1. My graph is implemented using an adjacency list. An adjacency list is a data structure that stores each node's neighbor list as an array or vector of linked lists. In this implementation, a vector of vectors is used, where each index of the vector represents a node, and the corresponding vector stores the node's neighbors.
   An adjacency list is a common choice for representing graphs, especially for sparse graphs where the number of edges is much smaller than the total number of possible edges. The main advantage of using an adjacency list is that it allows for efficient access to a node's neighbors, as only the neighbors of a particular node need to be stored.
   An adjacency list is a suitable choice for storing the web graph since it is likely to be sparse, and efficient access to a node's neighbors is required for calculating the PageRank scores.
2. calculate_pagerank(): This function has a nested loop over all the nodes and their incoming links, and runs for num_iterations. The outer loop runs num_iterations - 1 times, so the total number of iterations is num_iterations - 1 times the number of nodes in the graph. The time complexity of the inner loop is proportional to the number of incoming links of each node, which in the worst case is the total number of edges in the graph. Therefore, the worst-case time complexity of this function is O(num_iterations * V * E) = O(V * E) because num_iterations is a constant provided by user input.
   The space complexity of this function is O(V*V + V) = O(V*V) because it creates a 2D vector to store incoming links of each node, and in the worst case each node has incoming links from each other node. Another vector of size V is created to store page ranks.
3. main(): The time complexity of building the graph involves reading and processing each input line. It iterates through each input line, first checking if the node exists in the map with O(log(num_nodes)) = O(logV) and inserting a new node into the map would also be O(log(num_nodes)) = O(logV). Inserting into the vector (adjacency list) is O(1). Therefore, building the graph has time complexity O(num_lines * logV) = O(logV) because the number of input lines is a constant provided by user input.
   The time complexity of calculating the PageRank scores is O(V * E) given by above. The time complexity of creating the pagerank_map is O(num_nodes * log(num_nodes)) = O(V * logV), since it goes through each node in the map of size num_nodes and inserts pageranks in the map. Finally, printing the map goes through each node again with time complexity O(V).
   Therefore, the overall worst-case time complexity of main() is O(V*E + V*logV + V + logV)
   The main creates a map that stores websites and its index of size V, a 2d vector (adjacency list) that stores each node and its outgoing nodes, and in the worst case each node has outgoing links to each other node, a vector of size V to store page ranks and a map of size V to sort the nodes. Therefore, the overall space complexity is O(V*V).
4. If I were to start again I would write a function for creating the graph and not doing so in the main. That would make my code cleaner and more structured, and also easier to test. And I would probably create an adjacency list that stores each website and the websites that link to it, and that would make calculating the pagerank easier. I learned that it is important to think about the problem before starting to code and there might be easier solutions.