The file is parsed using bs4 and “xml” parser. “Lxml” was not working for me as it formatted the <HEAD> tag and so the content after parsing using bs did not contain any <HEAD> tags. So “xml” parser has been used. Since “xml” only parses the topmost tag and its containing children, I had to make a temporary tag at the start at and at end of the file so that xml parses everything inside it, i.e., all the documents in the file, not just the topmost one.

All the DOCNO tags are found and stored in an array. Now, for each document we iterate over the tags given in xml-tags-info file. The file is read from the second line, as the first one always contains DOCNO. A document counter is maintained which increments by 1 with each document. For each tag, content is found inside the tag and is then split using delimiters. Now, these term list obtained after splitting goes through the stopword check and is proceeded only if the term does not lie in the stopword list. After passing the stopword check, the term goes through the stemming process using porter stemmer. Finally, the document has to be added to the postings list of the stemmed term. A dictionary(Inverted index) is maintained and the doc id is added to the key-value pair whose key is the stemmed term. Another counter is maintained that checks how many document ids have been added to the inverted index and when it reaches a threshold, a postings list file is made, the dictionary is updated and the inverted index is emptied. A document id-name map is also made which records which integer is assigned to which document. From the sorted inverted index, each doc id is converted to binary and written to the postings list file and the dictionary is updated with the startbyte, bytelength, and the name of the postings list file for the corresponding term.

After all the smaller postings lists are made and all the files are traversed, the postings lists are merged using the dictionary which contains names of smaller postings files, start byte and content length. Using these, a final postings list file is constructed for the terms in sorted order.

Compressions:
This final postings list file is then compressed if it is given that any compression should be applied.
Since we know that this file has each integer as a 32-bit binary(4 bytes), this file is read byte by byte and the relevant compression is applied.
For c1, 4 bytes are read, their binary string is stored in a string and then it is grouped in 7 bits and a new binary string is generated which is then converted to the hex form using to_bytes. Whereas, for c2, each byte is read and is converted to its hex form and written to the file.
For c3, the final postings list file, without any compression, is given as an argument to snappy and the compressed file is generated.

Query retrieval:
The query file is parsed line by line, the terms in the query are separated, checked using the stopwords list(which has been added to the file at the time of indexing), stemmed and the final list of terms to search for is created.
For each term, using the dictionary, the postings list is retrieved. If any compression has been performed then the bytes read are decompressed accordingly. After the document list for each

term is retrieved separately, its intersection is taken and the final list of document ids is then mapped to their names using the doc id-name map.

METRICS:

ISR - without compression: 0.46
ISR - C1 compression: 0.28
ISR - C2 compression: 0.30
ISR - C3 compression: 0.34

Average time taken by one query = 2505 microseconds.