Homework 3 - Write up

💡 Consulted with @Jeevan Prakash

1. to run the program

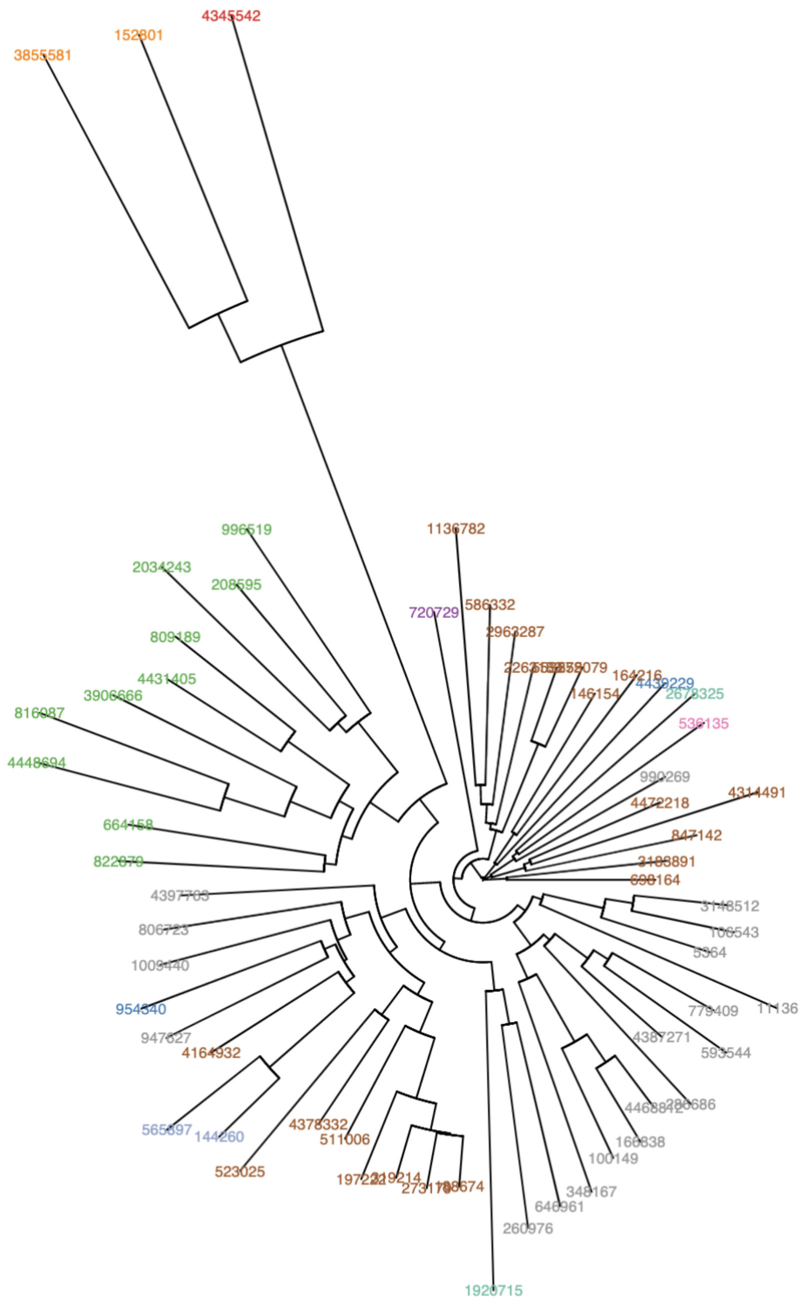
```
javac Homework3Driver.java
java Homework3Driver hw3.fna
```

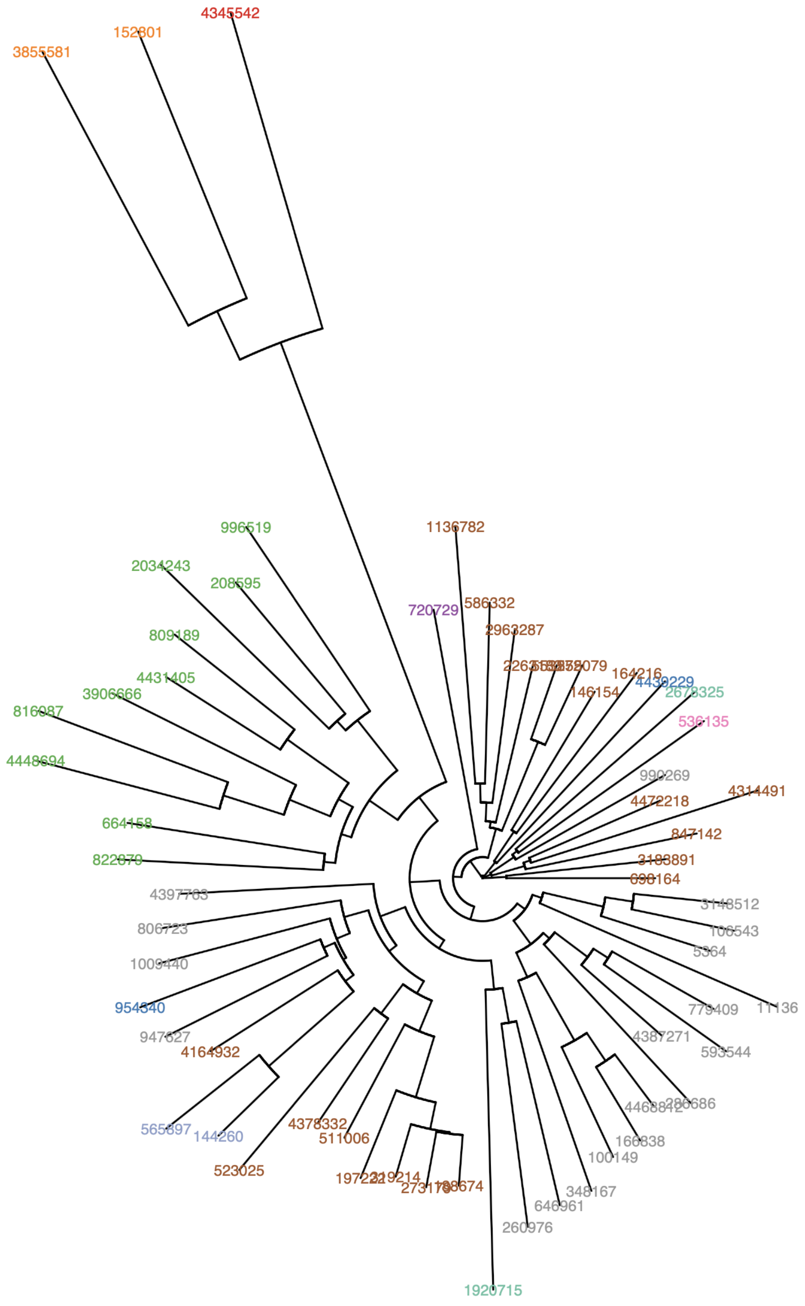
```
import java.util.*;
import java.io.*;

public class GeneticDistance {
    private List<String[]> seqs;
    private List<List<Double>> distances;

    public GeneticDistance(Scanner sc) {
        seqs = new LinkedList<>();
        while (sc.hasNextLine())
            seqs.add(new String[] { sc.nextLine().substring(1), sc.nextLine() });
        computeDistances();
    }

    public GeneticDistance(List<String[]> seqs) {
        this.seqs = seqs;
        computeDistances();
    }

    private void computeDistances() {
        distances = new LinkedList<>();
        for (int i = 0; i < seqs.size(); i++) {
            List<Double> temp = new LinkedList<>();
            for (int j = 0; j < seqs.size(); j++) {
                int dis = 0;
                for (int k = 0; k < seqs.get(i)[1].length(); k++)
                    if (seqs.get(i)[1].charAt(k) != seqs.get(j)[1].charAt(k))
                        dis++;
                double score = (double) dis / seqs.get(i)[1].length();
                temp.add(score);
            }
            distances.add(temp);
        }
    }

    public void printDistances() {
        System.out.println(distances);
    }

    public void writeToFile() {
        try {
            FileWriter w = new FileWriter("genetic-distances.txt");
            w.write("\t\t");
            for (int i = 0; i < seqs.size(); i++)
                w.write(seqs.get(i)[0] + '\t');
            for (int i = 0; i < distances.size(); i++) {
                w.write('\n');
                w.write(seqs.get(i)[0] + '\t');
                for (int j = 0; j < seqs.size(); j++)
                    w.write(String.format("%.5f", distances.get(i).get(j)) + '\t');
            }
            w.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    public List<List<Double>> getDistances() {
        return distances;
    }

    public List<String[]> getSeqs() {
        return seqs;
    }
}
```

2. 3.

```
import java.util.*;
import java.io.*;

public class NeighborJoining {
    public static void main(String[] args) {
        File f = new File(args[0]);
        try {
            GeneticDistance gb = new GeneticDistance(new Scanner(f));
            NeighborJoining nn = new NeighborJoining(gb, gb.getSeqs().size() * 2);
            nn.writeEdgesToFile();
            nn.writeTreeToFile();
        } catch (FileNotFoundException fe) {
            System.out.println(fe.toString());
        }
    }

    public class Pair {
        int node;
        double edge;

        public Pair(int node, double edge) {
            this.node = node;
            this.edge = edge;
        }

        @Override
        public String toString() {
            return ("" + node + "->" + edge);
        }
    }

    private GeneticDistance gb;
    private Map<String, Integer> mapOfIds;
    private Map<Integer, Set<Pair>> map;
    private int root;
    private int seq;

    public NeighborJoining(GeneticDistance gb, int seq) {
        this.gb = gb;
        mapOfIds = new LinkedHashMap<>();
        map = new LinkedHashMap<>();
        for (int i = 0; i < gb.getSeqs().size(); i++)
            mapOfIds.put(gb.getSeqs().get(i)[0], i + 1);
        List<String> headers = new LinkedList<>();
        for (String[] x : gb.getSeqs())
            headers.add(x[0]);
        this.seq = seq;
        nn(headers, gb.getDistances(), seq);
    }
    private void nn(List<String> headers, List<List<Double>> dists, int seq) {
        if (dists.size() <= 2) {
            int key = mapOfIds.get(headers.get(0)) == null ? Integer.parseInt(headers.get(0))
                    : mapOfIds.get(headers.get(0));
            Set<Pair> s = map.getOrDefault(key, new LinkedHashSet<>());
            s.add(new Pair(mapOfIds.get(headers.get(1)) == null ? Integer.parseInt(headers.get(1))
                    : mapOfIds.get(headers.get(1)),
                    dists.get(0).get(1)));
            root = mapOfIds.get(headers.get(0)) == null ? Integer.parseInt(headers.get(0))
                    : mapOfIds.get(headers.get(0));
            return;
        }
        double[] qinfo = getQ(dists);
        double[] edges = edge(dists, (int) qinfo[0], (int) qinfo[1]);
        Set<Pair> s = map.getOrDefault(seq, new LinkedHashSet<>());
        s.add(new Pair(
                mapOfIds.get(headers.get((int) qinfo[0])) == null ? Integer.parseInt(headers.get((int) qinfo[0]))
                        : mapOfIds.get(headers.get((int) qinfo[0])),
                edges[0]));
        s.add(new Pair(
                mapOfIds.get(headers.get((int) qinfo[1])) == null ? Integer.parseInt(headers.get((int) qinfo[1]))
                        : mapOfIds.get(headers.get((int) qinfo[1])),
                edges[1]));
        map.put(seq, s);
        headers.remove((int) qinfo[0]);
        headers.remove((int) qinfo[0] > (int) qinfo[1] ? (int) qinfo[1] : (int) qinfo[1] - 1);
        headers.add(seq + "");
        nn(headers, newDist(dists, (int) qinfo[0], (int) qinfo[1]), --seq);
    }

    private double[] getQ(List<List<Double>> dists) {
        double[][] q = new double[dists.size()][dists.get(0).size()];
        int mini = 0, minj = 0;
        double min = Double.MAX_VALUE;
        for (int i = 0; i < dists.size(); i++) {
            for (int j = 0; j < dists.get(i).size(); j++) {
                if (i != j) {
                    q[i][j] = (dists.size() - 2) * dists.get(i).get(j) - listSum(dists.get(i)) - listSum(dists.get(j));
                    if (q[i][j] < min) {
                        mini = i;
                        minj = j;
                        min = q[i][j];
                    }
                }
            }
        }
        return new double[] { mini, minj };
    }

    private double[] edge(List<List<Double>> dists, int i, int j) {
        double ei = (1 / 2.0 * dists.get(i).get(j))
                + (1 / (2.0 * (dists.size() - 2))) * (listSum(dists.get(i)) - listSum(dists.get(j)));
        return new double[] { ei, dists.get(i).get(j) - ei };
    }

    private double listSum(List<Double> l) {
        double sum = 0;
        for (double x : l)
            sum += x;
        return sum;
    }

    private List<List<Double>> newDist(List<List<Double>> dists, int mini, int minj) {
        double[][] temp = new double[dists.size() + 1][dists.get(0).size() + 1];
        for (int i = 0; i < dists.size(); i++)
            for (int j = 0; j < dists.size(); j++)
                temp[i][j] = dists.get(i).get(j);
        for (int i = 0; i < dists.size(); i++) {
            temp[dists.size()][i] = 0.5 * (dists.get(mini).get(i) + dists.get(minj).get(i) - dists.get(mini).get(minj));
            temp[i][dists.size()] = temp[dists.size()][i];
        }
        double[][] ans = new double[dists.size() - 1][dists.get(0).size() - 1];
        int x = 0, y = 0;
        for (int i = 0; i < dists.size() + 1; i++) {
            if (!(i == mini || i == minj)) {
                y = 0;
                for (int j = 0; j < dists.size() + 1; j++)
                    if (!(j == mini || j == minj))
                        ans[x][y++] = temp[i][j];
                x++;
            }
        }
        List<List<Double>> toRtn = new LinkedList<>();

        for (int i = 0; i < ans.length; i++) {
            List<Double> l = new LinkedList<>();
            for (int j = 0; j < ans[i].length; j++)
                l.add(ans[i][j]);
            toRtn.add(l);
        }
        return toRtn;
    }

    private List<double[]> preorder(int root, List<double[]> l) {
        if (!(map.get(root) == null || map.get(root).isEmpty())) {
            for (Pair p : map.get(root)) {
                l.add(new double[] { root, p.node, p.edge });
                preorder(p.node, l);
            }
        }
        return l;
    }

    public void writeEdgesToFile() {
        try {
            FileWriter w = new FileWriter("edges.txt");
            for (double[] x : preorder(root, new LinkedList<>()))
                w.write(new StringBuilder().append((int) x[0]).append('\t').append((int) x[1]).append('\t').append(x[2])
                        .append('\n').toString());
            w.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    private String postorder(int root) {
        if (map.get(root) == null)
            if (root >= 1 && root <= seq / 2 + 1)
                return gb.getSeqs().get(root - 1)[0];
        List<String> l = new LinkedList<>();
        for (Pair p : map.get(root))
            l.add(new StringBuilder().append(postorder(p.node)).append(':').append(p.edge).toString());
        return new StringBuilder().append('(').append(l.toString().substring(1, l.toString().length() - 1))
                .append(')').toString();

    }

    public void writeTreeToFile() {
        try {
            FileWriter w = new FileWriter("tree.tre");
            w.write(postorder(root).replaceAll("\\s", "") + ';');
            w.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
    public GeneticDistance getGb() {
        return gb;
    }

    public Map<Integer, Set<Pair>> getMap() {
        return map;
    }

    public int getRoot() {
        return root;
    }
}
```

4.


4345542
152801
3855581
996519
2034243
208595
809189
4431405
3906666
816087
4448694
664158
822070
4397703
806723
1009440
954540
947627
4164932
565097
144260
523025
4378332
511006
219214
273179
108674
1920715
260976
646961
348167
100149
160838
4468842
286686
4387271
593544
779409
11136
5364
106543
3148512
698164
3183891
847142
4472218
4314491
990269
536135
2678325
4439229
164216
146154
2963287
586332
720729
1136782



5. Why are tips of similar color mostly clustered together?

The architecture of the allows for congregation between parent and child nodes. And the tips with the same color tend to be closer in genetic distance which leads us to believe that they are from the same species.

6. Name two reasons as to why clustering by color isn't perfect

The first reason is due to accessibility since the full color palette may not be clear to everyone that may need to use it. Secondly, once clusters start increasing in size, the differences in color may become too close to call it regularly.

7.

```java
import java.util.*;
import java.io.*;

public class Bootstrap {
    private NeighborJoining nn;
    private double[] vals;
    List<Integer> ids;

    public Bootstrap(NeighborJoining nn) {
        this.nn = nn;
        ids = dfs(nn.getRoot(), new LinkedList<>(), nn);
        double[] vals = new double[ids.size()];
        Map<Integer, List<Integer>> partitions = partition(nn.getRoot(), nn);
        for (int i = 0; i < 100; i++) {
            List<String[]> bs = new LinkedList<>();
            List<Integer> idxs = new LinkedList<>();
            int N = nn.getGb().getSeqs().get(0)[1].length();
            List<String[]> org = nn.getGb().getSeqs();
            for (int j = 0; j < N; j++)
                idxs.add(new Random().nextInt(N - 1));
            for (String[] seq : org) {
                StringBuilder sb = new StringBuilder();
                for (int idx : idxs)
                    sb.append(seq[1].charAt(idx));
                bs.add(new String[] { seq[0], sb.toString() });
            }

            GeneticDistance bsgb = new GeneticDistance(bs);
            // System.out.println("bsdis: " + bsgb.getDistances() + " \n\n " + "gbdis: " +
            // nn.getGb().getDistances());
            NeighborJoining bsnn = new NeighborJoining(bsgb, 120);
            Map<Integer, List<Integer>> bspartitions = partition(bsnn.getRoot(), bsnn);
            for (int j = 0; j < ids.size(); j++) {
                System.out.println("p: " + partitions.get(ids.get(j)) + " <-> " +
                        bspartitions.get(ids.get(j)));
                if (partitions.containsKey(ids.get(j)) && bspartitions.containsKey(ids.get(j))
                        && partitions.get(ids.get(j)).equals(bspartitions.get(ids.get(j))))
                    vals[j]++;
            }

        }
        for (int i = 0; i < vals.length; i++)
            vals[i] = vals[i] / 100.0;
        this.vals = vals;
    }

    public void writeTreeToFile() {
        try {
            FileWriter w = new FileWriter("bootstrap.txt");
            for (int i = 0; i < vals.length; i++)
                w.write(new StringBuilder().append(ids.get(i)).append('\t').append(vals[i]).append('\n')
                        .toString());
            w.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    private List<Integer> dfs(int root, List<Integer> l, NeighborJoining nn) {
        if (!(nn.getMap().get(root) == null || nn.getMap().get(root).isEmpty())) {
            l.add(root);
            for (NeighborJoining.Pair p : nn.getMap().get(root))
                dfs(p.node, l, nn);
        }
        return l;
    }
    private Map<Integer, List<Integer>> partition(int root, NeighborJoining nn) {
        Queue<Integer> q = new ArrayDeque<>();
        q.add(root);
        Map<Integer, List<Integer>> p = new LinkedHashMap<>();
        while (!q.isEmpty()) {
            int f = q.poll();
            if (nn.getMap().containsKey(f) && !nn.getMap().get(f).isEmpty()) {
                p.put(f, dfs(f, new LinkedList<>(), nn));
                for (NeighborJoining.Pair pair : nn.getMap().get(f))
                    q.add(pair.node);
            }
        }
        return p;
    }
}
```

8. Based on the bootstrap tree pictured above, why is bootstrap support generally higher for the internal nodes closest to the tips, and lower for the internal nodes closest to the root?

In the center of the tree, there are more possibilities for your partition which makes it a higher chance that they don’t equate however, at the tips, there are less partitions which is why have a higher percent.