# Written Submission For

# CSC 483/583: Assignment #1 (Qns 1 to 4)

# Spring 2017

**By Mithun Paul**

Today's Date: 27th Jan 2017

Due by 11:59 P.M., January 29 (upload both code and report to D2L)

Because the credit for graduate students adds to more than 75 points, graduate students grades will be normalized at the end to be out of 75. For example, if a graduate student obtains 80 points on this assignment, her final grade will be 80 × 75/85 = 70.6. Under- graduate students do not have to solve the problems marked grad students only. If they do, their grades will not be normalized but will be capped at 75. For example, if an undergraduate student obtains 80 points on this project (by getting some credit on the grad students only problem), her final grade will be 75.

Note that the first four problems do not require coding. Only problem 5 requires coding.

## Problem 1 (15 points)

Consider these documents:

Doc 1 breakthrough drug for schizophrenia

Doc 2 new approach for treatment of schizophrenia

Doc 3 new hopes for schizophrenia patients

Doc 4 new schizophrenia drug

**1. Draw the term-document incidence matrix for this document collection.**

**Ans:**

| | Doc1 | Doc2 | Doc3 | Doc4 | Doc5 |
|---|---|---|---|---|---|
| | | | | | |
| breakthrough | 1 | 0 | 0 | 0 | 0 |
| drug | 1 | 0 | 0 | 1 | 0 |
| for | 1 | 1 | 1 | 0 | 0 |
| schizophrenia | 1 | 1 | 1 | 1 | 0 |
| new | 0 | 1 | 1 | 1 | 0 |
| approach | 0 | 1 | 0 | 0 | 0 |
| treatment | 0 | 1 | 0 | 0 | 0 |
| of | 0 | 1 | 0 | 0 | 0 |
| hopes | 0 | 0 | 1 | 0 | 0 |
| patients | 0 | 0 | 1 | 0 | 0 |

**2. Draw the inverted index representation for this collection, as in Figure 1.3 in IIR.**

**Ans:**

| | | | | | |
|---|---|---|---|---|---|
| breakthrough | ----------> | 1 | | | |
| drug | ----------> | 1 | 4 | | |
| for | ----------> | 1 | 2 | 3 | |
| schizophrenia | ----------> | 1 | 2 | 3 | 4 |
| new | ----------> | 2 | 3 | 4 | |
| approach | ----------> | 2 | | | |
| treatment | ----------> | 2 | | | |
| of | ----------> | 2 | | | |
| hopes | ----------> | 3 | | | |
| patients | ----------> | 3 | | | |

**3. What are the returned results for these queries:**

**(a) schizophrenia AND drug**

**Ans: 1,4**

i.e doc1 and doc4 has these two words

**Steps are as below:**

1. schizophrenia AND drug
2. (1,2,3,4) AND (1,4)
3. (1,4)

**(b) for AND NOT(drug OR approach)**

**Ans: doc3**

**Steps are as below:**

1. for AND NOT (1,4 OR 2)
2. for AND NOT (1,4,2)
3. for AND (3)
4. (1,2,3) AND (3)

## Problem 2 (20 points)

1. Write out a postings merge algorithm, in the style of Figure 1.6 in IIR, for an x OR

y query.

**Ans:**

Algorithm/Assuming these steps are done:

1. Locate x in the Dictionary
2. Retrieve its postings=p1
3. Locate y in the Dictionary
4. Retrieve its postings=p2
5. Disjunct the two postings lists, as shown in the function below.

Note: I was finding it difficult to understand/write the format of pseudocode given in textbook. So instead I wrote an actual code itself as given below. This a python 2.7 function tested against various sample input lists.

**Solution 1: Disjunction (X or Y):**

This solution uses two lists and two pointers like in the vanilla algorithm shown in class.

```
list1=[4,55,90,0]
list2=[5,6,7]
def disjunctionWithLists(list1,list2):
  disjList=[]
  childListCounter=0
  parentListCounter=0
  if(list1.__len__() > list2.__len__()):
     parentlist = list1
     childlist = list2
  else:
     parentlist = list2
     childlist = list1
  while(parentListCounter< parentlist.__len__() and childListCounter < childlist.__len__()):
     if(childlist[childListCounter] == parentlist[parentListCounter]):
        disjList.append(childlist[childListCounter])
        childListCounter=childListCounter+1
        parentListCounter=parentListCounter+1
     else:
        if(childlist[childListCounter] < parentlist[parentListCounter]):
           disjList.append(childlist[childListCounter])
           childListCounter=childListCounter+1
        else:
           disjList.append(parentlist[parentListCounter])
           parentListCounter=parentListCounter+1
  #once we break out of the loop- which means the smaller list has finished, now write out the left over in parent list
  while (parentListCounter < parentlist.__len__()):
     disjList.append(parentlist[parentListCounter])
     parentListCounter = parentListCounter + 1
  return disjList

output= disjunctionWithLists(list1,list2)
print "List1=" + `list1`
print "List2=" + `list2`
print "XorY=" + `output`
```

## Sample inputs and outputs:

1. Sample 1
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/disjunction.py
    b. List1=[4, 55, 90, 0]
    c. List2=[5, 6, 7]
    d. XorY=[4, 5, 6, 7, 55, 90, 0]
    e. Process finished with exit code 0
2. Sample 2
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/disjunction.py
    b. List1=[4, 55, 90, 0]
    c. List2=[5, 6, 7, 66, 33, 777, 745, 565]
    d. XorY=[4, 5, 6, 7, 55, 66, 33, 90, 0, 777, 745, 565]
    e. Process finished with exit code 0
3. Sample 3
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/disjunction.py
    b. List1=[]
    c. List2=[5, 6, 7, 66, 33, 777, 745, 565]
    d. XorY=[5, 6, 7, 66, 33, 777, 745, 565]
    e. Process finished with exit code 0
4. Sample 4
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/disjunction.py
    b. List1=[5, 6, 7, 66, 33, 777, 745, 565]
    c. List2=[]
    d. XorY=[5, 6, 7, 66, 33, 777, 745, 565]
    e. Process finished with exit code 0
5. Sample5:
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/disjunction.py
    b. List1=[]
    c. List2=[]
    d. XorY=[]
    e. Process finished with exit code 0

**Solution 2:**

This is another equivalent function using hashtables. This was done before it was told on piazza, not to use hashmaps. But I am keeping it nevertheless.

```
def disjunctionWithHash(list1,list2):
  disjHash={}
  for items1 in list1:
     if (items1 in disjHash):
        print "item already in list"
     else:
        disjHash[items1]=1

  for items2 in list2:
     if (items2 in disjHash):
        print "item already in list"
     else:
        disjHash[items2]=1
  return disjHash
```

**Solution 3:**

Assumption: the highest document id is tangible. Say, 1000. In that case here is an alternate method which can save some memory space

1. Create a bit vector of n, where n is the highest number of document id.
    a. i.e if 1000 documents= a bit array of 1000 bits.
2. Calloc it to all zeroes.
3. Go to postings list one, for each doc id, flip the corresponding position's bit to one.
    a. Eg: if term X occurs first in document 25, all bits will be 0 except 25th bit.
    b. Do the same for all the ids of documents in which term X is present.
4. Go to list two, for each of the document id, check if bit is 1 or 0. If 1, leave it as is, else Flip it to 1.
    a. Eg: if term Y occurs in document 2, Go to position 2 of the bit array. If the bit is zero flip it to one. Else leave as is.
    b. Eg: if term Y occurs in document 25, Go to position 25 of the bit array. Since the bit was flipped to one in step 3, leave as is.
    c. Do the same for all the ids of documents in which term Y is present.
5. In the end all the position of all the bits which have 1 will be X or Y value
6. Qn) what is the advantage?
    a. Ans: less memory used. You don't have to use 1000*4 Bytes for storing each integer. Might work out cheaper if the number of terms is high.

**2. 2. Write out a postings merge algorithm, in the style of Figure 1.6 in IIR, for an x AND NOT y query.**

**Ans: Solution 1. This is tested on python 2.7 for multiple list cases**

```
list2=[6,8,999,292929]
list1=[2]

def xNotYWithLists(list1, list2):
  childListCounter=0
  parentListCounter=0
  disjList = []
  parentlist = list1
  childlist = list2
  while (parentListCounter < parentlist.__len__() ):
     #quit only if the first list is done
     if (childListCounter >= childlist.__len__()):
        # if the Y list ends, Then write down the rest from the X list to the output
        #then keep going until the parent list ends
        disjList.append(parentlist[parentListCounter])
        parentListCounter = parentListCounter + 1
        continue
     if(childlist[childListCounter] == parentlist[parentListCounter]):
        childListCounter=childListCounter+1
        parentListCounter=parentListCounter+1
     else:
        if(childlist[childListCounter] < parentlist[parentListCounter]):
           childListCounter=childListCounter+1
        else:
           disjList.append(parentlist[parentListCounter])
           parentListCounter=parentListCounter+1
  return disjList

output= xNotYWithLists(list1,list2)
print "List1=" + `list1`
print "List2=" + `list2`
print "XnotY=" + `output`
```

## Sample inputs and outputs:

1. Sample 1
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/xNotY.py
    b. List1=[6, 8]
    c. List2=[2, 6, 9, 99, 879]
    d. XnotY=[8]
    e. Process finished with exit code 0
2. Sample 2
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/xNotY.py
    b. List1=[6, 8, 9, 999, 292929]
    c. List2=[2, 6, 9, 99, 879]
    d. XnotY=[8, 999, 292929]
    e. Process finished with exit code 0
3. Sample 3
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/xNotY.py
    b. List1=[6, 8, 999, 292929]
    c. List2=[2, 6, 9, 99, 879]
    d. XnotY=[8, 999, 292929]
    e. Process finished with exit code 0
4. Sample 4
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/xNotY.py
    b. List1=[6, 8, 999, 292929]
    c. List2=[2]
    d. XnotY=[6, 8, 999, 292929]
    e. Process finished with exit code 0
5. Sample5:
    a. /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/mithun/PycharmProjects/hw1cs583qn2/xNotY.py
    b. List1=[2]
    c. List2=[6, 8, 999, 292929]
    d. XnotY=[2]
    e. Process finished with exit code 0

**Solution 2: The solution below gives the same output but uses hash tables.**

```
def xNotYWithHash(list1,list2):
  xNotYhash={}
  for items1 in list1:
     if (items1 in xNotYhash):
        print "item already in list"
     else:
        xNotYhash[items1]=1

  for items2 in list2:
     if (items2 in xNotYhash):
        del xNotYhash[items2]
        print "item already in list. remove it from the list"
     else:
        print""
  return xNotYhash
```

## Problem 3 (10 points)

Recommend a query processing order for:

(tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

given the following postings list sizes:

Term Postings size

eyes 213312

kaleidoscope 46653

marmalade 107913

skies 271658

tangerine 87009

trees 316812

**Ans:**

**Bottomline**: Whenever there are multiple AND queries, pick the one that has the smallest list, preferably on either sides of the AND query. The advantage is that, if the pointer of the smaller list hits null, we can stop the conjunction process altogether.

**Step 1**: Translate the query to corresponding postings size:

(tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

I.e

(87009 OR 316812) AND (107913 OR 271658) AND (46653 OR 213312)

**Step 2**: For OR query: it does not matter which you pick first, since both will be zipped together.

Note: If two postings have common terms, it is better to do copying using two parallely moving pointers- along with checking for duplicates. This will reduce complexity from O(n+m) where n and m are the sizes of postings, respectively.

**Case 1:** assumption, there are no common document ids between tangerine postings list and trees postings list.

So summing both postings size to denote OR operation (and assuming there are no duplicates) we get new postings size as:

(403821) AND (379571) AND (259965)

**Step 3:** Process the AND query which contains the smallest sizes.

Process (379571) AND (259965) = **Result** first .

Then process **Result** AND (403821).

In other words, the order of query processing can be written as follows:

1. (tangerine OR trees)
2. (marmalade OR skies)
3. (kaleidoscope OR eyes)
4. (marmalade OR skies) AND (kaleidoscope OR eyes)
5. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

Or in other words, the order of query processing can also be written as follows:

1. (tangerine OR trees) = Result1
2. (marmalade OR skies) =Result2
3. (kaleidoscope OR eyes)=Result3
4. Result2 AND Result3= Result4
5. Result4 AND Result1

**Case 2: Assumption: There are say 50% common document ids between Tangerine postings list and Trees postings list.**

Its not possible to estimate the exact OR operation details of the case when there are common document ids in both the postings list, since no such detail is given. However, like I mentioned earlier, that will have an impact on the order of query processing order. A good thumb rule to follow in that case is (i.e to do an OR operation) to run two pointers like the vanilla algorithm, and quit when the smaller list (say Tangerine list) ends. Plus add the left over document ids from the larger list (say Trees).

## Problem 4 (10 points)

How should the Boolean query **x OR NOT y** be handled? Why is the naive evaluation of this query normally very expensive? Write out a postings merge algorithm that evaluates this query efficiently.

Ans:

Various solutions for the naive evaluation of the query **(x OR (NOT y))** can be done as follows.

**Naive Solutions:**

**Solution 1:**

Assumption : We already know the entire list of document ids.

Steps:

1. Run through the master list of document ids, insert all the document ids into a hashmap. (Complexity= O(m) where m is the total number of document ids.)
2. Then run through the postings list of Y, search if the document id in Y's list, exists in the hashmap , if yes, remove it from the hashmap.(Complexity= O(n) where n is the total number of document ids in Y's postings list.)
3. Once we are done going through the postings list of Y, the left over hashmap will represent (Not Y).
4. Merge it with X using the DISJUNCTION algorithm shown in Problem 2.1 above.
    a. (Complexity= O(r +q) where r is the total number of document ids in the list from step 3 above (NOT Y) and q is the total number of document ids in X's posting list .)

For example:

Say the total list of document ids (and the postings for the terms X and Y) are as follows:

A = [23, 34, 555, 765, 876]

X= [34, 765]

Y= [23, 765, 876 ]

Not Y= Everything in A but not In Y = [ 34, 555]

Therefore **(x OR (NOT y)) =** [34, 765] OR [ 34, 555] = [34, 555, 765]

Note: Step 2 can be done with two parallel pointers also, instead of hashmap, which is more

efficient.

**Solution 2:**

Assumption: we know only the dictionary postings list of all the terms, but do not know the entire list of the document ids.

Then to get (Not Y) :

1. Take each word, copy their postings list into a big linked list (Note: just copy the list, do not traverse it now), go to the next word, copy postings list etc...until all the terms in the dictionary and their corresponding postings list are covered. Note: omit Y , obviously
2. Sort this list (Complexity =O(N Log N) where N is the total number of items in the list from step 1)
3. Find and remove duplicates by running two nearby pointers. (O(N) where N is the total number of items in the list from step 1).
4. This is (X or (Not Y))
5. Note: this is because X is included in the parent list.

**For example:**

Say the inverted index of 3 terms, A, X and Y looks like this:

A = [1,2,3,4,5,6,7,8,9,0]

X= [3,4]

Y= [6,7]

**(x OR (NOT y))=** Everything in [A] (which includes X) but not In Y = [1,2,3,4,5,8,9,0]

I.e Go through each of the words, for each word, (go through each of the document id, add it to a hashtable). Then go to the term Y, for each of its members in postings list, remove that id from the hashtable.

**Solution 3:**

Assumption: we have only the dictionary postings list of all the terms, but do not know the entire list of the document ids.

Then to get (Not Y) :

1. Take each word (Except Y), traverse their postings list .
2. For each document id check if it exists in the hashmap
    a. If yes, don't do anything.
    b. Else add it to the hashmap.
3. (Complexity =O(N) where N is the total number of document ids in all the postings.
4. The keys of the hashmap thus formed is (X or (Not Y)) - Assuming X is included in the parent superset, A

**Non-Naive solution:**

**Solution 4:**

Given equation: **X OR (NOT Y) ----Eqn (0)**

Consider DeMorgan's laws:

1. **NOT(A OR B) = NOT(A) AND NOT(B) ----Eqn (1)**
2. **NOT(A AND B)= NOT(A) OR NOT(B) -----Eqn (2)**

Lets define a super set U, which includes X , Y and other sets (which are terms in this case.)

Lets call (NOT Y) as Q. i.e Q= (NOT Y)

Now lets consider negating the given Eqn (0)

i.e : **NOT** (**X OR Q ) --- Eqn (3)**

Using Eqn 1 on Eqn 3, we get:

**Eqn 3 = (NOT(X) AND NOT(Q) ) ----Eqn (4)**

Expanding Q and substituting in right hand side of Eqn 4, we see that (NOT(NOT Y))= Y

So we get:

**Eqn 4 = ((NOT(X)) AND Y) ---- Eqn (5)**

Since we defined U as a super set:

(Not X)= U- X.

Therefore we can write:

**Eqn 5 = ((U-X) AND Y) ----Eqn (6)**

Refer: Algebra of relative complements in this link.

Using the Algebra of relative complements i.e

((B -A) AND C) = ((B AND C) - A) = (B AND (C -A))

we can rewrite Eqn 5 as:

**Eqn 6=(U AND (Y - X)) ---Eqn (7)**

Since U is defined as an all inclusive set which includes X and Y (i.e since Y - U =null), intersection of U with any set is the set itself.

**i.e (U and A)= A ---Eqn (8)**

For example if U={1,2,3,4,5} and A ={5}

(U AND A) ={5}

Using Eqn (8) on Eqn (7) we get:

**Eqn 7 =(U AND (Y - X)) = (Y - X) ----Eqn (9)**

**This is a much easier form to solve than the starting form given in Eqn (0)** . By easier, I mean, lesser complexity. .e the complexity of this is O(n+m), where n= number of document ids in postings list of Y and m is the number of documents ids in the postings list of X. Then take another NOT to get back to actual equation.

Compare this to the running complexity of Eqn 0.

The postings merge algorithm to find Y- X is given as a solution to Qn 2.2 above (i.e Y - X= Y and NOT X).

Once you have m=Y-X, to take NOT(m), you just need to pick out everything in U, except the elements in m. Use the same postings merge algorithm.

**Consider the following example.**

U= list of all works of shakespeare= {1,2,3,4,5,6,7,8,9}

X= list of works in which the word BRUTUS appears = {2,5,6}

Y= list of works in which the word CAESER appears = {1,2,4}

The query is (X or (NOT Y)) i.e give me the list of all documents which has (BRUTUS or (DOES NOT HAVE CAESER).

Finding all documents that does not have CAESER maens we will have to go through all the documents in U and find if CAESER exists in it. That is O(U)- i.e you have to go through almost all the documents in the super set U.= say this list is = M

Combining both these lists (M and X) means = O(M+X).

And then removing duplicates (to correctly get OR) = another O(M+X).

Instead if you use my formula:

To find (Y-X) - i.e all documents that exist in Y not in X= worst case O(Y+X)= {1,2,4}-{2,5,6}= {1,4}

Which means you don't have to go through the entire U list.

Now for the final step, to take NOT(Y-X) i.e to get back to Eqn 0, create a bit vector of size maximum document id and initialize everything to zero. Just flip the bits in positions corresponding to {1, 4}- we got from (Y-X) step above. Just print the rest, that will be your answer. O(1) complexity. You can even choose to use a hashmap instead of a bit vector.

Bottom line: Complexity wise an AND operation is anytime better than OR operation.

**Quad Erad Demonstradum (Q.E.D)**

## Problem 5 (20 points undergrad / 30 grad)

Implement an inverted index that supports Boolean search. Your program must take in one file containing one document per line, in a format close to the one used in Problem 1. For example, you can use the file below to test your code:

```
Doc 1 breakthrough drug for schizophrenia
Doc 2 new approach for treatment of schizophrenia
Doc 3 new hopes for schizophrenia patients
Doc 4 new schizophrenia drug
```

In particular, the format requires:

• One document per line in the input file.

• The first token in each line be the document id. The id is not to be indexed as a term!

• The rest of tokens in a line be all terms appearing in the document. Assume that the text is already tokenized and the tokens are normalized.

To code this problem, you can use any programming language that is familiar to the instructor (C/C++, Java, Scala, Python). You can use data structures available in your programming language of choice, e.g., dictionaries in Python or hash maps in Java/Scala, but you are not allowed to use open-source code that implements inverted indices, such as Lucene. You have to implement the inverted index and corresponding search operations from scratch.

The code submitted must compile and run. You must also include in your submission a Makefile/pom.xml/build.sbt file or shell script that allows the instructor to run the code, together with a README file that describes usage.

Please implement the following:

**Note: Code is submitted herewith in D2L as a zip file of sbt folder. Below are just the milestones/stepsCovered in the project**

1. (20 points) Construct the inverted index and implement support for binary AND queries. What does your code return for the file above and the query: schizophrenia AND drug?

**Todo**:

1. ~~Read from command line~~
2. ~~Split and check if AND or OR occurs~~
3. ~~Test with random two words~~
4. ~~Test with schizophrenia AND drug~~
    a. ~~Answer must be [1,4]~~
5. ~~Check if you find term that does not exist- throw a try catch error~~
6. ~~Give chance for user to enter another term even if he succeeds.~~

2. (GRAD STUDENTS ONLY: 5 points) Add support for binary OR queries. What

does your code return for the query: breakthrough OR new?

**Todo:**

1. ~~Copy paste and make conjunction run for disjunction.~~
2. ~~Test with random two words~~
3. ~~Test with breakthrough OR new~~
    a. ~~Answer must be [1,4]~~

3. (GRAD STUDENTS ONLY: 5 points) Add support for queries that combine multiple binary AND and OR operators, such as: (drug OR treatment) AND schizophrenia. You can choose a default operator priority, e.g., AND has a higher priority than OR, or use parentheses to make processing order explicit. What does your code return for this query?

1. Read on SCALA-PARSER-COMBINATORS
    a. Done. This is a very good starting tutorial
2. Implement a basic parser that identifies a word ---Done
    a. Capture word followed by a number
3. Extend your parser to identify ( ---Done
4. Make your parser recognize a word after (---Done
5. Make your parser recognize AND after ( and word---Done
6. Make your parser recognize OR after ( and word---Done
7. Make your parser recognize term1 , term2 and operator---Done
8. Make your parser recognize (drug OR treatment) ---Done
9. Make your parser recognize (drug AND treatment) ---Done
10. Feed all 3 above to a simple boolean query code---Done
11. Get results for (drug OR treatment)---Done
    a. Done. results are as expected.
12. AND schizophrenia ----done
    a. Make multiple terms compatible
    b. Look for closing of parenthesis, else throw error.
13. Negative test cases---Done

| **Query** | **Expected Result** | **My Result** |
|---|---|---|
| (drug OR treatment) AND schizophrenia | PASS =[1,2,4] | PASS |
| (drug OR treatment | FAIL | FAIL |
| drug OR treatment AND schizophrenia | FAIL | FAIL |
| (drug OR treatment) | PASS =[1,2,4] | PASS |

| | | |
|---|---|---|
| (drug OR treatment) AND (hopes OR schizophrenia) | PASS=[1,2,3,4] | FAIL |
| drug OR treatment | PASS =[1,2,4] | FAIL |
| drug AND schizophrenia | PASS=[1,4] | FAIL |
| (drug OR hopes) AND schizophrenia) | FAIL | PASS |
| (drug OR hopes) AND schizophrenia | PASS=[1,3,4] | PASS |
| ((drug OR treatment) AND schizophrenia) AND ((drug OR hopes) AND schizophrenia) | PASS=[1,2,4] AND [1,3,4] = [1,4] | FAIL |
| ((drug OR treatment) AND schizophrenia) AND new) | PASS | FAIL |

a.

14. Test and see if it can be used for qn5.2 and qn 5.3
    a. Make parenthesis optional

**Ans: code for this will be uploaded to D2L.**