Exercise 1a

Skiplists are basically a data structure that can help us optimize the search during the query time by skipping some parts of the ordered lists that we know for sure to not be a part of the final search result. This technique is particularly useful when we have to merge/intersect 2 postings lists. We divide each postings list into several sections of the same size. Each section has a skip pointer that points to the next section of the list. During the intersection, we compare the current posting of the 1st list with the skip pointer of the 2nd list. If the skip pointer is less that the current posting, then we know that the section corresponding to the skip pointer will never be in the results set, which means that we can skip this entire section and move on the next section in the 2nd list.

Skiplists are not always beneficial for performance. This might be due to some of these following reasons:

- We might be unlucky and get 2 postings lists that are very similar in content. In this case, the intersection of the 2 lists will contain almost all the documents in the lists, making the skiplists useless (because there is nothing to skip).
- We may have placed too few or too many skip pointers within a single postings lists.
  - If there are too few skip pointers (i.e. long skip spans), then we will have very few successful skips on average, which means that the skip lists are not useful at all.
  - If there are too many skip pointers (i.e. short skip spans), then on average, it's more likely for the skip to be successful. However, the downside is that we will need to make a lot of skip comparisons.
- Since skiplists are often only available for the original postings lists in the inverted index, they may not be useful for a simple Boolean retrieval model that processes a compounded Boolean query. For example, let's say that we want to process the Boolean query "(*quantum* AND *cryptography*) AND *computing*". Skiplists may be useful for the merge between *quantum* and *cryptography*, but since this intermediate postings list doesn't have any skiplists, so the skiplist of the *computing* list will not be beneficial at all.

Exercise 1b

In vector space model, documents and queries are represented by vectors of the same number of dimensions (say n). Each dimension represents a unique term t_i in the global dictionary. To perform the ranking of the documents, we compute $\text{cosine}(q, d)$, which is the cosine score between the query q and the document d. But in order to do this, we must figure out the inner product between the column vectors $\mathbf{q} = [q_{t1} \quad \cdots \quad q_{tn}]^T$ and $\mathbf{d} = [d_{t1} \quad \ldots \quad d_{tn}]^T$, where each q_t (and d_t) is the weight that corresponds to the dictionary term t. The inner product is thus equal to:

$$\mathbf{q} \cdot \mathbf{d} = q_{t1}d_{t1} + \cdots + q_{tn}d_{tn}$$

In other words, the inner product is equal to the sum of the contributions from each of the dictionary term. Of course, if a dictionary term t doesn't appear in the query or in the document, then that term doesn't contribute anything to the product, which means that we only need to look at the contributions from the *query* terms during the computation.

To compute the full ranking of all the documents in the collection, we must compute the inner product between the query and each of the documents in the collection. Document-at-a-time evaluation (DAAT) basically computes the cosine score for one document at a time (i.e. it computes $\mathbf{q} \cdot \mathbf{d_1}$ for the document d_1 before moving on to $\mathbf{q} \cdot \mathbf{d_2}$ for document d_2). On the other hand, term-at-a-time evaluation (TAAT) computes the contribution from one query term at a time for all the documents in the collection, before moving on to the next query term.

If the postings in the postings lists of the query terms are all arranged in one common ordering (e.g. by docID, or by some static quality scores), then we can use DAAT. This is because we can then concurrently traverse through all of these lists (similar to when we perform the merging of the lists), picking out the documents that appear in all lists, and compute the cosine scores for all of these documents, but one at a time.

On the other hand, if we use an ordering that is not common in all of the postings lists for the query terms, then such concurrent traversal is impossible, which means that we can't pick out the documents that contain all of the query terms. In this case TAAT is useful, because we can then go through each postings list one at a time, and accumulate the score for each document by adding the contribution from the query term represented by this postings list.

In impact-ordering, the impact score is usually the tf.idf rating, which depends not only on the document, but also on the query term as well (i.e. the impact score is for a pair of document and query term). This means that the impact score for a particular document will differ across the postings lists. Because of this, TAAT must be used (as already explained in the previous paragraph).

Exercise 1c

Static quality score is the measure of the *authoritativeness* for a document. This is a query-independent static property of a document, which can be based on PageRank, website reputation, journal reputations, etc. It's beneficial because a high static score usually indicates that the website is popular among people and that it is a trustworthy source of information. There are plenty of examples of this in Google. Say that you search for the name of a film, then Wikipedia (or perhaps IMDB) is usually the first result that comes up, rather than a blog of some random critic discussing about the film. This is because Wikipedia/IMDB have higher static score (i.e. more popular and trustworthy) compared to some random blogs. Another example is video search. YouTube is likely to have higher static score compared to other video hosting websites, which means that YouTube will be ranked higher, even though other websites might have the exact same video content as YouTube.