Simon Anguish

1. $O(n^2)$

For each of the nodes, we will have to iterate through the matrix n^2 times: n times for each row, and n times for each column in that row. For each of these, we will add m_n nodes to the list L, where m_n is a constant of the number of edges for that row. All together, we will have n^2 to check each row, and m_n to add each edge to the list. Since $m_n$is a constant, we can ignore it. Therefore, we have $O(n^2)$.

2. To have a node be strictly greater than $\frac{n}{2}$ nodes away, you must have at least $\frac{n}{2}$ edges between s and t. This means you must have at least $\frac{n}{2}+1$ nodes between s and t. Therefore, you must have $\frac{n}{2}+1+2$ nodes used in the path from s to t: $\frac{n}{2}+1$ for the path itself, and 2 for s and t. Therefore, we only have $n-\left(\frac{n}{2}+3\right)$, or $\frac{n}{2}-3$ nodes free. This shows that, if there were an alternative path from s to t, it would in the best case use those $\frac{n}{2}-3$ nodes and not pass the restriction that the path be greater than $\frac{n}{2}$ nodes away, or it uses one of the nodes from the original path of $\left(\frac{n}{2}+3\right)$ nodes, and therefore, that node v can be deleted to destroy the path.

To create an algorithm for this, we can use the algorithm for a breadth-first search of a graph G with an adjacency list. This provides the runtime analysis of O(m+n), where m is the number of edges in the graph, and n is the number of nodes. We would want to run BFS from either root nodes s or t to build a node tree. We can then remove any paths from the root to the end node where:

- The end node is not the target node, and
- The target node is less than or equal to $\frac{n}{2}$ levels away from the root.

We should then be able to find the level where there is only one node, v, linking s and t.