As always, sit with a partner and work through these together.

Activity 1: Prim-Jarnik

while PQ not empty
N = removeMin and circle it
connect N to MST (except start node)
for each neighbor M of N
update distance to M if smaller

Activity 2: Runtime of Prim-Jarnik’s
Find the runtime of Prim-Jarnik’s Algorithm based on the following pseudocode by filling in the runtime for each appropriate line of code.

function prim(G):
    for all v in V: 1. O(_______)
        v.cost = ∞
        v.prev = null
    source = a random v in V
    source.cost = 0
    MST = []
    PQ = PriorityQueue(V) 2. O(_______)
    while PQ is not empty: 3. O(_______)
        v = PQ.removeMin() 4. O(_______)
        if v.prev != null:
            MST.append((v, v.prev))
    for all incident edges (v, u) of v: 5. O(_______)
        if u.cost > (v, u).weight:
            u.cost = (v, u).weight
            u.prev = v
            PQ.replaceKey(u, u.cost) 6. O(_______)
    return MST

Runtime of Prim-Jarnik’s: __________________________
Activity 4: Runtime of Naïve Kruskal’s
Fill in the left-hand run times below for Kruskal’s Algorithm based on the naïve union-find implementation of merging clouds.

function kruskal(G):
//Input: undirected, weighted graph G
//Output: list of edges in MST
for vertices v in G: -------------------- 1. O(_______) 1. O(_______)
    makeCloud(v)
MST = []
Sort edges by weight --------------------- 2. O(_______) 2. O(_______)
for all edges (u,v): --------------------- 3. O(_______) 3. O(_______)
    if u and v are not in same cloud:----- 4. O(_______) 4. O(_______)
        add (u,v) to MST
        merge clouds containing u and v ----- 5. O(_______) 5. O(_______)
return MST

Runtime of Naïve Kruskal’s: ____________

Activity 5: Runtime of Path-Compression Kruskal’s
Fill in the right-hand run times below of Kruskal’s Algorithm based on the path-compression union-find implementation of merging clouds.

Activity 3: Kruskal Simulation
for each edge in shortest order
    add edge to MST if it doesn’t make a cycle