a. Give the pseudocode for your implementation of Dijkstra's algorithm and Heap. The pseudocode for Dijkstra's algorithm should involve maintaining the Heap.

findEveryShortestPathLength (Node root){

1. Initialize the graph and root:

For each node v:

Set v.mindistance=∞

Set v.parent=nil

Set root.distance =0

2. Given the input graph, generate a min heap Q
   Call buildheap
3. While (Q.is_not_empty)
   v=extract_min(Q)
   solution.add(v)
   for each edge (v,u)
   if (u.mindistance > v.mindistance+ weight (v,u))
   set u.mindistace= v.mindistance +weight(v,u)
   set u.parent=v
   call heapify_up

return solution

}

findShortestPathLength(Node root, Node x)

{

solution =findEveryShortestPathLength(root)

Return the minimum distance of x in solution

}

findAShortestPath (Node root, Node x)

{

solution =findEveryShortestPathLength(root)

parent=x.getparent()

add both x and parent to shortestpath

while(parent!=root){

    keep getting the parent of current node

    add it to shortestpath

}

reverse shortestpath

return shortestpath

Build heap

{

For i=[n/2] to 0 (n=size of input nodes)

    heapify-Down(i)

}

InsertNode

{

Add node to the last position of heap

Call heapify_up

}

Findmin

{

Return first element of heap

}

Extractmin

{

Store the first element of heap into temp

Use the last element in heap to replace the first element

Delete last element

Call heapify_down(0)

Return temp

}

b. *Justify why your pseudocode runs in O((E + V)(log(V)).*

First for the initialization of graph and root, it takes O(V) times to set their mindistance to infinite and O(V) to set each of their parent to nil. So the total running time in this stage is O(V).

The second step is to build a heap given a set of input nodes, because we only call heapify_down from V/2 to 1, it only takes O(V) time. The proof is given in lecture notes.

The third step is a loop, which will go through every vertex, so the loop will go V times. To extract_min from heap takes O(log V) because we need to remove an element from heap. Then it takes O(e) to check whether we need to update the mindistance, where e is the number of edges for a single vertex. And O(log V) for updating the heap by calling heapify_up for each edge. The total running time at this stage is O(V*logV+E*logV).

Then we sum up all three stages and the total running time of the algorithm is O((E+V)(log(V)).

c. *You implemented Dijkstra's algorithm from a graph represented using an adjacency list. How would your algorithm change if the graph was represented by an adjacency matrix?*

Give and explain the runtime of this approach. For what reasons might it be better to use an adjacency list? Use (5-9 sentences).

The difference happens when we relax on each node. If we use adjacency matrix, we need to go through the entire row of that node to find out which nodes it points to. This will take O(V) for each node. So the running time at stage three becomes O (V*log(V)+V*V*log(V)). Then the total running time becomes $O(V^2 log(V))$. Obviously, the running time of this approach is longer than the case with adjacency list. Therefore, it is better to use an adjacency list.