(Part 1 a, b)

Pseudocode

Find Every Shortest Path (Node root) { $\boxed{O(V)}$

Step 1: Copy all vertices to local vertices

Step 2: Initialize local vertices, set {vertex i. parent = null, vertex i. mindistance = inf, root. mindistance = 0.}

$\boxed{O(V)}$

Step 3: Build Minheap (local_vertices) {

a) $O(V/2) \rightarrow \boxed{O(V)}$ { Heapify down from vertices.size to 0

} Heapify Down.

$O(\log V)$ { In Heapify Down, compare vertex i to its children and swap if children values < i.

Step 4: While MinHeap Not Empty:

$O(V)$

$O(\log V)$ { Extract_Min : Remove min node, replace with last node in array, heapify Down

$O(E)$ { Update all Neighbors of extracted Node. ~~Heapify~~

$O(\log V)$ Heapify-Up for updated node,

Conclusion! $\underset{\text{step1+2}}{O(V)}$ to initialize. $\underset{\text{step3}}{O(V)}$ to build Heap.

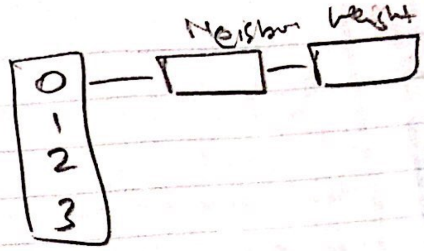
Step4 : ~~F~~ $O(V\log V)$ to Extract Min for V nodes.

Run total of E times { $O(\log V)$ time to ~~upd~~ Heapify Up for each updated neighbor

$(|E| \log (V))$

Ans: $O(V) + O(V) + O(|V+E| \log V) = O(|V+E|\log V)$

1c) Current vertices = Adjacenky List:



New vertices = Adjacency Matrix
↳ Nested for Loop to access Nodes makes it a $O(V^2)$ Call for V nodes.

· With an adjacey Matrix representation, the part where all edges for an extracted node in Step 4 are visited and updated would take $O(V^2)$ time instead of $O(E \log V)$.