# CMSC 676
# HOMEWORK 3

**Structure of the program:**

The following steps were followed in the writing of the code in this program:

Take in the arguments in the format :

filename input-directory output-directory

I have considered my homework 2 program to be modified on. I introduced a record class to store fixed length entry of an index file. It mainly includes the word, postings and offset of the index file.

**Preprocessing:**

I considered the files batch wise in 10,20,40,80,100,200,300,400,500. After parsing the file as done in the previous files. I created the hashmaps for document frequency and distribution frequency as the previous homework. For creating the frequency hashmap I have used the set function to convert any iterable words to distinct elements and sorted sequences of iterable words.
The remaining aspects of the program were taken from the previous homeowrk program.

**Calculating weights:**

Term Weights: I used the tfidf weights as used in project 2. Here i have considered the tfidf for every token of a file.

**Indexing:**

The following steps where followed for indexing of the files:

1) Considered the tokens of the document frequencies created.
2) Stored the frequency of the tokens on all documents in a separate document.
3) Made a record of the length of the token and increment the location of the postings by 1 to maintain the loaction of the posting file.
4) Added the record to the index.
5) Considered every document that contained the frequency of each token in separate documents and calculated the weights using tfidf. The calculated weights are the normalised values for each term.
6) Then appended this document and weight to all the postings.

**Output files:**

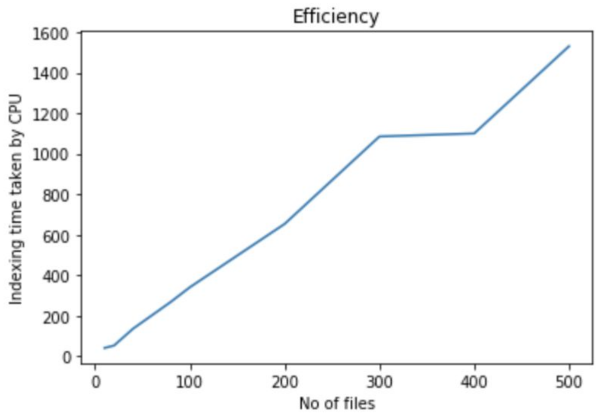
The postings.txt file has the document id and the normalized weight of the word in the document.
The index.txt file has the word, the number of documents that contain that word (which corresponds to the number of records that word gets in the postings file), the location of the first record for that word in the postings file

**Runtime and Analysis:** The below results are rounded off values from the execution of the program.

| No. of files | Elapsed CPU time(sec) | File size of index(kb) | File size of postings(kb) | File size(index + postings)(kb) |
|---|---|---|---|---|
| 10 | 40 | 71 | 157 | 228 |
| 20 | 52 | 110 | 266 | 376 |
| 40 | 135 | 241 | 616 | 857 |
| 80 | 268 | 479 | 1429 | 1908 |
| 100 | 340 | 535 | 1851 | 2386 |
| 200 | 653 | 890 | 3623 | 4514 |
| 300 | 1085 | 1170 | 5191 | 6361 |
| 400 | 1100 | 1426 | 6790 | 8216 |
| 500 | 1531 | 1717 | 9017 | 10734 |

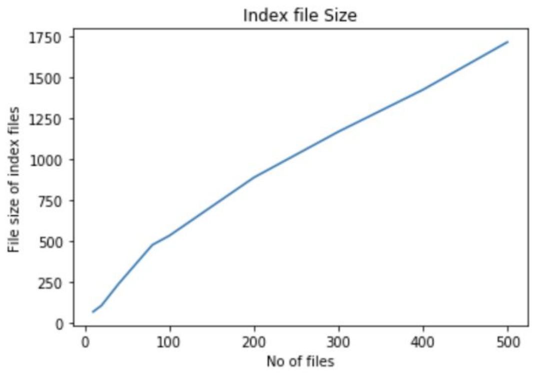
The below graph is of the indexing time as a function of the number of documents. Time includes the elapsed time as well as CPU time.

```
plt.xlabel('No of files')
plt.ylabel('Indexing time taken by CPU')
plt.title("Efficiency");
```



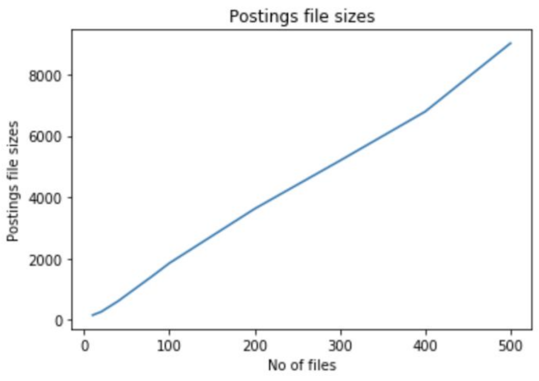
We can observe in the above graph that the time takne by CPU is linealry proportional to the number of files for the first 300 files and then there is no increase for the next 100 files and then there is steep increase in the next 100 files.

```
In [6]:
s = plt.plot([10,20,40,80,100,200,300,400,500], [71 ,110,241,479,535,890,1170,1426,1717])

plt.xlabel('No of files')
plt.ylabel('File size of index files')
plt.title("Index file Size");
```



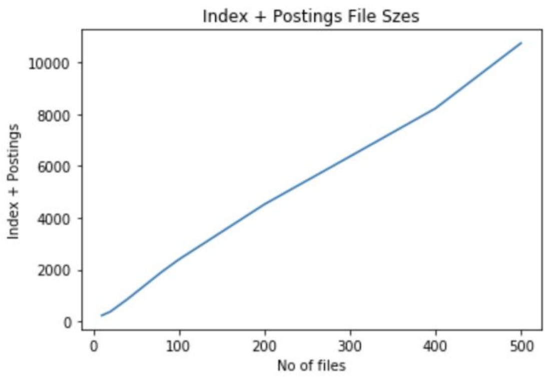
In this graph we can observe the file sizes after the index file contains the indexed terms. At first it increased at a higher rate but after around 100 files it increases consistantly. The file sizes increases as the numnber of files increase.

```
In [7]:
s = plt.plot([10,20,40,80,100,200,300,400,500], [157,266,616,1429,1851,3623,5191,6790,9017])

plt.xlabel('No of files')
plt.ylabel('Postings file sizes')
plt.title("Postings file sizes");
```



The above graph shows that the sixe of the posting files increase as the number of files increases.

```
In [8]:
s = plt.plot([10,20,40,80,100,200,300,400,500], [228,376,857,1908,2386,4514,6361,8216,10734])

plt.xlabel('No of files')
plt.ylabel('Index + Postings')
plt.title("Index + Postings File Szes");
```



The above graph shows that the sizes of the output file increases as the numebr of files increase.

Therefore, we can conclude that the as the input size increaes the running time and the size increases.