1. What is the big-Oh space complexity of Dijkstra's and Prim's? Justify your answer.

Dijkstra: O(V) the set to store vertices distance; another array to store Boolean indicating the vertice is visited or not O(V); therefore O(2V)~O(V)

Prim: O(V) the set to store vertices distance; another array to store parent nodes O(V); another array to store Boolean indicating the vertice is included in MST or not; O(V) to create min-heap; therefore O(4V)~O(V)

2. What is the big-Oh time complexity for Dijkstra's and Prim's? Justify your answer.

it depends on the data structure used to implement

Dijkstra: $O(|E| * T_{decrease-key} + |V| * T_{extract-minimum})$

$O(|V|^2)$ for adjacency matrix representation since extract-minimum is linear search.

for adjacency list representation:

dense graph: $O(V^2 * log|V|)$

sparse graph: $O(|E| + |V|log|V|)$with binary heap which is used as priority queue to extract the minimum distance vertex from the set that not visited and the time complexity of extract-min & decrease-key operation is O(logV) for each edge, but in practice if choose Fibonacci heap whose implementation induces large constant amortized overhead unless it’s a large graph.

Prim:

PQ implementation	insert	delete-min	decrease-key	total
array	1	V	1	V^2
binary heap	logV	logV	logV	ElogV
d-way heap	$log_d V$	$d * log_d V$	$log_d V$	$E * log_{\frac{E}{V}} V$
Fibonacci heap(amortized)	1	logV	1	E+VlogV

Array implementation is suitable for dense graph, requires linear search to find min weight;

Binary heap is suitable for sparse graph;

d-way heap is suitable for problems requires fast performance.

Fibonacci heap is bad in practical situations.

3. Write up the *proof by induction* of the correctness of Dijkstra's algorithm. This should be done similar to the formal proof that we provided for the interval scheduling algorithm. Hint: review Lesson 11.6.

Assumption: for each vertex v, dist[v] is the shortest distance from source to v on the path of visited nodes or infinity if no path exists.

Base case: source node is the first visited node with distance 0

Inductive part: assume the assumption holds for n-1 visited nodes, choose the edge v->u where dist[u]=dist[v]+weight[v->u] is the least among unvisited nodes, then dist[u]would be the shortest distance from source to node u because if shorter path exists and w is the shortest unvisited node then dist[w]>dist[u] which is contradict to the original assumption. Moreover, if shorter path to u among visited nodes exists and last node is w then dist[u]=dist[w]+weight[w->u] which is also a contradiction.

Then after visiting node u, for the unvisited node w, dist[w] is the shortest distance from source to w using visited nodes.

Conclusion: after visiting all nodes, the shortest path from source to any nodes are all visited and dist[v] is the shorted distance.