of the targets {

if (DNAsequences is not empty) {

matches ← A[i].countSubStringMatch(targets[j]);

matchesTotal ← matchesTotal + matches;

}

}

avg[i] ← matchesTotal / targets.length;

}

double avgHighest ← 0;

int indexHighest ← 0;

double tempAvg ← 0;

String tempName ← "";

Create new DNAsequence tempDNAsequences ← new DNAsequence("temp","A",1);

for each of the DNAsequences {

avgHighest ← avg[i];

indexHighest ← i;

for each of the DNAsequences starting at i {

if (avg[j] > avgHighest) {

avgHighest ← avg[j];

indexHighest ← j;

}

}

//Once found the highet, swap highest avg array to i position

tempAvg ← avg[i];

avg[i] ← avg[indexHighest];

avg[indexHighest] ← tempAvg;

//swap dnaSequences

tempDNAsequences.setName(A[i].getName());

A[i].setName(A[indexHighest].getName());

A[indexHighest].setName(tempDNAsequences.getName());

tempDNAsequences.setSequence(A[i].getSequence());

A[i].setSequence(A[indexHighest].getSequence());

A[indexHighest].setSequence(tempDNAsequences.getSequence());

A[i].setLength(A[i].getSequence().length());

A[indexHighest].setLength(A[indexHighest].getSequence().length());

}

Print Sorted Sequence Array;

**Method SortByLetter (DNAsequence[] A, char c)**

String theSequence ← "";

int counter ← 0;

int[] characterTotal ← new int[A.length];

for each of the DNAsequences {

theSequence ← A[i].getSequence();

for each of the sequence characters {

if(theSequence.charAt(j) == c) {

increase counter by one;

}

}

characterTotal[i] ← counter;

counter ← 0;

if (c is equal to 'A'){

A[i].setAs(characterTotal[i]);

}

if (c is equal to 'C') {

A[i].setCs(characterTotal[i]);

}

if (c is equal to 'G') {

A[i].setGs(characterTotal[i]);

}

else if (c equal to 'T') {

A[i].setTs(characterTotal[i]);

}

}

int charCountHighest ← 0;

int indexHighest ← 0;

int tempCharCount ← 0;

Create new DNAsequence tempDNAsequences ← new DNAsequence("temp","A",1);

for each of the DNAsequences {

charCountHighest ← characterTotal[i];

indexHighest ← i;

for each of the sequence characters {

if (characterTotal[j] > charCountHighest) {

charCountHighest ← characterTotal[j];

indexHighest ← j;

}

}

//Once found the highet, swap highest characterTotal array to i position

tempCharCount ← characterTotal[i];

characterTotal[i] ← characterTotal[indexHighest];

characterTotal[indexHighest] ← tempCharCount;

//swap dnaSequences the same way as characterTotal array

tempDNAsequences.setName(A[i].getName());

A[i].setName(A[indexHighest].getName());

A[indexHighest].setName(tempDNAsequences.getName());

tempDNAsequences.setSequence(A[i].getSequence());

A[i].setSequence(A[indexHighest].getSequence());

A[indexHighest].setSequence(tempDNAsequences.getSequence());

A[i].setLength(A[i].getSequence().length());

A[indexHighest].setLength(A[indexHighest].getSequence().length());

if (c equals to 'A'){

tempDNAsequences.setAs(A[i].getAs());

A[i].setAs(A[indexHighest].getAs());

A[indexHighest].setAs(tempDNAsequences.getAs());

}

else if (c equals to 'C') {

tempDNAsequences.setCs(A[i].getCs());

A[i].setCs(A[indexHighest].getCs());

A[indexHighest].setCs(tempDNAsequences.getCs());

}

else if (c equals to 'G') {

tempDNAsequences.setGs(A[i].getGs());

A[i].setGs(A[indexHighest].getGs());

A[indexHighest].setGs(tempDNAsequences.getGs());

}

else if (c equals to 'T') {

tempDNAsequences.setTs(A[i].getTs());

A[i].setTs(A[indexHighest].getTs());

A[indexHighest].setTs(tempDNAsequences.getTs());

}

}

Print Sorted Sequence Array;

**Pseudocode of each of the methods**

**ATTRIBUTES**

private String name;

private String sequence;

private int length;

private int As;

private int Cs;

private int Gs;

private int Ts;

**CONSTRUCTORS**

Default Constructor: It takes no parameters.

DNAsequence () {

}

Constructor that takes the following parameters

String seqName

String seq

int seqlength

DNAsequence(String seqName, String seq, int seqlength) {

name = seqName;

sequence = seq;

length = seqlength;

}