**Challenge #2**

Imagine your company has created a service which can translate small snippets of text (say, up to 140 characters long) from one language to another (e.g., from English to Danish). The translation is computationally expensive, and your backend systems are having difficulty keeping up with the growing popularity of the service.

To reduce load on the backend systems, you are tasked with designing a cache to store commonly-translated snippets. Assume it is not an option to use existing caching systems.

Describe how your caching system would handle the following operations (for example, what algorithms/data structures would you use, and why):

* a) Locate the translation of a given snippet, if it is present in the cache
* b) Determine which snippet should be removed from the cache to make room for new translations. This should be the snippet which has been in the cache for the longest time without having had a 'hit'

NOTE: I am not a linguistics expert so talking to one would definitely help.

Answer question a:

I would break the snippet into sentences and assume these sentences can be translated without having a correlation with the previous or next sentence in the snippet. Therefore increasing the chance of finding previously translated pieces of text.

These sentences (or snippets) would be hashed to increase finding speed.

Answer question b:

Keeping a DateTime for the last hit will result in a value to remove snippets based on time. The object would consist of a snippet or sentence (in both language), and a DateTime (last hit).

Lists:

1. List with objects
2. Indexed list Hashes -> point to one or more objects (List<KeyValuePair<string, List<object>>>) where string is the hash

Steps:

* Get object(s) for hash
* String check to avoid hash collision
  + Add new if not existing
* Update DateTime and return string

Separate process that removes objects based on DateTime

Functions to get the correct objects would be:

ConvertTextToHash = to enable hash search   
GetDateTimeLastHitForHash = to know when the last hit was

There is a chance that the same hash can occur more than once for different text. Hash collisions can occur as the input entropy of the hash function is bigger than the output entropy. Because of this a string compare is done afterwards.