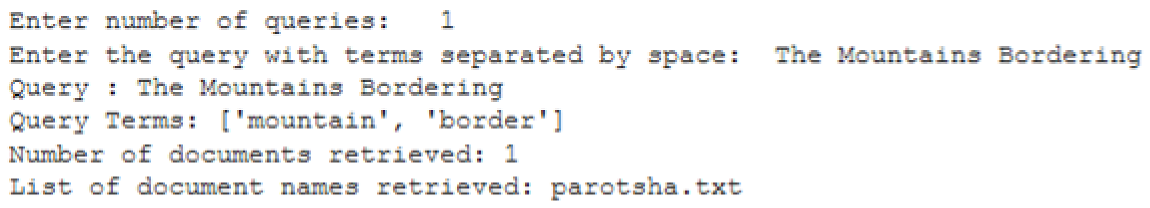
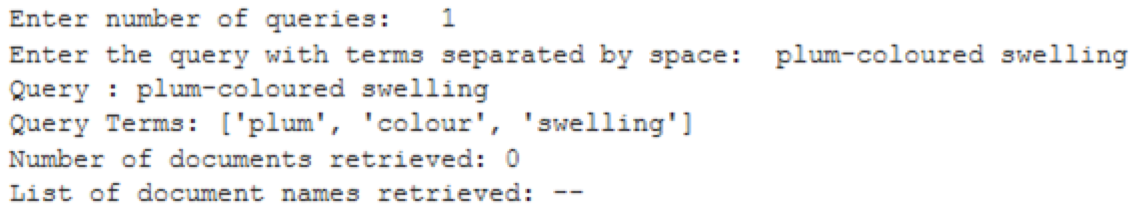
Question 1 :
Sample outputs:
Given the query - “The Mountains Bordering”



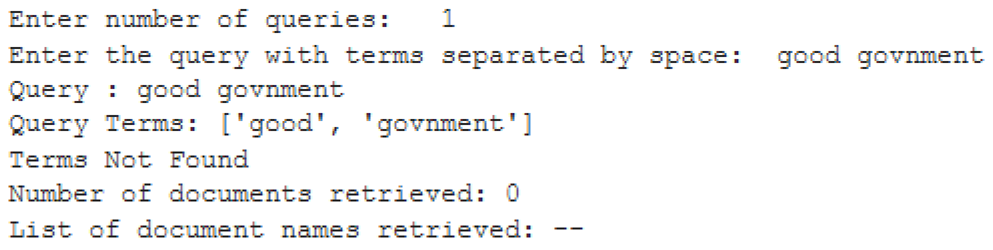
```
Enter number of queries:   1
Enter the query with terms separated by space:  The Mountains Bordering
Query : The Mountains Bordering
Query Terms: ['mountain', 'border']
Number of documents retrieved: 1
List of document names retrieved: parotsha.txt
```

Given the query - “plum-coloured swelling” -- Query does not match



```
Enter number of queries:   1
Enter the query with terms separated by space:  plum-coloured swelling
Query : plum-coloured swelling
Query Terms: ['plum', 'colour', 'swelling']
Number of documents retrieved: 0
List of document names retrieved: --
```

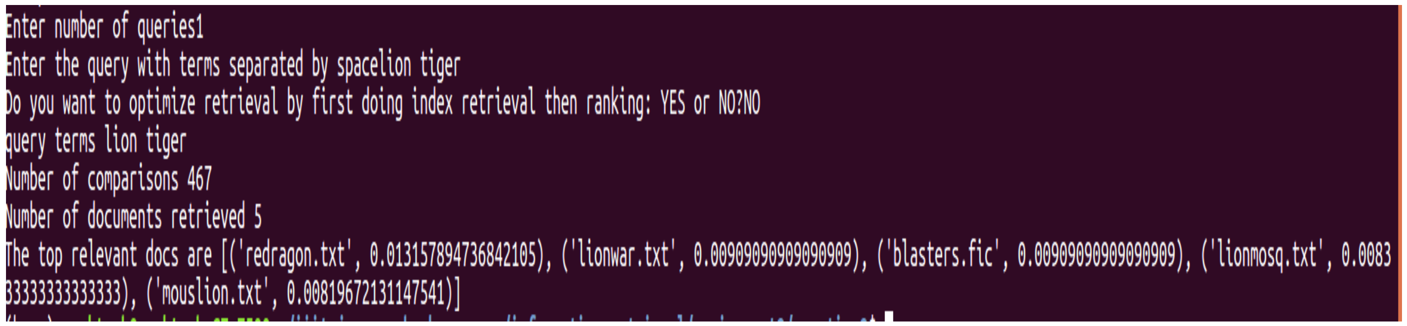
Given the query - “good govnment”   -- Query terms not in dictionary



```
Enter number of queries:   1
Enter the query with terms separated by space:  good govnment
Query : good govnment
Query Terms: ['good', 'govnment']
Terms Not Found
Number of documents retrieved: 0
List of document names retrieved: --
```

Question 2 :
Jaccard similarity output:

Without optimization (compared with all documents):



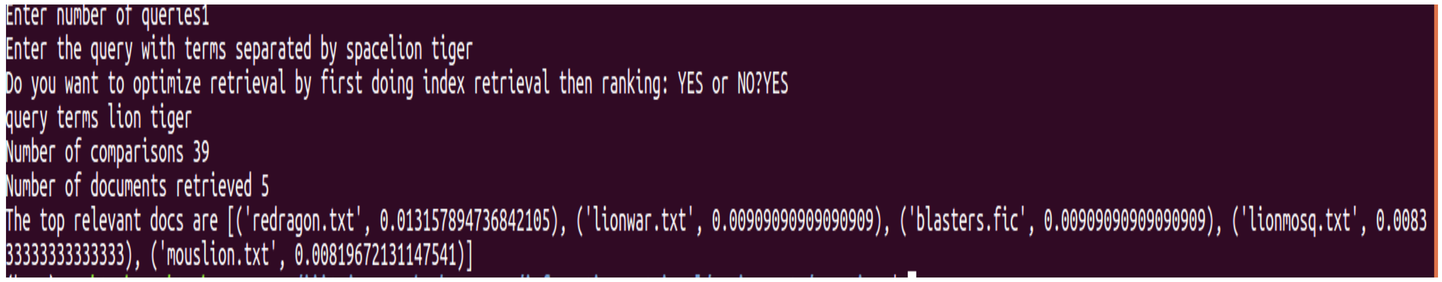
```
Enter number of queries1
Enter the query with terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?NO
query terms lion tiger
Number of comparisons 467
Number of documents retrieved 5
The top relevant docs are [('redragon.txt', 0.013157894736842105), ('lionwar.txt', 0.00909090909090909), ('blasters.fic', 0.00909090909090909), ('lionmosq.txt', 0.0083
33333333333333), ('mouslion.txt', 0.00819672131147541)]
```

Output (text form):

Enter number of queries **1**
Enter the query with terms separated by space **lion tiger**
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?
NO
query terms **lion tiger**

Number of comparisons **467**
Number of documents retrieved **5**
The top relevant docs are **[('redragon.txt', 0.013157894736842105), ('lionwar.txt', 0.00909090909090909), ('blasters.fic', 0.00909090909090909), ('lionmosq.txt', 0.008333333333333333), ('mouslion.txt', 0.00819672131147541)]**

With optimization : Depending on the choice made by user we optimize the computation of Jaccard similarity between the query and the document. We use the inverted index to filter out documents and retain only the documents that contain atleast one of the terms in the query. This helps reduce the number of Jaccard similarity computation as it has to be computed for every query-document pair.



```
Enter number of queries1
Enter the query with terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
query terms lion tiger
Number of comparisons 39
Number of documents retrieved 5
The top relevant docs are [('redragon.txt', 0.013157894736842105), ('lionwar.txt', 0.00909090909090909), ('blasters.fic', 0.00909090909090909), ('lionmosq.txt', 0.0083
33333333333333), ('mouslion.txt', 0.00819672131147541)]
```

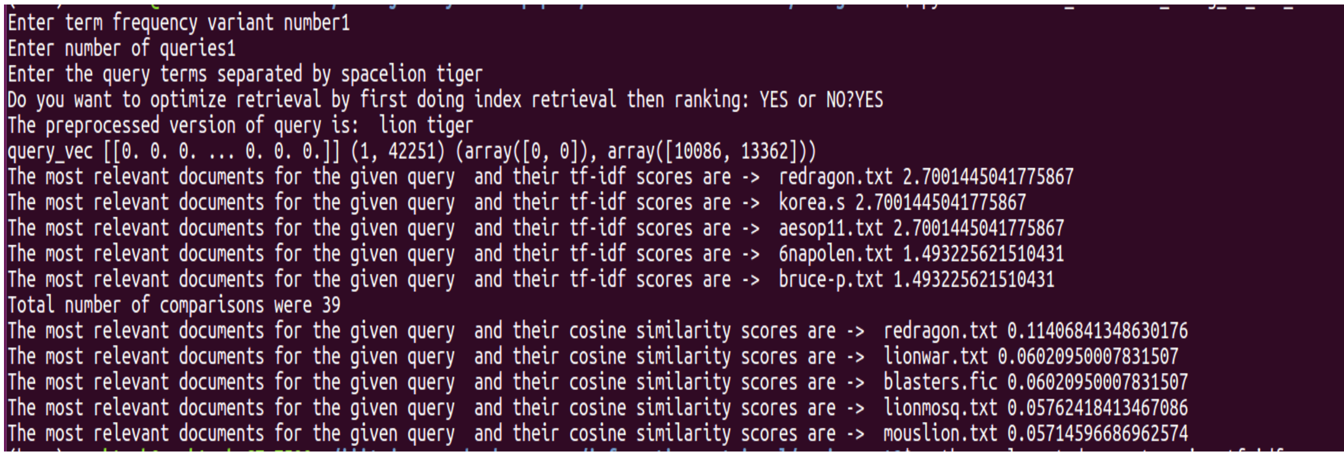
Output (text form):

Enter number of queries **1**
Enter the query with terms separated by space **lion tiger**
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?
YES
query terms **lion tiger**
Number of comparisons **39**
Number of documents retrieved **5**
The top relevant docs are **[('redragon.txt', 0.013157894736842105), ('lionwar.txt', 0.00909090909090909), ('blasters.fic', 0.00909090909090909), ('lionmosq.txt', 0.008333333333333333), ('mouslion.txt', 0.00819672131147541)]**

Analysis:
From the outputs we observe that with the optimization where only K documents that have at least one query term are used for Jaccard similarity computation and finally 5 top documents are given as output. We observe that the optimization reduced 467 computations to 39 documents. For much larger collections this would save a lot of time. However we observe that the Jaccard similarity scores are low. This is because the documents are large and the query consists of lesser number of tokens. Hence the union is larger than the intersection leading to small values for the Jaccard coefficient. Overlap coefficient could be used which uses the min length of the two sets in the denominator

Question 2 part 2 and part 3 outputs:
Output for cosine similarity based relevance ranking for query document vectors with tf variants:

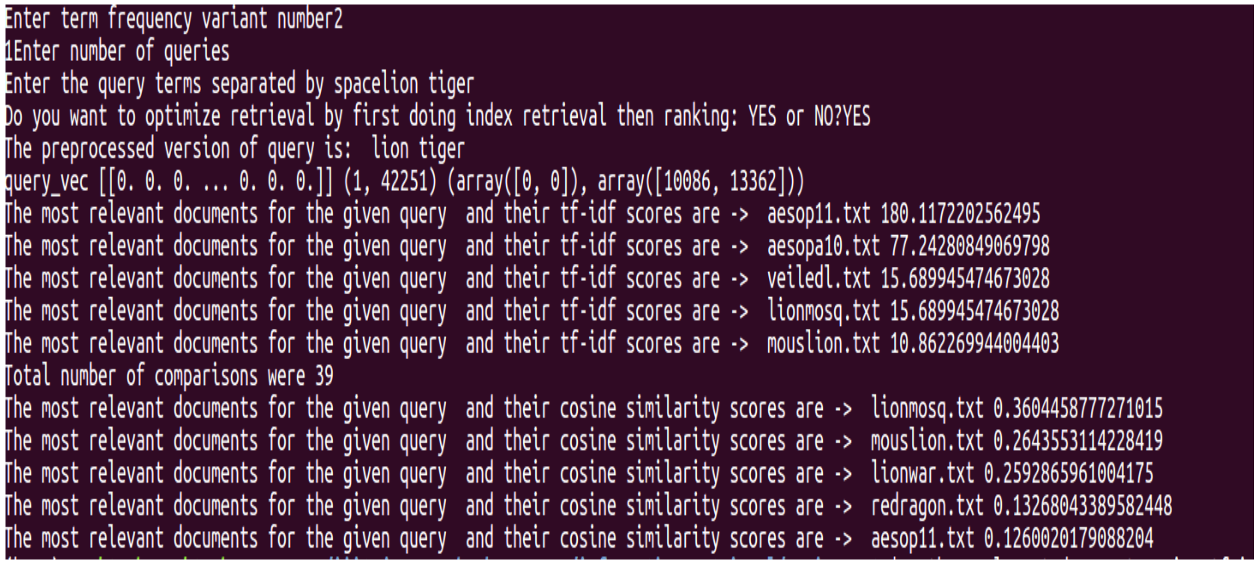
- **Variant 1: Binary:**



```
Enter term frequency variant number1
Enter number of queries1
Enter the query terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
The preprocessed version of query is:  lion tiger
query_vec [[0. 0. 0. ... 0. 0. 0.]] (1, 42251) (array([0, 0]), array([10086, 13362]))
The most relevant documents for the given query  and their tf-idf scores are ->  redragon.txt 2.7001445041775867
The most relevant documents for the given query  and their tf-idf scores are ->  korea.s 2.7001445041775867
The most relevant documents for the given query  and their tf-idf scores are ->  aesop11.txt 2.7001445041775867
The most relevant documents for the given query  and their tf-idf scores are ->  6napolen.txt 1.493225621510431
The most relevant documents for the given query  and their tf-idf scores are ->  bruce-p.txt 1.493225621510431
Total number of comparisons were 39
The most relevant documents for the given query  and their cosine similarity scores are ->  redragon.txt 0.11406841348630176
The most relevant documents for the given query  and their cosine similarity scores are ->  lionwar.txt 0.06020950007831507
The most relevant documents for the given query  and their cosine similarity scores are ->  blasters.fic 0.06020950007831507
The most relevant documents for the given query  and their cosine similarity scores are ->  lionmosq.txt 0.05762418413467086
The most relevant documents for the given query  and their cosine similarity scores are ->  mouslion.txt 0.05714596686962574
```

Analysis: From the output we observe that the tf-idf weighting and the cosine similarity only match in the topmost output. We observe that though the rest of the documents are somewhat relevant to the query terms they are different when using tf-idf weighting and cosine similarity. This may be because tf-idf weighting outputs a single score for a document by summing up tf-idf weights of terms in the query that appear in the document. Moreover here we use binary tf-idf weighting where the frequency of occurrence of a term is not taken into account but rather only the presence or absence of a term is taken into account. Also the tf-idf vector for query would be very sparse but for document it may be less sparse and hence cosine similarity may not reflect semantic similarity though the vectors are normalized by the norm of the vectors. It is also intuitive that the file **redragon.txt** is ranked at top in both the ranking methods as it is the only file that has both the query terms.

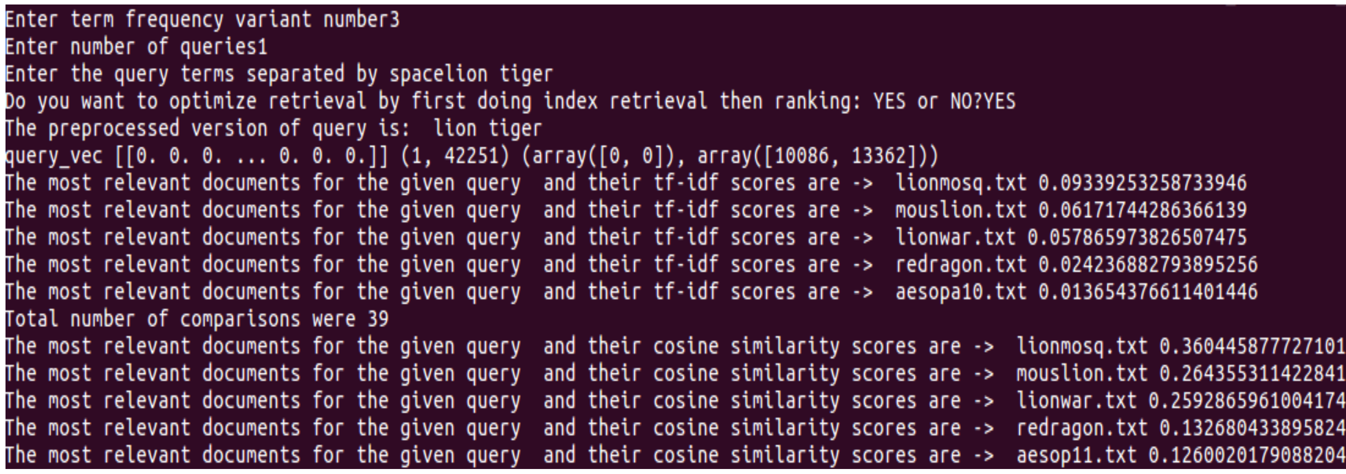
- **Variant 2: raw term frequency:**



```
Enter term frequency variant number2
1Enter number of queries
Enter the query terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
The preprocessed version of query is:  lion tiger
query_vec [[0. 0. 0. ... 0. 0. 0.]] (1, 42251) (array([0, 0]), array([10086, 13362]))
The most relevant documents for the given query  and their tf-idf scores are ->  aesop11.txt 180.1172202562495
The most relevant documents for the given query  and their tf-idf scores are ->  aesopa10.txt 77.24280849069798
The most relevant documents for the given query  and their tf-idf scores are ->  veiledl.txt 15.689945474673028
The most relevant documents for the given query  and their tf-idf scores are ->  lionmosq.txt 15.689945474673028
The most relevant documents for the given query  and their tf-idf scores are ->  mouslion.txt 10.862269944004403
Total number of comparisons were 39
The most relevant documents for the given query  and their cosine similarity scores are ->  lionmosq.txt 0.3604458777271015
The most relevant documents for the given query  and their cosine similarity scores are ->  mouslion.txt 0.2643553114228419
The most relevant documents for the given query  and their cosine similarity scores are ->  lionwar.txt 0.2592865961004175
The most relevant documents for the given query  and their cosine similarity scores are ->  redragon.txt 0.13268043389582448
The most relevant documents for the given query  and their cosine similarity scores are ->  aesop11.txt 0.1260020179088204
```

Analysis: From the output we see that almost 3 documents match between tf-idf weighting based ranking and cosine similarity based ranking. But the ranking order of these documents are different. Primarily we observe that since raw counts are used for tf weighting in **tf-idf** score based ranking **aesop11.txt** which has about 155 occurrences of “lion” and 1 occurrence of tiger is ranked at the top. Similarly **aesop10.txt** has 64 occurrences of lion and hence is ranked second when ranking by tf-idf weighting. However when computing cosine similarity since it is normalized by the norm of the vectors we observe a slight change in the ranking order. The ranking order in cosine similarity also outputs documents like **redragon.txt** which contains both the terms “lion” and “tiger” which was not given as output in tf-idf based ranking.

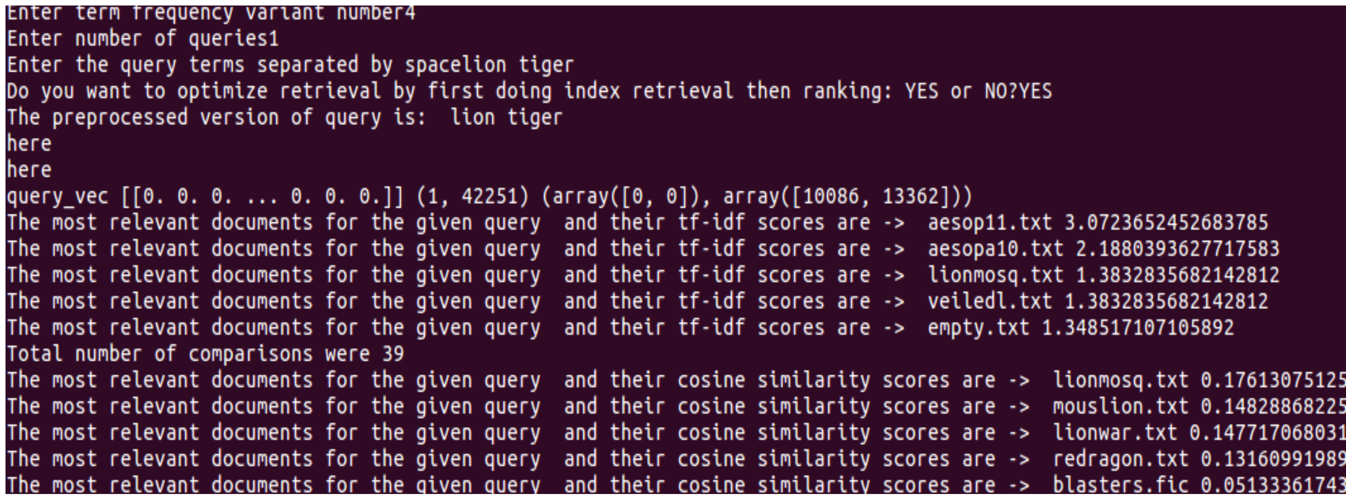
- **Variant 3: term freq variant:**



```
Enter term frequency variant number3
Enter number of queries1
Enter the query terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
The preprocessed version of query is:  lion tiger
query_vec [[0. 0. 0. ... 0. 0. 0.]] (1, 42251) (array([0, 0]), array([10086, 13362]))
The most relevant documents for the given query  and their tf-idf scores are ->  lionmosq.txt 0.09339253258733946
The most relevant documents for the given query  and their tf-idf scores are ->  mouslion.txt 0.06171744286366139
The most relevant documents for the given query  and their tf-idf scores are ->  lionwar.txt 0.057865973826507475
The most relevant documents for the given query  and their tf-idf scores are ->  redragon.txt 0.024236882793895256
The most relevant documents for the given query  and their tf-idf scores are ->  aesopa10.txt 0.013654376611401446
Total number of comparisons were 39
The most relevant documents for the given query  and their cosine similarity scores are ->  lionmosq.txt 0.360445877727101
The most relevant documents for the given query  and their cosine similarity scores are ->  mouslion.txt 0.264355311422841
The most relevant documents for the given query  and their cosine similarity scores are ->  lionwar.txt 0.2592865961004174
The most relevant documents for the given query  and their cosine similarity scores are ->  redragon.txt 0.132680433895824
The most relevant documents for the given query  and their cosine similarity scores are ->  aesop11.txt 0.1260020179088204
```

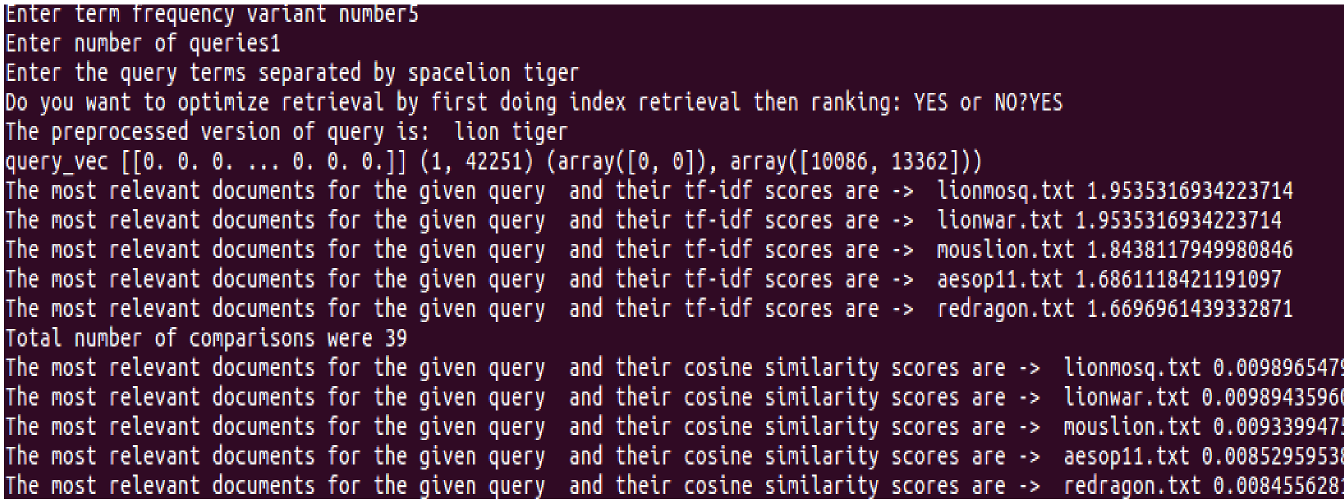
Analysis: Here we see that the ranking order and documents are returned are mostly the same for both the ranking functions except for the last document. This may be because the term frequency is normalized if tf-idf weighting scheme and hence raw count of terms does not dominate.

- **Variant 4: log normalization:**



```
Enter term frequency variant number4
Enter number of queries1
Enter the query terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
The preprocessed version of query is:  lion tiger
here
here
query_vec [[0. 0. 0. ... 0. 0. 0.]] (1, 42251) (array([0, 0]), array([10086, 13362]))
The most relevant documents for the given query  and their tf-idf scores are ->  aesop11.txt 3.0723652452683785
The most relevant documents for the given query  and their tf-idf scores are ->  aesopa10.txt 2.1880393627717583
The most relevant documents for the given query  and their tf-idf scores are ->  lionmosq.txt 1.3832835682142812
The most relevant documents for the given query  and their tf-idf scores are ->  veiledl.txt 1.3832835682142812
The most relevant documents for the given query  and their tf-idf scores are ->  empty.txt 1.348517107105892
Total number of comparisons were 39
The most relevant documents for the given query  and their cosine similarity scores are ->  lionmosq.txt 0.17613075125
The most relevant documents for the given query  and their cosine similarity scores are ->  mouslion.txt 0.14828868225
The most relevant documents for the given query  and their cosine similarity scores are ->  lionwar.txt 0.147717068031
The most relevant documents for the given query  and their cosine similarity scores are ->  redragon.txt 0.13160991989
The most relevant documents for the given query  and their cosine similarity scores are ->  blasters.fic 0.05133361743
```

- **Variant 5: double normalization**



```
Enter term frequency variant number5
Enter number of queries1
Enter the query terms separated by spacelion tiger
Do you want to optimize retrieval by first doing index retrieval then ranking: YES or NO?YES
The preprocessed version of query is:  lion tiger
query_vec [[0. 0. 0. ... 0. 0. 0.]] (1, 42251) (array([0, 0]), array([10086, 13362]))
The most relevant documents for the given query  and their tf-idf scores are ->  lionmosq.txt 1.9535316934223714
The most relevant documents for the given query  and their tf-idf scores are ->  lionwar.txt 1.9535316934223714
The most relevant documents for the given query  and their tf-idf scores are ->  mouslion.txt 1.8438117949980846
The most relevant documents for the given query  and their tf-idf scores are ->  aesop11.txt 1.6861118421191097
The most relevant documents for the given query  and their tf-idf scores are ->  redragon.txt 1.6696961439332871
Total number of comparisons were 39
The most relevant documents for the given query  and their cosine similarity scores are ->  lionmosq.txt 0.0098965479
The most relevant documents for the given query  and their cosine similarity scores are ->  lionwar.txt 0.00989435960
The most relevant documents for the given query  and their cosine similarity scores are ->  mouslion.txt 0.0093399475
The most relevant documents for the given query  and their cosine similarity scores are ->  aesop11.txt 0.00852959538
The most relevant documents for the given query  and their cosine similarity scores are ->  redragon.txt 0.0084556281
```

In this variant we observe that the docuemnts and their rankign are simialr for both tf-idf based weighting and cosine similarity based weighting.

Question 3:
The outputs are pretty straightforward, calculated as expected in questions asked.

Output:

2)
Part 1: Max DCG Score: 28.98846753873482
Part 1: Combinations of files with max DCG:
14344570258120032115933237896806400000000

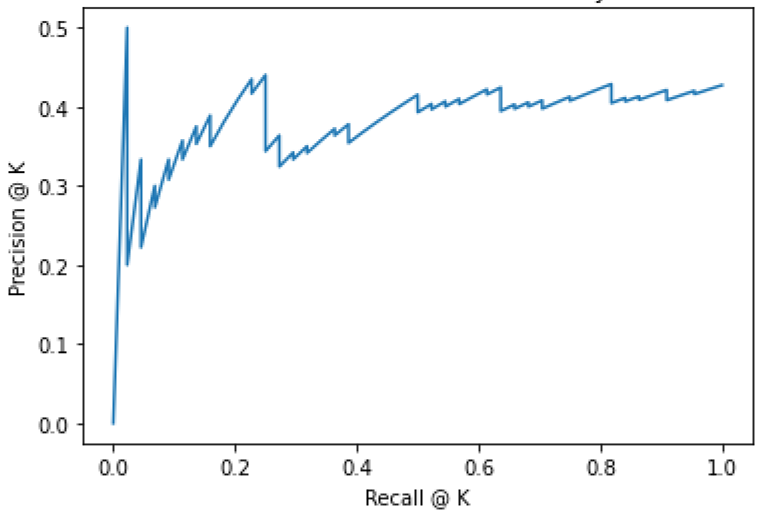
3)

a)
Normalized DCG at 50: 0.35612494416255847

b)
Normalized DCG (for whole dataset): 0.5784691984582591

4) The order of documents with feature 75 tend to be distributed across, and since about 47 documents were relevant out of 103, the precision value reached about 0.42.

Precision vs Recall for documents ranked by tf-idf scores


Precision @ K
Recall @ K