Week 9 Tute solutions:

Q1)

	Iter-0	Iter-1	Iter-2	Iter-3	Iter-4	iter-5	Iter-6	Iter-7	Iter-8	Iter-9	Iter-10	Iter-11
p	inf	inf	inf	inf	inf	inf	inf	inf	29(s)	28(q)	28(q)	28(q)
q	inf	inf	inf	inf	inf	inf	inf	31(t)	23(s)	23(s)	23(s)	23(s)
r	inf	inf	inf	inf	inf	inf	29(u)	29(u)	24(s)	24(s)	24(s)	24(s)
s	inf	inf	inf	inf	23(v)	23(v)	22(u)	22(u)	22(u)	22(u)	22(u)	22(u)
t	inf	inf	inf	inf	19(v)	19(v)	19(v)	19(v)	19(v)	19(v)	19(v)	19(v)
u	inf	inf	inf	19(x)	18(v)	18(v)	18(v)	18(v)	18(v)	18(v)	18(v)	18(v)
v	inf	inf	18(y)	13(x)	13(x)	13(x)	13(x)	13(x)	13(x)	13(x)	13(x)	13(x)
w	inf	18(z)	17(y)	17(y)	16(v)	16(v)	16(v)	16(v)	16(v)	16(v)	16(v)	16(v)
x	inf	inf	8(y)	8(y)	8(y)	8(y)	8(y)	8(y)	8(y)	8(y)	8(y)	8(y)
y	inf	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)	3(z)
z	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)	0(z)

Q2)

Given a graph G(V,E) with nonnegative weights, in essence, the time complexity of Dijkstra's is O(|V|*T_extract_min + |E|*T_dist_decrease)

The first part of the two part sum above, |V|*T_extract_min, is obvious because we have to iterate over the "Remaining" set of vertices, which is initially of size |V|, and in each iteration a vertex with minimum distance to the source is selected and removed from "Remaining" set. Therefore, each iteration needs to factor in the time T_extract_min required to extract a mindist vertex.

The second part of the two part sum, |E|*T_dist_decrease, comes from the fact that, for each vertex extracted from Remaining, we are traversing its edges and then performing some operations. This gives a complexity of this step that grows as a function involving |E|. While traversing each edge at most once, we update/decrease the distance (where applicable) the distance estimates corresponding to the vertices adjacent in Remaining. Thus, we need to factor in the time T_dist_decrease required to decrease the distance (key) associated with vertices in "Remaining".

Implementing "Remaining" as a LinkedList
Again, the general time-complexity formula of Dijkstra's is
O(|V|*T_extract_min + |E|*T_dist_decrease)
For a linkedlist implementation of Remaining:

● T_extract_min is a O(|V|) operation, as it requires scanning the list to find the vertex with minimum distance estimate so far.
● T_dist_decrease is a O(1) operation, as we are maintaining dist[1...|V|] array where we can simply decrement the distance of immediate neighbours of the extracted vertex x to its immediate neighbour.

Therefore, for a linked list, the time complexity is O(|V|*|V| + |E|) = O(|V|^2+ |E|). Since, in the worst case |E| = |V|^2, the algorithm using a linked list runs in O(|V|^2)time

Implementing “Remaining” as a Priority Queue
Again, the general time-complexity formula of Dijkstra’s is
O(|V|*T_extract_min + |E|*T_dist_decrease)

For a minheap implementation of Remaining:
● Finding the min distance item is simply a O(1) operation on a minHeap, as it is always the top node. However, this vertex has to be extracted/removed from the minheap, which requires a downHeap operation in addition, which is O(log |V|), since the Remaining (maintained as a min-Heap), in the worst case, has size = |V|. Therefore, T_extract_min using a min-Heap is a O(log|V|) operation.
● T_dist_decrease on a min-Heap requires performing an upHeap this node with decreased priority. This operation is O(log|V|).

Therefore, for binary (min)heap implementation of the set “Remaining”, the time complexity is O(|V|*log|V| + |E|*log|V|) = O((|V|+ |E|)*log |V|). This is dominated by O(|E|*log|V|)

As a side note, (not examinable), other variants of priority queues (eg. Brobal Heap or even a Fibonacci Heap) allows T_dist_decrease to be made in O(1)time, making the algorithm run in O(|V|*log|V| + |E|)time.

Q3)
Refer to week 8 lecture slides