1. The directory structure of your code

- Root folder contains following
    - **Indexer.py**
        - takes the xml dump as command line argument and builds index
    - **Search.py**
        - Reads queries from **query.txt** and outputs the results to **query_output.txt**
    - **Invertedindex_stat.txt**
        - Contains 2 lines
        - 1st line corresponds to number of docs parsed
        - 2nd line corresponds to size of vocabulary i.e., number of token encountered in the dump
    - **pageIndex**
        - Folder which contains text files and each file stores the inverted index of 5000 files each
    - **globalIndex**
        - After parsing the whole xml dump, we should have now divided the global inverted index into pageIndex
        - We now need to perform K-way merging of files to build the global inverted index
        - Each file in this folder contains 20000 tokens and its corresponding posting list
    - **secondaryIndex.txt**
        - Contains the starting token of each file in globalIndex
        - This is used while searching
    - **titles.txt**
        - Contains titles of each document
        - This is used while searching so that we can return the title of the ranked document
    - **titleIndex.txt**
        - Contains the logical offsets of titles in titles.txt so that when searching, we can directly seek to the location and read the title instead of loading the whole titles.txt file

2. Optimizations Used

- Splitted the global index into multiple files in sorted order so that it can be efficient during searching
- For this, after parsing 5000 documents, I am storing the inverted index into a file and after all document are parsed, used k-way merge of these files to create global inverted index

- Created index for titles so that we can return them using seek instead of loading all the titles into main memory

3. Improvements in terms of time and space from those optimizations

- **Space optimization**
  - Used compressed representation of posting list, for example if a word occurred 6 times in title and 5 times in categories, instead of storing 6,0,0,5,0,0 i used t6b5
- **Time Optimization**
  - While searching, in order to maintain top k results, I maintained a min-heap of size k and everytime a document comes, if it is greater than minimum element in heap, i am removing the top element in heap and pushing this document so that we can maintain top k documents

4. Index creation time and size

- 40000 seconds ~ 11 hours for index creation
- 30gb index size

5. The format of the final index created -> what are the keys and values etc

- Index spans across multiple files and the format is
- Token + ; + docfreq + ; + [docid : + t*i*b*c*|*r*]*
- Example
  - baiji;6;52474:b1;122040:b3;164458:b1;348059:b2;360796:i1;432109:b1;