Search History

Rahul is trying to design a custom search engine and wants to prioritize visibility to your temporary search history.

He believes maintaining access to all unique search results of prior searches would potentially speed up retrieval of future searches! Note that users can search the same thing multiple times.

Every time a user clicks the bar, you always want them to view the "num" unique most recent searches!

Help Rahul design a system that will allow you to manage the search history across that one instance, and the according search results, designing in English for the following methods:

```
public class SearchEngine {

   // instantiates the base search engine model
   public SearchEngine() {}

   // return a List of the "num" most recent
   // search keywords, up to total cache size, as Strings
   public List<String> selectSearchBar(int num) {}

   // makes a search on the engine using the keyword
   public void search(String keyword) {}

   // clear cache of previous search results
   public void clear() {}
}
```

1. What ADT(s) best fit the required functionality for this scenario?
   List

2. Describe in 1 to 2 sentences why this ADT is a good fit for this scenario, including any assumptions about how users will behave in this scenario.
   A List ADT will work because you can access intermediate indices to search unique searches. Additionally, order of the searches is important here.

3. What Data Structure implementation(s) of your chosen ADT is optimal for this scenario?
   Linked List

4. Implementation and Optimization

   a. Describe how your chosen data structure would be used to implement each of the functions of the program. (A few sentences per function).

      - For SearchEngine(), we can instantiate a linked list of nodes that contain a data field for the keyword, and results map.
      - For selectSearchBar(num), we traverse through the list until we either hit num nodes or reach the end. We return those "num" search keywords as a List of strings.
      - For search(), we traverse the list until we either reach the end or find an existing node with the same keyword. If such a node exists, we move it to the front of the list. Otherwise, we make a new node and add it to the front.
      - For clear(), we set front pointer to null.

   b. Describe which functions of the program are optimized by your selected data structure, e.g. What is the most efficient task your ADT accomplishes?
      - This approach optimizes selectSearchBar(), as we guarantee that all nodes in the list are unique. This means that there will be no duplicated memory and that the method only requires a single traversal of our list of searches.

   c. Describe which functions (if any) of the program are not optimized by your selected data structure, e.g. What tasks are not fully efficient with your ADT?
      - This data structure does not optimize the search() function, as we might have to traverse the entire linked list of searches. However, this is more optimized than other designs because, for example, moving a node to the front of the list simply requires reassigning pointers rather than shifting elements in an array.

5. Describe the best-case scenario for your design, including discussion of how this impacts each of the programs's functions.

   For this design, selectSearchBar() has no best or worst case, as grows with "num."
   Best case scenario for search() is when the current search keyword is the same as the previous search keyword, meaning it is already at the very front of the list.

6. What is the Tight Big O of the best-case scenario for each of the functions you described in part 5?

   In the event that the current search is the same as the previous one, the runtime of search() is O(1). As said previously, selectSearchBar() lacks a best case scenario.

7. Describe the worst-case scenario for your design, including discussion of how this impacts each of the program's functions.

   The worst-case scenario for seach() iis when the search keyword has never been searched before. This means that we will have to iterate through the whole linked list.

8. What is the Tight Big O of the worst-case scenario for each of the functions you described in part 5?

   As said prior, selectSearchBar has no best or worst case. For search() the runtime is O(n). The constructor and clear() method remain constant.