Information Retrieval Tutorial - Set 1

1. A. Draw the term-document incidence matrix and the inverted index representation for the following document collection:
    - Doc 1 : breakthrough drug for schizophrenia
    - Doc 2 : new schizophrenia drug
    - Doc 3 : new approach for treatment of schizophrenia
    - Doc 4 : new hopes for schizophrenia patients

Term-Doc Incidence Matrix

term\doc	Doc1	Doc2	Doc3	Doc4
breakthrough	1	0	0	0
drug	1	1	0	0
for	1	0	1	1
schizophrenia	1	1	1	1
new	0	1	1	1
approach	0	0	1	0
treatment	0	0	1	0
of	0	0	1	0
hopes	0	0	0	1
patients	0	0	0	1

Inverted Index

- breakthrough → Doc1
- drug → Doc1 → Doc2
- schizophrenia → Doc1 → Doc2 → Doc3 → Doc4
- approach → Doc3
- treatment → Doc3
- hopes → Doc4

- patients → Doc4

B. What are the returned results for these queries-
- schizophrenia AND drug
- for AND NOT(drug OR approach)

schizophrenia →	1 1 1 1
drug →	1 1 0 0
schizophrenia AND drug →	1 1 0 0

Returned: Doc1, Doc2

drug →	1 1 0 0
approach →	0 0 1 0
drug OR approach →	1 1 1 0

for	1 0 1 1
NOT (drug OR approach) →	0 0 0 1
for AND NOT (drug OR approach) →	0 0 0 1

Returned: Doc4

2. What is its time complexity of the postings merge algorithm to arbitrary Boolean query formulas? For instance, consider: (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

x OR y: $O(|x| + |y|)$
x AND y: $O(|x| + |y|)$

x AND NOT y: $O(|x| + |y|)$ [You can leave out all docs from x that are also present in y, thus just scanning both lists will do]
x OR NOT y: $O(N)$ [You need to go over all docs in the corpus]

3. Write out a postings merge algorithm that evaluates this query efficiently-

x AND NOT y

```
INTERSECT_AND_NOT(p1, p2)
    answer ← < >
    while p1 ≠ NIL and p2 ≠ NIL
        do if docID(p1) = docID(p2)
            then p1 ← next(p1)
            p2 ← next(p2)
        else if docID(p1) < docID(p2)
            then ADD(answer, docID(p1))
            p1 ← next(p1)
        else
            p2 ← next(p2)
    while p1 ≠ NIL
        do ADD(answer, docID(p1))
        p1 ← next(p1)
```

4. We have a two word query. For one term the postings list consist of the following 16 entries.

   [2, 4, 9, 12, 14, 16, 18, 20, 24, 32, 47, 81, 120, 125, 158, 180]

   and for the other list it is the one entry postings list
   [81]

   Work out how many comparisons would be done to intersect the two postings list with the following two strategies.
   i. Using standard postings list.
   ii. Using postings list stored with skip pointers, with the suggested skip length of √P (P=length of the list).

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

i: 12 comparisons

ii: 7 comparisons – (2,81), (14,81), (24,81), (120,81), (32,81), (47,81), (81, 81)

Consider the following fragment of a positional index with the format:

word: document: <position, position, . . .>; document: <position>,...

Gates: 1:<3>; 2:<6>; 3:<2,17>; 4:<1>;
IBM: 4:<3>; 7:<14>;
Microsoft: 1: <1>; 2:<1,21>; 3:<3>; 5:<16,22,51>;

The /k operator, word1 /k word2 finds occurrences of word1 within k words of word2 (on either side), where k is a positive integer argument. Thus k=1 demands that word1 be adjacent to word2.

Describe the set of documents that satisfy the query -> Gates /k Microsoft for k=1, 2

Gates → Doc1 → Doc2 → Doc3 → Doc4
Microsoft → Doc1 → Doc2 → Doc3 → Doc5

Matches for Doc1, Doc2, Doc3

<u>Gates /1 Microsoft</u>
Cross-product of positions
Doc1: (3,1)

Doc2: (6,1), (6,21)
Doc3: (2,3), (17,3)

Gates /2 Microsoft
Cross-product of positions
Doc1: (3,1)
Doc2: (6,1), (6,21)
Doc3: (2,3), (17,3)

5. If |S| denotes the length of string S, show that the edit distance between s1 and s2 is never more than max{|s1|, |s2|}.

Assume |s1| <= |s2|. Also assume the worst case scenario — all characters of s1 and s2 are different.
So, we need to go from s1 to s2 using standard operations (add/delete/replace).
- First, replace all characters of s1 with all characters of s2 upto position |s1| — |s1| operations
- Then, keep adding the rest of the characters of s2 — |s2| - |s1| operations
- Total no. of operations = |s2| = max(|s1|,|s2|)

6. Compute the edit distance between paris and alice. Write down the 5 × 5 array of distances between all prefixes.

	є	a	l	i	c	e
є	0	1	2	3	4	5
p	1	1	2	3	4	5
a	2	1	2	3	4	5
r	3	2	2	3	4	5
i	4	3	3	2	3	4

s	5	4	4	3	3	4