Question 1:

1. O(n^6)
2. O(n^4)
3. O(n^2)
4. O(2^n)
5. O(sqrt(n))

Question 2:

1. def my_function(n):	
2. if n < 1:	2. Will not run since the condition is false
3. return	
4. while n > 1:	4. While will run until n reaches 1.
5. if n % 2 != 0:	5. Will not run since the given number is even.
6. n = 3 * n + 1	
7. else:	
8. n = n / 2	8. Will keep running until n reaches 1
9. return n	9. Will return n once it reaches 1
	There is one operation happening to n in this function when n > 1 and n is a power of 2 which is n/2, this implies that for every doubling of length n there is 1 additional operation which makes this function have the time complexity O(log(n))

Question 3:

The problem with the case when n is not a power of 2 and greater than 1 is when n is an odd number the first condition triggers which multiplies that odd number with 3 then adds 1 resulting in an even number which is then divided into 2 in the next iteration which causes the first condition to trigger again in the next iteration of the while loop, since the while loop never breaks the function will have unlimited repetitions, thus not have a time complexity.