If I want to implement a dictionary containing the words and their definitions, most probably, the words are already ordered alphabetically to make the search easier for the user. In this case, it is either a sorted linked list or a sorted array.

The user will perform reading and insertion or deletion operations, which means, first of all that the user will perform search operations. Reading doesn't require much more, but searching. Insertion will require push operation in addition.

For searching a word in an ordered array, I can apply binary search algorithm whose complexity is O(logN). It is fast, because it reduces significantly the number of possibilities after every iteration, positioning to the right side of the array. So, it won't examine all possibilities- which is exactly what searching through a linked list will do. It will examine all elements, therefore its complexity is O(N). The time for iterating a linked list is directly proportional with its dimension. In conclusion, search is more efficient with arrays than with linked lists.It has the advantage that is quicker. However, an important factor to consider is the dimension of the dictionary.

For insertion or deletion operation, after performing a search (with O(logN) or O(N)- depends on the case, as I described earlier),I need to push the new element . So, for an array this would mean to move each element one position to the right. (if not inserted at the very end)- so I will visit every element. This means O(N) complexity. But if I have a linked list, after finding the right node and allocating memory, the complexity will be only O(1).
But the question is whether the total is less.
For linked lists: search & push = O(N) & O(1)
For arrays: search & push = O(logN) & O(N)

It seems that for insertion, linked lists have more advantages. But when I imagine this into practice, it is very uncommon to have more insertions than searchings when it comes to dictionaries. Most people usually search for definitions and only a few introduce modifications and this happens very seldom. And deletions happen even more rarely. This is why the disadvantage of linked lists with searching counts so much.

Apart from these two options, I can think of using binary tree data structures for implementing a dictionary. With trees, the complexity of the search operation is, also, logarithmic. This means it can be quicker than with linked lists if it is a balanced tree. The possible advantage is, again, the speed of reading, but we cannot be sure of it since we don't know exactly the dimension and the balance of the tree.