1. I see the timings for Hanoi increases dramatically as I increase the n. When n=3 it runs approximately in 0.0022s; when n=6 it takes 0.017s; but when n=12, it takes 1.6s. I think it's not a performance class we have seen before, since it's beyond the quadratic performance. I was thinking that probably the cause is the three lines of print() inside the Hanoi_rec(), which occupies the IO and takes a lot of time. And when n increases, the number of steps taken by the Hanoi increases a lot, so the print line is executed way more, which increases the time a lot. After I commented out the three print lines and test again, the timings decrease a lot, which proves my guess.
2. To employ Python programs that accept arguments/parameters when they are invoked on the command line, we need to import the *sys* class and uses sys.args, which is what we write on the command line. Sys.args is the complete line, while sys.args[0], for example, is the first segment of string (separated by spaces) in the line. To employ this together with factory method infrastructure provided by Deque_Generator, we can import sys in DSQ_Test.py, and then replace the self.__deque = get_deque(1) by self.deque=get_deque(sys.args[2]). In the main section, we need to allow the user to write 0 or 1 after "python DSQ_Test.py". Then, we can specify which kinds of deque we want to test from the command line.
3. When I test Hanoi, I think of the positions of s, a, d as the parameter in each line of Hanoi_rec() and adjust them. Then I consider the positions of the print lines to see when do I need to print the Hanoi to see the progress. And then I'm done.
4. When I test Delimiter_Check, I first think of the special case as professor mentioned in class, which is when the delimiter is in a quotation. So I use several if/else to decide whether the current position of the iteration is inside a quotation. Then, I write codes to see if the iterator reaches a "{", "[", or "(". If so, push it into the stack. Else if the iterator reaches a ")", "]" or "}", check if the top of the stack matches the delimiter. If not, the file fails the test. If yes, continue until the iterator reaches the end of the file. Then I write a test file to see if the Delimiter_Check works. Done.