Part B Explanation:

In my code, in order to query for DNSSec, you simply write into the terminal "./mydig verisigninc.com A +dnssec". I found that records with and without "www." have the same terminal server (found on: https://dnsviz.net/d/kirei.se/dnssec/?rr=2&a=all&ds=all&ta=.&tk=), but often returns a different one based on load bearing, therefore, if an input website has "www.", I remove it. Additionally, before DNSSec verification begins, I query my function from part a (`iterateQuery(website, ipAddress, queryType, startTime, dateTime, msgSize, rootServerFound, orgWebsite, DNSSec=False)` with DNSSec set = True) and see if the query will return an answer. If it returns an answer, I proceed with DNSSec verification, otherwise, I output "Sorry, there is no solution to your query."

To verify DNSSec, I start by verifying the root (my function: `rootServerDNNSECValidate(website, queryType)`). I do this by obtaining a trust anchor, from http://data.iana.org/root-anchors/root-anchors.xml (webpages that confer with this information: https://github.com/iana-org/get-trust-anchor and https://www.afrinic.net/blog/265-dnssec-new-root-zone-ksk-appears-on-the-dns ). After such, I query the root for DNSKEY records and query type records ("A","MX" or "NS") with the want_dnssec flag set equal to true (ie. `answer = dns.message.make_query(website, queryType, want_dnssec=True), answer2 = dns.message.make_query(".", "DNSKEY", want_dnssec=True)`). Doing such returns to me, the next Ip address to query, the DS record, its corresponding RRSIG, the DNSKEY record, and its corresponding RRSIG. Respectively, I validate the records with their RRSIG values. To validate the DS record requires the DNSKEY, so validation must occur first with the DNSKEY record, and then with the DS record. After such, I obtained the Public Key Signing Key (Pub KSK) (present in the answer where the DNSKEY records were found). This Pub KSK is a DNSKEY record marked with a flag number of 257. Once found, I then hash the record (done with either SHA256 or SHA1 as the algorithm) and compare the value with my trust anchor. If any of these verifications fail, I know that my dns record has been corrupted.

After obtaining the next IP address, I put it through an iterative function (called: `iterateDNNSecQuery(website, websitePart, queryType, ipAddr, dicValues, orgWebsite, rootServerFound, flag, rootDicValues))` that will continue to check for validation as well as continue the DNS procedure to resolve the IP address. It will make similar queries as that which was done for the root. However, I will not be comparing the trust anchor, but rather, verifying the zone. In order to do this, I will hash my Pub KSK (done with either SHA256 or SHA1 as the algorithm) and compare this value with the DS record of the parent. If the values match, then I know that the zone is correct.

This methodology will typically result in an answer. However, when ".org" is queried, I have to do additional steps and therefore I have created a separate function in order to handle this: `dnssecOrg(website, queryType, ipAddr, dicValues, rootIP, rootDicValues)` (which gets called upon in iterateDNNSecQuery). It makes three requests: `answer = dns.message.make_query(website, queryType, want_dnssec=True), answer2 = dns.message.make_query("org.","RRSIG", want_dnssec=False), answer3 = dns.message.make_query("org.","DNSKEY", want_dnssec=False).` The RRSIG and DNSKEY records are requested separately, but

used in a similar manner as previously for validation. Typically, the query will result in a NS record, which will be put into my original iterative function from part A. The IP will then be passed back to the original iterateDNNSecQuery function.