2. KMP and Performance Evaluation (40 points)

   To generate my test strings for comparing KMP, Python's `find()`, and the naive string searching algorithm, I chose to rely mostly on randomness. The function I wrote to generate a haystack and needle is `genHaystackNeedle()` in my source file `kmp.py`. I start out by defining the final length of the haystack, which will be a close approximation to what the actual length ends up being due to the randomness. I also define the needle length and as well as a "max jump length" which means the maximum number of random characters we can insert in the haystack with each iteration of the loop.

   I then generate a needle by randomly choosing lowercase ASCII characters. Looping while i is less than the final length, we first add a random number (less than max jump length) of random characters to the haystack, and then we might add some of the needle, the full needle, or nothing.

   This implementation results in different performance every time, but a typical run may be:

```
$ python3 kmp.py
found at: 156913
101
found at: 156913
0
found at: 156913
65
```

   The needle was found at position 156,913 in a haystack of at least length 10,000,000. No matter what strings I generate, the built-in `find()` operation seems to always perform better. I think this is a feature of Python as it uses an experimental string searching algorithm that must perform better than KMP.