1. Draw the inverted index that would be built for the following document collection.

**Doc 1** new pants sales top forecasts
**Doc 2** pants sales rise in july
**Doc 3** increase in pants sales in july
**Doc 4** july new pants sales rise

*(I am listing all the steps in the first exercise just for internalizing the process better. The answers in the subsequent exercises are not as detailed)*

**Step 1**: tokenizing the text:

| new | pants | sales | top | forecasts | rise | in | july | increase |
|---|---|---|---|---|---|---|---|---|

**Step 2**: Linguistic processing:

| new | pants | sale | top | forecast | rise | in | july | increase |
|---|---|---|---|---|---|---|---|---|

**Step 3**: generate postings:

| term | doc |
|---|---|
| new | 1 |
| pants | 1 |
| sale | 1 |
| top | 1 |
| forecast | 1 |
| pants | 2 |
| sale | 2 |
| rise | 2 |
| in | 2 |
| july | 2 |
| increase | 3 |
| in | 3 |
| pants | 3 |
| sale | 3 |
| in | 3 |
| july | 3 |
| july | 4 |
| new | 4 |
| pants | 4 |
| sale | 4 |
| rise | 4 |

**Step 4**: sort postings:

| term | doc |
|---|---|
| forecast | 1 |
| in | 2 |
| in | 3 |
| in | 3 |
| increase | 3 |
| july | 2 |
| july | 3 |
| july | 4 |
| new | 1 |
| new | 4 |
| pants | 1 |
| pants | 2 |
| pants | 3 |
| pants | 4 |
| rise | 2 |
| rise | 4 |
| sale | 1 |
| sale | 2 |
| sale | 3 |
| sale | 4 |
| top | 1 |

**Step 5)** create lists, count document frequency

| term | doc frequency | docs | within doc frequency |
|---|---|---|---|
| forecast | 1 | 1 | 1(1) |
| in | 2 | 2, 3 | 2(1), 3(2) |
| increase | 1 | 3 | 1(1) |
| july | 3 | 2, 3, 4 | 2(1), 3(1), 4(1) |
| new | 2 | 1, 4 | 1(1), 4(1) |
| pants | 4 | 1, 2, 3, 4 | 1(1), 2(1), 3(1), 4(1) |
| rise | 2 | 2, 4 | 2(1), 4(1) |
| sale | 4 | 1, 2, 3, 4 | 1(1), 2(1), 3(1), 4(1) |
| top | 1 | 1 | 1(1) |

**Step 6)** split the result into a lexicon and a postings list – this is now our **Inverted Index**

| | | |
|---|---|---|
| forecast | 1 | 1 |
| in | 2 | 2 3 |
| increase | 1 | 3 |
| july | 3 | 2 3 4 |
| new | 2 | 1 4 |
| pants | 4 | 1 2 3 4 |
| rise | 2 | 2 4 |
| sale | 4 | 1 2 3 4 |
| top | 1 | 1 |

lexicon postings lists

---

2. Consider these documents:

**Doc 1** breakthrough drug for peace
**Doc 2** new peace drug
**Doc 3** new approach for peace in our time
**Doc 4** new hopes for peace process

(a) Draw the term-document incidence matrix for the collection.

(b) Draw the inverted index representation of the collection.

(c) What are the returned results for these queries:

i. peace **AND** drug
ii. for **AND NOT** (drug **OR** approach)

**a)**

| | **doc 1** | **doc 2** | **doc 3** | **doc 4** |
|---|---|---|---|---|
| **approach** | 0 | 0 | 1 | 0 |
| **breakthrough** | 1 | 0 | 0 | 0 |
| **drug** | 1 | 1 | 0 | 0 |
| **for** | 1 | 0 | 1 | 1 |
| **hopes** | 0 | 0 | 0 | 1 |
| **in** | 0 | 0 | 1 | 0 |
| **new** | 0 | 1 | 1 | 1 |
| **our** | 0 | 0 | 1 | 0 |
| **peace** | 1 | 1 | 1 | 1 |
| **process** | 0 | 0 | 0 | 1 |
| **time** | 0 | 0 | 1 | 0 |

**b)**

| | | | | | |
|---|---|---|---|---|---|
| **approach** | 1 | 3 | | | |
| **breakthrough** | 1 | 1 | | | |
| **drug** | 2 | 1 | 2 | | |
| **for** | 3 | 1 | 3 | 4 | |
| **hope** | 1 | 4 | | | |
| **in** | 1 | 3 | | | |
| **new** | 3 | 2 | 3 | 4 | |
| **our** | 1 | 3 | | | |
| **peace** | 4 | 1 | 2 | 3 | 4 |
| **process** | 1 | 4 | | | |
| **time** | 1 | 3 | | | |

**b) Query results for:**

i) peace **AND** drug **= 1 , 2**

ii) for **AND NOT (**drug **OR** approach**) = 4**

---

3. In the lecture we went through an algorithm for intersecting two lists involved in an **AND** query that ran in time $O(x + y)$. For the queries below, can we still run through the intersection in time $O(x + y)$? If not, what can we achieve?

   (a) Brutus **AND NOT** Caesar

   (b) Brutus **OR NOT** Caesar

a) We would have to get the listing for **NOT** Caesar. Then, if we have the listing for "Brutus" and we have the listing for "**NOT** Caesar", and if they are both ordered lists, we can go ahead run through the intersection in time *O(x + y2)* . (if y was the length of the listing for Caesar, y2 is the length of the listing for **NOT** Caesar. y and y2 are unlikely to be the same length)

b) Again, the length for the listing of NON Caesar (y2) is unlikely to be the same size as y (length of the listing for Caesar). It will take time *O(x + y2)* to run through the terms.

---

4. Recommend a query processing order for the following query:

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

   given the following posting list sizes:

   - eyes: 213312
   - kaleidoscope: 87009
   - marmalade: 107913
   - skies: 271658
   - tangerine: 46653
   - trees: 316812

We first look at the sizes for the **OR** listings:

tangerine OR trees = 46653 + 316812 = 363465

marmalade OR skies = 107913 + 271658 = 379571

kaleidoscope OR eyes = 87009 + 213312 = 300321

To optimize the query, we will want to process in terms of increasing frequency, i.e., start with the two shortest lists, intersect, then intersect with the remaining list:

i) *Process* (kaleidoscope **OR** eyes) **AND** (tangerine **OR** trees) (the two shortest lists)

ii) *Process the result from i)* **AND** (marmalade **OR** skies)

To put it in just one line: [(kaleidoscope **OR** eyes) **AND** (tangerine **OR** trees)] **AND** (marmalade **OR** skies)

---

5. Write pseudocode for an algorithm for an $x$ **OR** $y$ query (in the style of the algorithm in the lecture slides for an $x$ **AND** $y$ query).

```
queryOR(p, q)
1       answer <- { }
2       while p ≠ null and q ≠ null do
3               if docID(p) = docID(q) then
4               # if the docID is the same, it only gets added once
5                       Add(answer, docID(p))
6                       p <- next(p)
7                       q <- next(q)
8               else if docID(p) < docID(q) then
9               # if the docID is not the same, both will get added – in the correct order
10                      Add(answer, docID(p))
11                      Add(answer, docID(q))
12                      p <- next(p)
13                      q <- next(q)
14              else
15                      Add(answer, docID(q))
16                      Add(answer, docID(p))
17                      p <- next(p)
18                      q <- next(q)
19      end while
20      return answer
```

---

6. How should the Boolean query $x$ **AND NOT** $y$ be handled? Why is naive evaluation of this query normally very expensive? Write pseudocode for an algorithm that evaluates this query efficiently.

In this context, "Expensive" is a measure of processing time, memory usage and disk CPU and IO.

There are more likely a lot fewer documents that contain a term than documents that do not contain a term. If we go for a "naïve" approach to "AND NOT", we would be intersecting the first term with a list of "NOT 2nd term", which could be huge and tax our resources (memory, CPU, IO, time).

---

7. Try using the Boolean search features on a couple of major web search engines. For instance, choose a word, such as hacker, and submit the queries (i) hacker, (ii) hacker AND hacker, and (iii) hacker OR hacker. Look at the estimated number of results and top hits. Do they make sense in terms of Boolean logic? Often they do not. Can you make sense of what is going on? What about if you try different words? For example, query for (i) author, (ii) prize, and then author OR prize. What bound should the number of results from the first two queries place on the third query? Is this what you see?

These are the results from my first queries:

| **Google search:** | | |
|---|---|---|
| **query** | **nr of results** | **top hit** |
| hacker | 354 000 000 | Wikipedia: Hacker |
| hacker AND hacker | 168 000 000 | Wikipedia: Security Hacker |
| hacker OR hacker | 339 000 000 | ZDNet: "Hacker leaks data of 2.28 million dating site users" |

| **Microsoft Edge (Bing) search:** | | |
|---|---|---|
| **query** | **nr of results** | **top hit** |
| hacker | 40,500,000 | Online Hacker Simulator (GeekPrank Hacker Typer) |
| hacker AND hacker | 40,000,000 | book: the difference between Hackers and Crackers |
| hacker OR hacker | 40,600,000 | Wikipedia: Hacker |

They really do not make sense in terms of Boolean logic:

- I would have expected that the nr of results for the queries "hacker" and "hacker AND hacker" to be the same. Instead, "hacker AND hacker" has fewer results.
- I would have expected that the nr of results for the queries "hacker" and "hacker OR hacker" to be the same. Instead, in one of the search engines I get fewer results and in the other I get more!

Trying with the other suggested queries:

| **Google search** | | |
|---|---|---|
| **query** | **nr of results** | **top hit** |
| author | 4 280 000 000 | dictionary: author |
| prize | 532 000 000 | dictionary: prize |
| author OR prize | 3 030 000 000 | thebookerprizes.com |

| **Microsoft Edge (Bing) search:** | | |
|---|---|---|
| **query** | **nr of results** | **top hit** |
| author | 568,000,000 | dictionary: author |
| prize | 57,900,000 | dictionary: prize |
| author OR prize | 580,000,000 | Cambridge dictionary: author |

In the query “author OR prize”, the maximum amount of results should be the sum of the nr of results for each of the two component queries (in the extreme case that none of the documents contained both words).

Both search engines that I tested obeyed that. The nr of results for “author OR prize” was smaller than the sum of the nr of results for “author” and for “prize”.

---

8. Draw the tric for the set of terms $\mathcal{R} = \{\text{manne, manu, minna, salla, saul, sauli, vihtori}\}$.