unordered_map is chosen for **actorMap** and **movieMap** because it provides fast constant-time average lookup for key-value pairs. Since the goal is to efficiently retrieve information related to actors and movies, unordered maps are suitable for this task.

The program constructs the data structures by parsing the database file. It uses regular expressions to tokenize the input, and for each actor, it adds the movies they've starred in to **movieMap** and the actors for each movie to **actorMap**. This algorithm iterates over each line in the database file, which has a total of "n" lines. For each line, it uses regex token iterator to split the line into tokens based on the delimiter. This process takes constant time O(1) per token. Then, it adds the movie to the **movieMap** and the actor to the **actorMap**, which takes constant time as well. Therefore, the overall time complexity of the code snippet is O(n), where n is the number of lines in the database file.

The search algorithm checks if the queried name (actor or movie) exists in either **actorMap** or **movieMap**. If found, it retrieves the corresponding data efficiently. The search operation in an unordered map is expected to be close to O(1) on average, totaling O(n) due to the iteration. The time complexity of reading each line is O(1) as it is a constant time operation. Inside the loop, there are two if conditions that check if the actor or movie exists in the respective data structures (**actorMap** and **movieMap**). The time complexity of finding an element in a hash map is O(1) on average. Finally, there are nested for loops that iterate over the vector of movies or actors, which has a time complexity of O(m), where m is the number of movies or actors. Overall, the time complexity of the search and retrieval algorithm is O(n * m), where n is the number of lines in the query file and m is the average number of movies or actors per query.

The construction time is proportional to the size of the database file because the program needs to read and process each line in the file to build the data structures. As the database file size increases, the number of lines to process also increases, leading to a linear relationship between the construction time and the database size. This is consistent with the O(N) time complexity for the construction process, where N is the number of lines in the file. The search time, on the other hand, remains relatively consistent, regardless of the database file size. This is because the search operation in an unordered map has an average time complexity close to O(1). The performance of unordered maps is not significantly impacted by the size of the data they contain. If the data can fit comfortably in memory, the search times should be fast and stable.

DB File	# of records	Time taken to create data structure	Time taken to search	Total time taken
dbfile1.txt	442 lines	0.050224 seconds	0.008575 seconds	0.058799 seconds
dbfile2.txt	7065 lines	0.942367 seconds	0.01031 seconds	0.952677 seconds
dbfile3.txt	14129 lines	1.75858 seconds	0.008529 seconds	1.76711 seconds

The efficiency of the chosen data structures and algorithm ensures that the program can handle larger database files without a significant increase in search time. This scalability is a key advantage of using unordered maps (hash maps) for this application.