DP and Graph Algorithms
Problem Set 2 – CS 6515/4540 (Fall 2025)

Answers to problem set 2, question 7.

7 Walking on the Plane

It's easy to observe that for any 2 points, c_1 and c_2 on the Euclidean plane, their dynamically changing circles would touch when $t = \frac{d(c_1,c_2)}{2}$ where $d(x,y)$ is the Euclidean distance between any 2 points x, y. I use a graph based algorithm with modified Dijkstra to find the solution.

1. We construct a set of vertices, V containing all n points on the Euclidean plane and the vertices u, v.
2. We construct set of edges, E by connecting any 2 pairs of points in V (**don't connect u and v unless they're among the n points given**), with weights, W computed by taking half of the pairwise Euclidean distance between points.
3. We then construct a graph $G = (V, E)$ and maintain an adjacency matrix representation. This is an almost fully connected graph, $|E| = \binom{|V|}{2} - 1$ with each edge weight as outlined above.
4. Now we're going to apply our modified Dijkstra's algorithm to find a path from u to v, such that among all paths we choose the path with the minimum maximum weight edge, or the minimum bottleneck. This is because we want to find the earliest time, which is computed from the distance between the vertices or the edge weights.

Modified Dijkstra's algorithm
We set u as our source vertex.

- Maintain a min priority queue containing entries of the form (k, t), this specifies that the min time needed to connect u with k is t. The queue will store elements such that we will always pop the element with min t.
- Maintain a dictionary mapping each vertex to the current know minimum max edge to reach that vertex from source u -> maxEdge

Initialize the priority queue with $(u, 0)$. Now until the pqueue is not empty or we haven't reached the target vertex v, we'll pop element by element from the pqueue, for each popped element = (c, t) we'll iterate over all of it's neighbours and update the time taken to reach the neighbour via the following condition:

$$\text{newMaxEdge} = \max(t, \text{w[c, nbr]})$$

$$(\text{maxEdge[nbr]} > \text{newMaxEdge})? \quad \text{maxEdge[nbr]} = \text{newMaxEdge and pqueue.push((nbr, newMaxEdge))}$$

As soon as we pop the target vertex v from the queue, we return the associated time taken. We can do this on the first occurence of v because our pqueue is a minheap on the time taken to reach a vertex from source.

Algorithm 1 Walking on the plane

```
Input: Euclidean point set $\rightarrow N$, source $\rightarrow u$, target $\rightarrow v$
Output: Earliest connection time $t^*$
1 Build complete graph $G = (V, E)$ with $V = N \cup \{u, v\}$ and edges between all pairs of points except u and v.
  Assign weight $w(i, j) \leftarrow \frac{d(i,j)}{2}$ for all $(i, j) \in E$
  Initialize maxEdge$[x] \leftarrow \infty$ for all $x \in V$
  Set maxEdge$[u] \leftarrow 0$
  Initialize min-priority queue $Q$ and insert $(u, 0)$
2 while !Q.empty() do
3     (c, $t$) $\leftarrow$ $Q$.pop()
      if $c == v$ then
4         return $t$
5     for each neighbor nbr of c do
6         newMaxEdge $\leftarrow$ max(t, $w(c, nbr)$)
          if newMaxEdge < maxEdge[nbr] then
7             maxEdge$[nbr]$ $\leftarrow$ newMaxEdge
              $Q$.push(($nbr$, newMaxEdge))
```

Since, this is just a modifed Dijkstra Algorithm, and the time complexity of Dijkstra is $O(mlogn)$, in this case we get $O(n^2 logn)$ which is $\widetilde{O}(n)$.