```
point[] P, point[] Q, point[] R

Path[P.length][Q.length][R.length]
main{
    first check dupes on list P, Q, R, if so, return false
    testLfeasible(0,0,0)
    return Path[n][n][n]
}
testLfeasible(pi,qi,ri, L)
{
    if(pi == n && qi == n && ri == n)
        return;

    if (max(||P[pi + 1] - Q[qi]||, ||R[ri] - Q[qi]||, ||P[pi + 1] - R[ri]||) =< L)
        path[pi + 1,qi,ri] = true
        testLfeasible(pi + 1,qi,ri,L)
    else
        path[pi + 1,qi,ri] = false

    if (max(||P[pi] - Q[qi + 1]||, ||R[ri] - Q[qi + 1]||, ||P[pi] - R[ri]||) =< L)
        path[pi,qi + 1,ri] = true
        testLfeasible(pi,qi + 1,ri,L)
    else
        path[pi,qi + 1,ri] = false

    if (max(||P[pi] - Q[qi]||, ||R[ri + 1] - Q[qi]||, ||P[pi] - R[ri + 1]||) =< L)
        path[pi,qi,ri + 1] = true
        testLfeasible(pi,qi,ri + 1,L)
    else
        path[pi,qi,ri + 1] = false

    if (max(||P[pi + 1] - Q[qi + 1]||, ||R[ri] - Q[qi + 1]||, ||P[pi + 1] - R[ri]||)
=< L)
        path[pi + 1,qi + 1,ri] = true
        testLfeasible(pi + 1,qi + 1,ri,L)
    else
        path[pi + 1,qi + 1,ri] = false

    if (max(||P[pi] - Q[qi + 1]||, ||R[ri + 1] - Q[qi + 1]||, ||P[pi] - R[ri + 1]||)
=< L)
        path[pi,qi + 1,ri + 1] = true
        testLfeasible(pi,qi + 1,ri + 1,L)
    else
        path[pi,qi + 1,ri + 1] = false

    if (max(||P[pi + 1] - Q[qi]||, ||R[ri + 1] - Q[qi]||, ||P[pi + 1] - R[ri + 1]||)
=< L)
```

```
        path[pi + 1,qi,ri + 1] = true
        testLfeasible(pi + 1,qi,ri + 1,L)
    else
        path[pi + 1,qi,ri + 1] = false

    if (max(||P[pi + 1] - Q[qi + 1]||, ||R[ri + 1] - Q[qi + 1]||, ||P[pi + 1] - R[ri
+ 1]||) =< L)
        path[pi + 1,qi + 1,ri + 1] = true
        testLfeasible(pi + 1,qi + 1,ri + 1, L)
    else
        path[pi + 1,qi + 1,ri + 1] = false
}
```

```
double MemoLmin[p index][q index][r index]

point[] P, point[] Q, point[] R
main{
    return minL(n,n,n)
}
minL(pi, qi, ri)
{
    if(pi < 0 || qi < 0 || ri < 0)
        return
    if(pi = 0 && qi == 0 && ri == 0)
        MemoLmin[0][0][0] = max(||P[pi] - Q[qi]||, ||P[pi] - R[ri]||, ||R[ri] -
Q[qi]||)
        return MemoLmin[0][0][0]

    if (pi, qi, ri) is in MemoLmin:
        return MemoLmin[pi][qi][ri]

    MemoLmin[pi][qi][ri] = max(max(||P[pi] - Q[qi]||, ||P[pi] - R[ri]||, ||R[ri] -
Q[qi]||), min(
        minL(pi-1, qi, ri),
        minL(pi, qi-1, ri),
        minL(pi, qi, ri-1),
        minL(pi-1, qi-1, ri),
        minL(pi, qi-1, ri-1),
        minL(pi-1, qi, ri-1),
        minL(pi-1, qi-1, ri-1),
        ))
    return MemoLmin[pi][qi][ri]
}
```

HW 7

Due on **Wednesday, March 13, 2019 at 10am**

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.21**

Unless stated otherwise, for all questions in this homework involving dynamic programming, you need to provide a solution with explicit memoization.

Worked with Jeremy Varghese.

21 (100 PTS.) Exploring Narnia.

Three travelers had decided to travel to Narnia. Since they are all anti-social, they do not want to travel together. Instead, the first traveler would move from location p_1 to location p_2, and so on till arriving to location p_n (here, $P = p_1, \ldots, p_n$). A location is just a point in the plane (i.e., $p_i \in \mathbb{R}^2$). Similarly, the second traveler is going to move along $Q = q_1, \ldots, q_n$, and the third traveler is going to move along $R = r_1, \ldots, r_n$. Every day, a traveler might decide to stay in its current location, or move to the next location (the traveling between two consecutive locations takes less than a day).

By the evening of each day, the travelers arrive to their desired locations for the day, which can be thought of as a configuration $(p, q, r) \in P \times Q \times R$.

A configuration (p, q, r) is *ℓ-legal* if

$$\Delta(p, q, r) = \max\big(\|p - q\|, \|p - r\|, \|q - r\|\big) \leq \ell,$$

where $\|p-q\|$ is the Euclidean distance between the points p and q (this distance can be computed in constant time). (Intuitively, the travelers do not want to be too far from each other, so that if one of them is hurt, the others can quickly come over and help.)

21.A. (50 PTS.) Given the point sequences P, Q, R, and a parameter $\ell > 0$, describe an algorithm, as fast as possible, that decides if there is a motion planning schedule from (p_1, q_1, r_1) to (p_n, q_n, r_n) such that all the configurations used are ℓ-legal. Such a schedule is an *ℓ-feasible schedule*.

Here, it is legal for several travelers to move in the same day, but they are not allowed to move back to a previous location they already used. Furthermore, a traveler can move, in one day, only one location forward in their sequence.

21.B. (50 PTS.) Given P, Q, R, as above, describe an algorithm, as fast as possible, that computes the minimum ℓ for which there is an ℓ-feasible schedule.

21

Solution:

1. **Idea:** The idea of our algorithm is to try each move individually and marking points in a boolean 3D array. If a move is feasible (i.e. the max distance is not more than l) we mark the cell as a feasible move (true). We then recurse along a path that performs that initial move until we come across a false value or reach the point (n,n). Once we find a move that is not feasible we stop recursing along that branch and look at the other options for moves. If any path makes it to point (n,n) and marks it as true we know there is a feasible move schedule for our travelers.

Psuedocode: See image 1.

Analysis: Our algorithm runs in O(n^3) time as it does not backtrack through the array visiting each cell exactly once as a worst case.

2. **Idea:** In our DP approach we will calculate a minimum value for l at each possible state of the points. These values are stored in a 3D array that is memoized. We do a bottom up recursive call of a function that takes coordinates for the current point that minimum l is being computed. With these coordinates the recursive function takes the maximum of the current point's max l (i.e. the max distance between each individual point) and the minimum of all previous l values for all options of previous moves. The idea behind this is that We can memoize these values in our bottom up approach by storing them after each call completes so that the future coordinates that depend on them can utilize the values without recalculation.

   Psuedocode: See image 2.

   Analysis: Our algorithm uses a memoized data structure that ensures we will never recalculate an l value at each point. This is because we follow a bottom up approach in calculating and comparing l values at every possible path. Since each call costs constant time our algorithm is equivalent to traversing the memoized 3D array. This gives us a runtime of $O(n^3)$.
