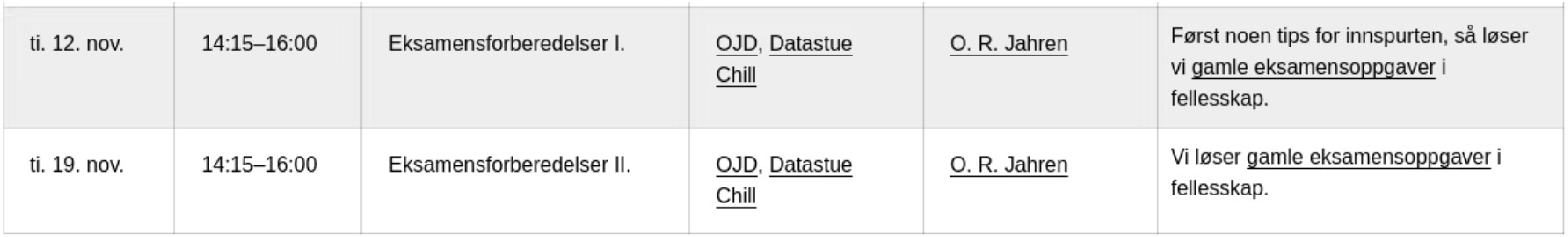
IN[34]120 - Søketeknologi 🖥️📚🔍

2024-11-12

Tema: Eksamensprep I

- Administravia
- Protips™ for eksamensprep
- Utvalgte eksamensoppgaver (8)



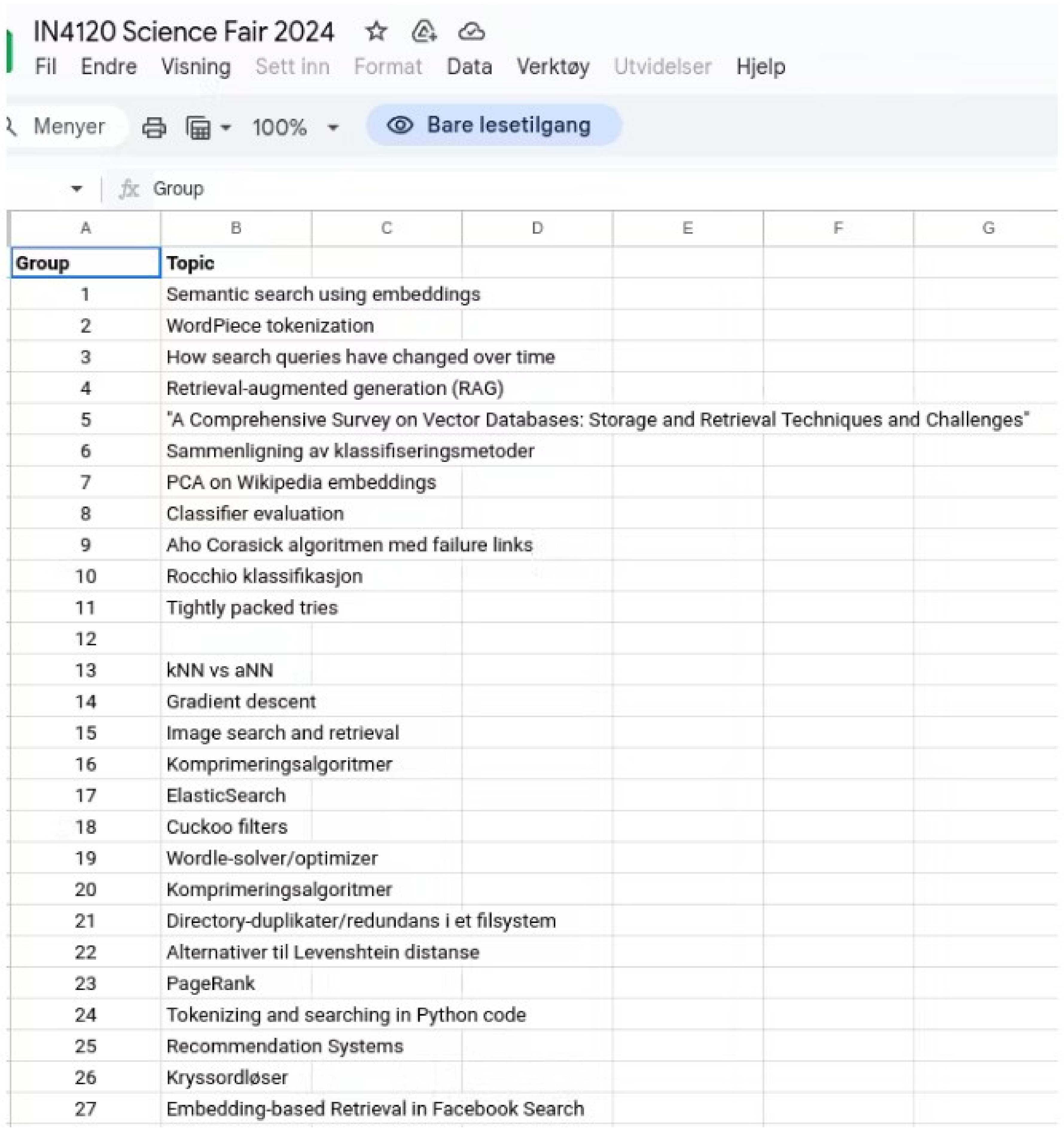
ti. 12. nov.	14:15–16:00	Eksamensforberedelser I.	OJD, Datastue Chill	O. R. Jahren	Først noen tips for innspurten, så løser vi gamle eksamensoppgaver i fellesskap.
ti. 19. nov.	14:15–16:00	Eksamensforberedelser II.	OJD, Datastue Chill	O. R. Jahren	Vi løser gamle eksamensoppgaver i fellesskap.

Eksamen fredag 29.11.

Her er vi. Nest siste gruppetime.

IN4120: Science fair 📚

- → 2024-11-18 10:15 @ KNH Lille aud
- → Nå på mandag 🙀
- → IN3120 burde se på 🤝



Group	Topic
1	Semantic search using embeddings
2	WordPiece tokenization
3	How search queries have changed over time
4	Retrieval-augmented generation (RAG)
5	"A Comprehensive Survey on Vector Databases: Storage and Retrieval Techniques and Challenges"
6	Sammenligning av klassifiseringsmetoder
7	PCA on Wikipedia embeddings
8	Classifier evaluation
9	Aho Corasick algoritmen med failure links
10	Rocchio klassifikasjon
11	Tightly packed tries
12	
13	kNN vs aNN
14	Gradient descent
15	Image search and retrieval
16	Komprimeringsalgoritmer
17	ElasticSearch
18	Cuckoo filters
19	Wordle-solver/optimizer
20	Komprimeringsalgoritmer
21	Directory-duplikater/redundans i et filsystem
22	Alternativer til Levenshtein distanse
23	PageRank
24	Tokenizing and searching in Python code
25	Recommendation Systems
26	Kryssordløser
27	Embedding-based Retrieval in Facebook Search

Temaer for SF

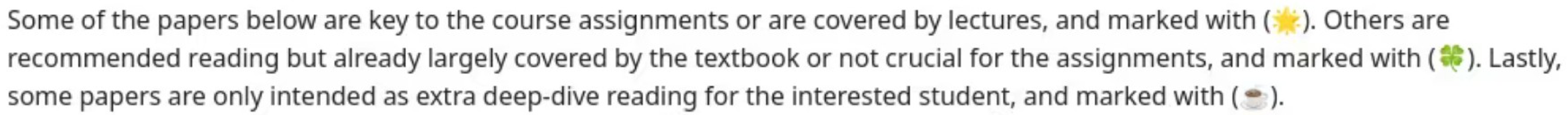
Assignment E

- Frist på fredag ☀️
- Siste oblig👏📈📈📈💯👏
- Rettes nu. Klart innen science fair (?)

Tips til eksamensprep 📚🕊️

Papers 🏮🏮

- Få oversikt over *poengene*
- Ikke finles alt
- Prioriter utifra emoji-skalaen(™)



Some of the papers below are key to the course assignments or are covered by lectures, and marked with (🌟). Others are recommended reading but already largely covered by the textbook or not crucial for the assignments, and marked with (🍀). Lastly, some papers are only intended as extra deep-dive reading for the interested student, and marked with (☕).

Emoji-skalaen.

Emoji-skalaen forklart (min tolking🤓)

→ 🌟: Burde ha god oversikt

→ 🍀: Burde vite hva de er om

→ 🍵: Ikke så farlig, mest for gøy

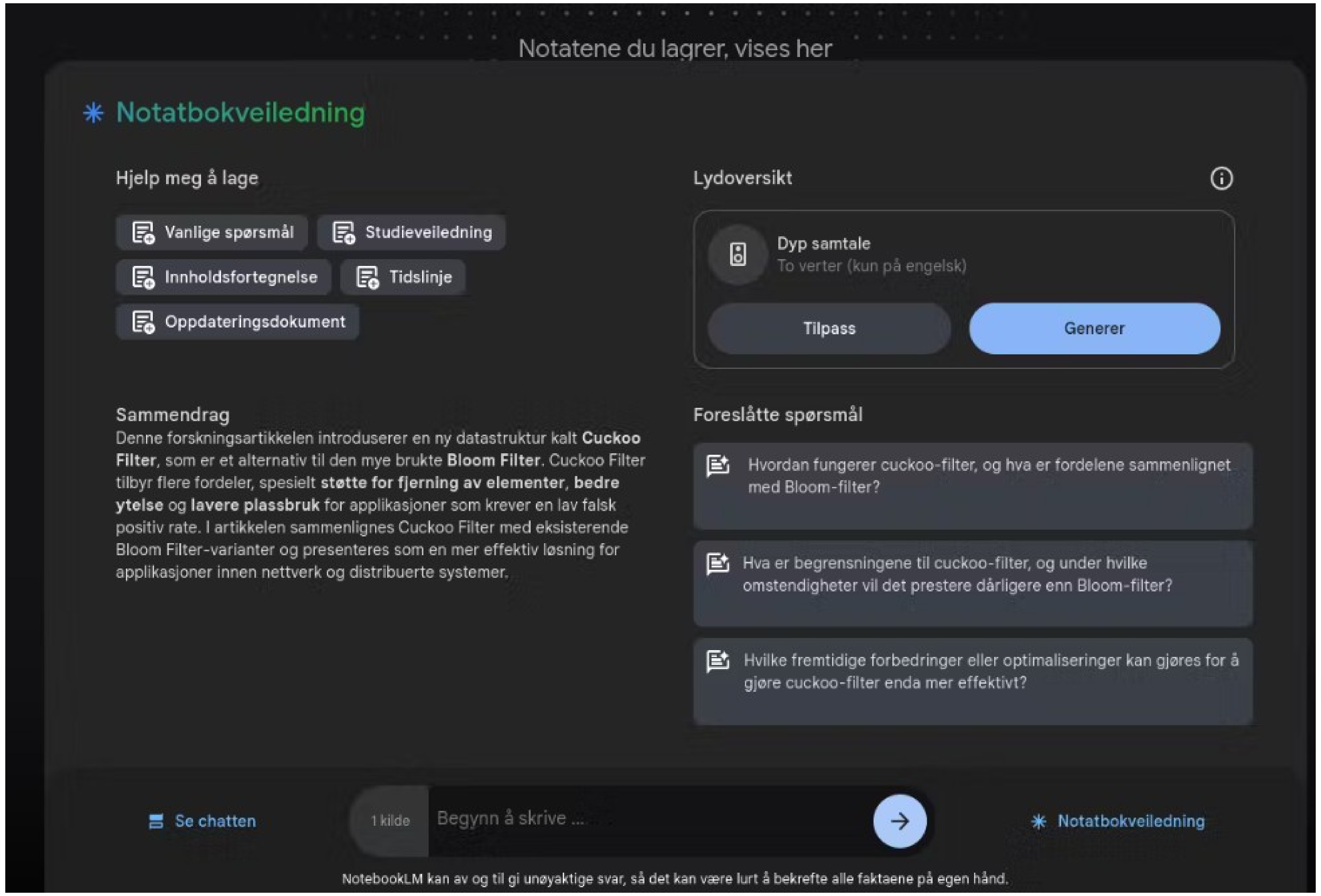
Boka 📖

- Viktig å lese.
- *Hele* er ikke pensum
- Se timeplanen for kapitler
- https://nlp.stanford.edu/IR-book/

Tips til studieverktøy

Ikke pensum: NotebookLM 💻📖

Atter et språkteknologisk bidrag til verden


Notatene du lagrer, vises her

Notatbokveiledning

Hjelp meg å lage

Lydoversikt

Dyp samtale
To verter (kun på engelsk)

Sammendrag
Denne forskningsartikkelen introduserer en ny datastruktur kalt **Cuckoo Filter**, som er et alternativ til den mye brukte **Bloom Filter**. Cuckoo Filter tilbyr flere fordeler, spesielt **støtte for fjerning av elementer**, **bedre ytelse** og **lavere plassbruk** for applikasjoner som krever en lav falsk positiv rate. I artikkelen sammenlignes Cuckoo Filter med eksisterende Bloom Filter-varianter og presenteres som en mer effektiv løsning for applikasjoner innen nettverk og distribuerte systemer.

Foreslåtte spørsmål

Hvordan fungerer cuckoo-filter, og hva er fordelene sammenlignet med Bloom-filter?

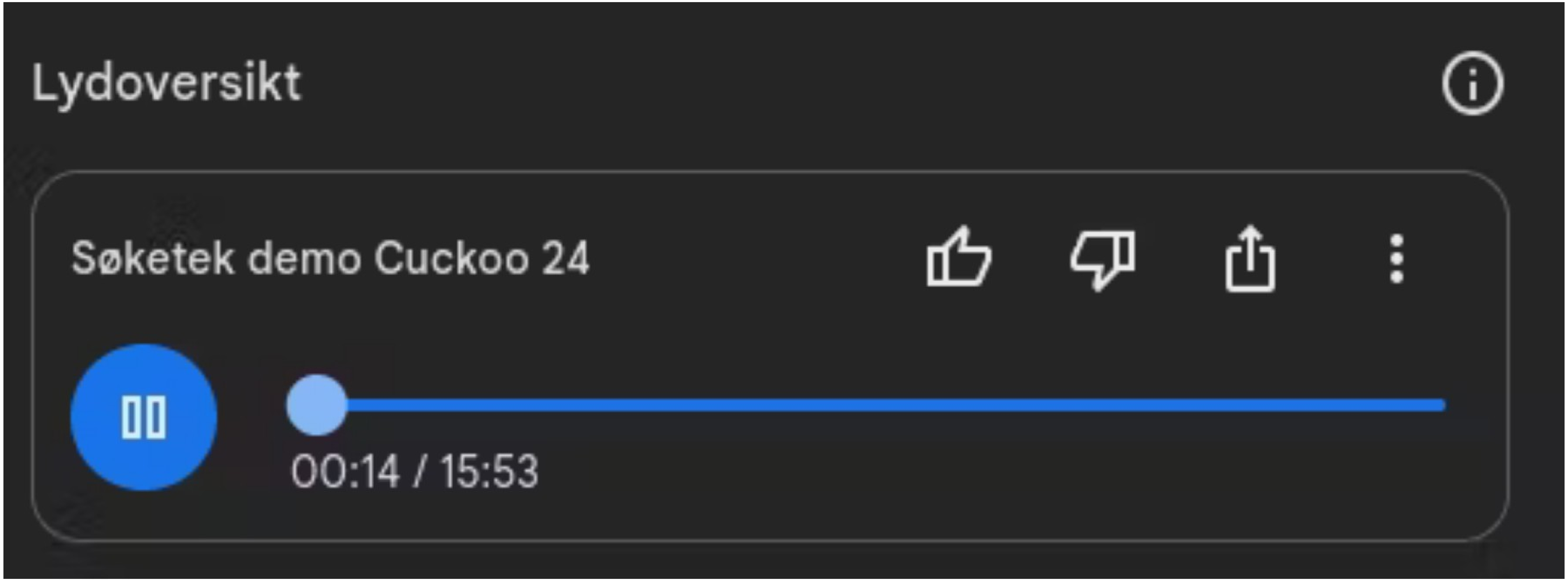
Hva er begrensningene til cuckoo-filter, og under hvilke omstendigheter vil det prestere dårligere enn Bloom-filter?

Hvilke fremtidige forbedringer eller optimaliseringer kan gjøres for å gjøre cuckoo-filter enda mer effektivt?

NotebookLM kan av og til gi unøyaktige svar, så det kan være lurt å bekrefte alle faktaene på egen hånd.

RAG!

https://notebooklm.google.com/



Får podcast om paperet(!)

Andre ressurser 🐈

- Truls sine notater fra ifjor:
- seminars/gruppe1/notater-eksamen
- Forelesningsopptak
- Wikipedia 🤫🤫

Gamle eksamensoppgaver

Finnes her: https://github.com/aohrn/in3120-2024/tree/main/exams

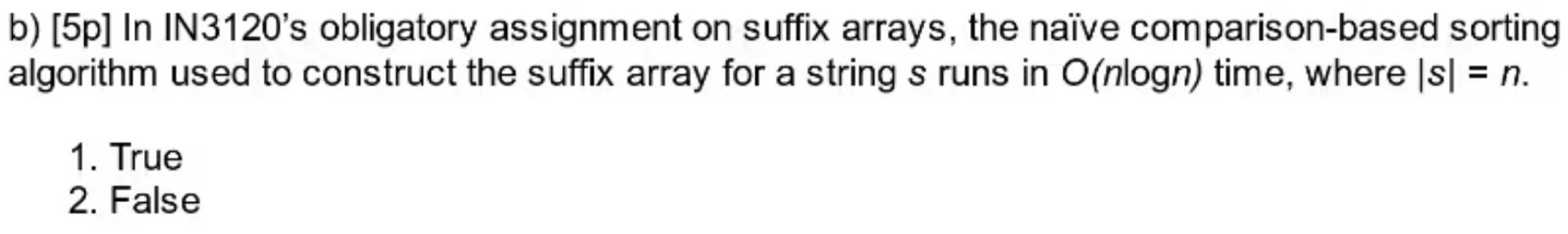
Om eksamensoppgaver

- → Begynn med de nyeste
- → Husk covid (hjelpemidler 20+21)
- → De som heter "LF" er løsningsforslag
- → Noen spørsmål rebrandes

Format på arbeidet 👨‍🏫

1. spørsmål vises på tavla
2. diskuter med gruppe(4ish) 5-10 min
3. oppsummerer spørsmålet i fellesskap
4. gjenta

Har alle gruppe?

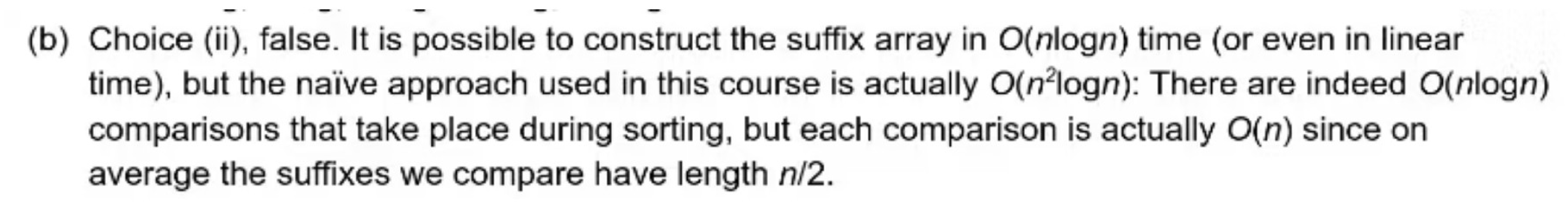


b) [5p] In IN3120's obligatory assignment on suffix arrays, the naïve comparison-based sorting algorithm used to construct the suffix array for a string *s* runs in *O(n*log*n)* time, where |*s*| = *n*.

1. True
2. False

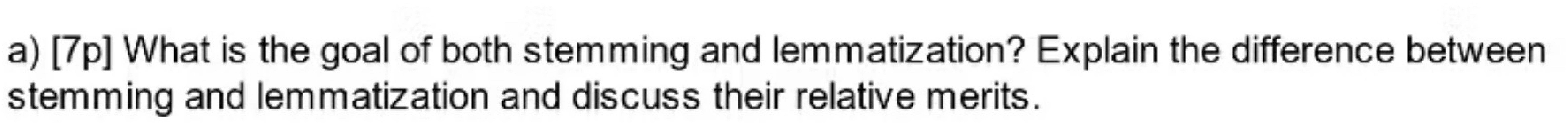
2022

Oppgave 1. Fra oblig B-1.

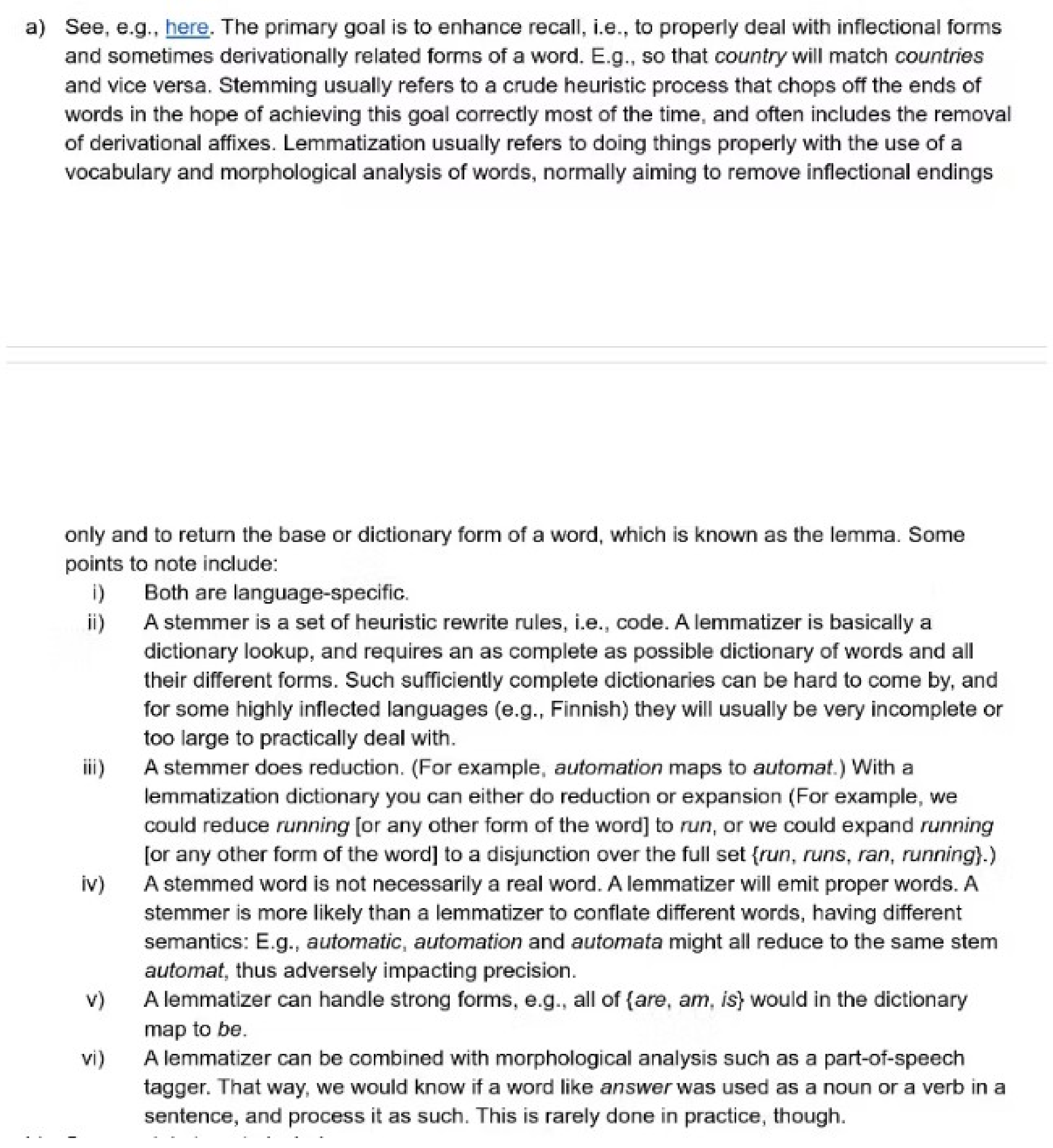


(b) Choice (ii), false. It is possible to construct the suffix array in $O(n\log n)$ time (or even in linear time), but the naïve approach used in this course is actually $O(n^2\log n)$: There are indeed $O(n\log n)$ comparisons that take place during sorting, but each comparison is actually $O(n)$ since on average the suffixes we compare have length $n/2$.

Oppgave 1 LF: False! Hvorfor?


a) [7p] What is the goal of both stemming and lemmatization? Explain the difference between stemming and lemmatization and discuss their relative merits.

Oppgave 2.



a) See, e.g., here. The primary goal is to enhance recall, i.e., to properly deal with inflectional forms and sometimes derivationally related forms of a word. E.g., so that *country* will match *countries* and vice versa. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings

only and to return the base or dictionary form of a word, which is known as the lemma. Some points to note include:

i) Both are language-specific.

ii) A stemmer is a set of heuristic rewrite rules, i.e., code. A lemmatizer is basically a dictionary lookup, and requires an as complete as possible dictionary of words and all their different forms. Such sufficiently complete dictionaries can be hard to come by, and for some highly inflected languages (e.g., Finnish) they will usually be very incomplete or too large to practically deal with.

iii) A stemmer does reduction. (For example, *automation* maps to *automat*.) With a lemmatization dictionary you can either do reduction or expansion (For example, we could reduce *running* [or any other form of the word] to *run*, or we could expand *running* [or any other form of the word] to a disjunction over the full set {*run*, *runs*, *ran*, *running*}.)

iv) A stemmed word is not necessarily a real word. A lemmatizer will emit proper words. A stemmer is more likely than a lemmatizer to conflate different words, having different semantics: E.g., *automatic*, *automation* and *automata* might all reduce to the same stem *automat*, thus adversely impacting precision.

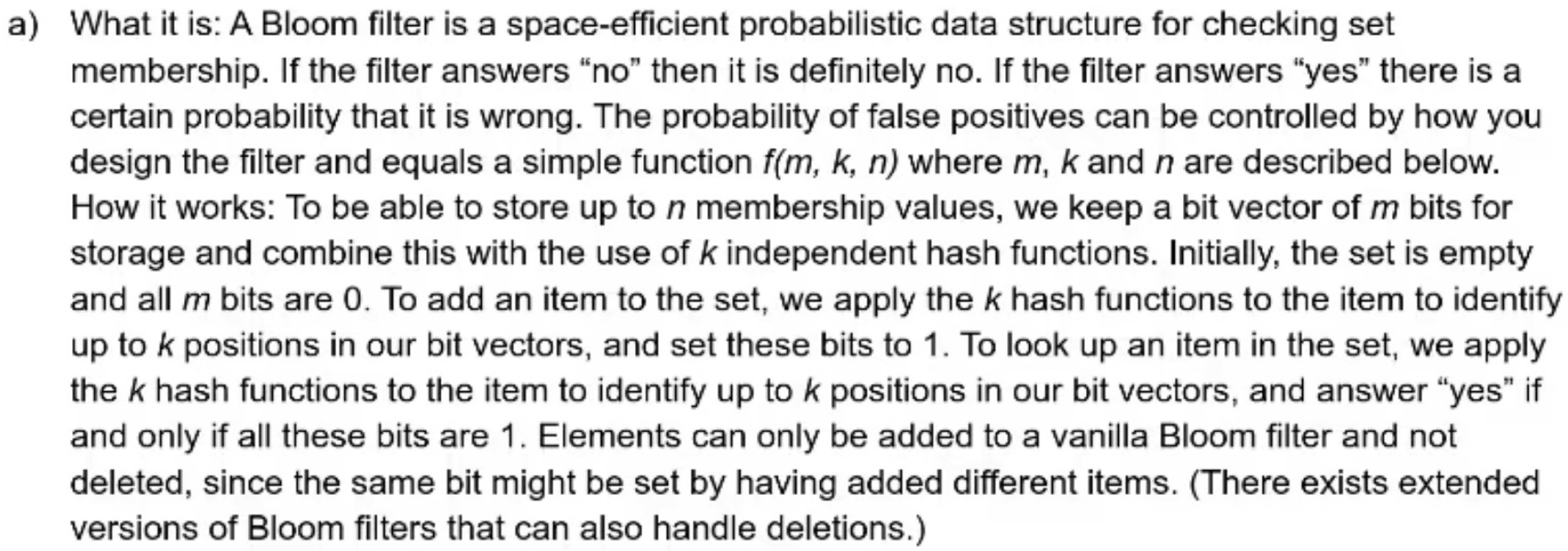
v) A lemmatizer can handle strong forms, e.g., all of {*are*, *am*, *is*} would in the dictionary map to *be*.

vi) A lemmatizer can be combined with morphological analysis such as a part-of-speech tagger. That way, we would know if a word like *answer* was used as a noun or a verb in a sentence, and process it as such. This is rarely done in practice, though.

Oppgave 2 LF

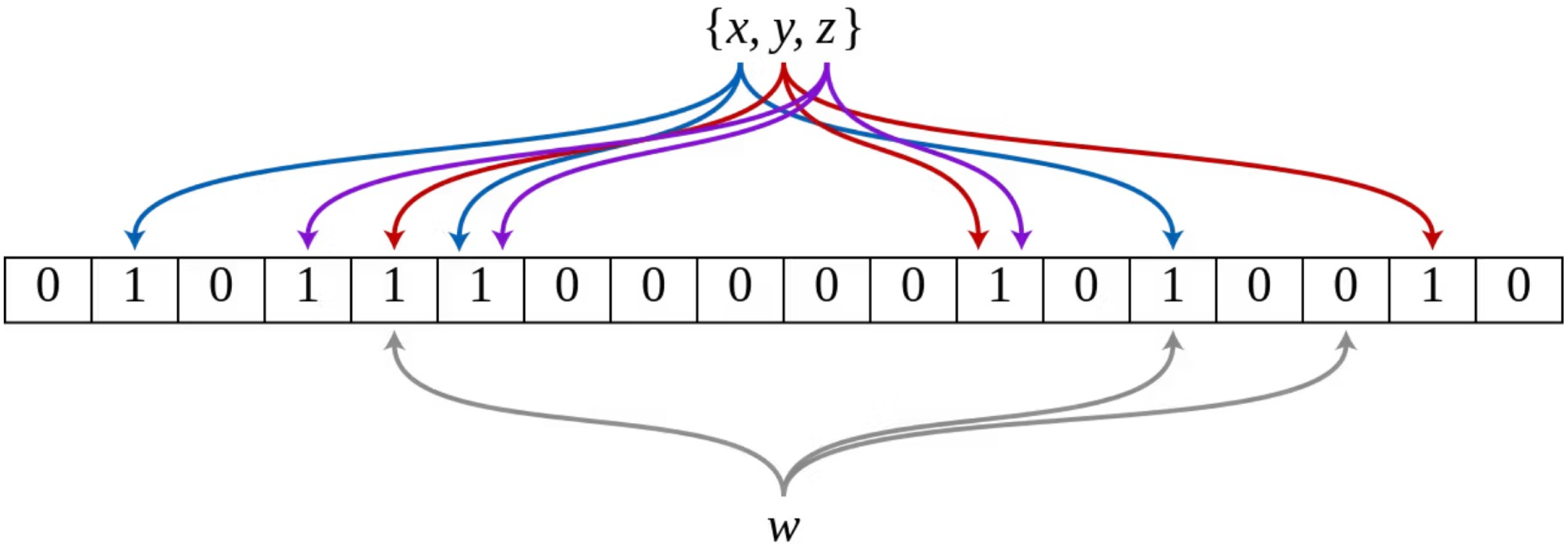
a) [5p] Describe what a Bloom filter is and how it works.

Oppgave 3



a) What it is: A Bloom filter is a space-efficient probabilistic data structure for checking set membership. If the filter answers “no” then it is definitely no. If the filter answers “yes” there is a certain probability that it is wrong. The probability of false positives can be controlled by how you design the filter and equals a simple function *f(m, k, n)* where *m*, *k* and *n* are described below. How it works: To be able to store up to *n* membership values, we keep a bit vector of *m* bits for storage and combine this with the use of *k* independent hash functions. Initially, the set is empty and all *m* bits are 0. To add an item to the set, we apply the *k* hash functions to the item to identify up to *k* positions in our bit vectors, and set these bits to 1. To look up an item in the set, we apply the *k* hash functions to the item to identify up to *k* positions in our bit vectors, and answer “yes” if and only if all these bits are 1. Elements can only be added to a vanilla Bloom filter and not deleted, since the same bit might be set by having added different items. (There exists extended versions of Bloom filters that can also handle deletions.)

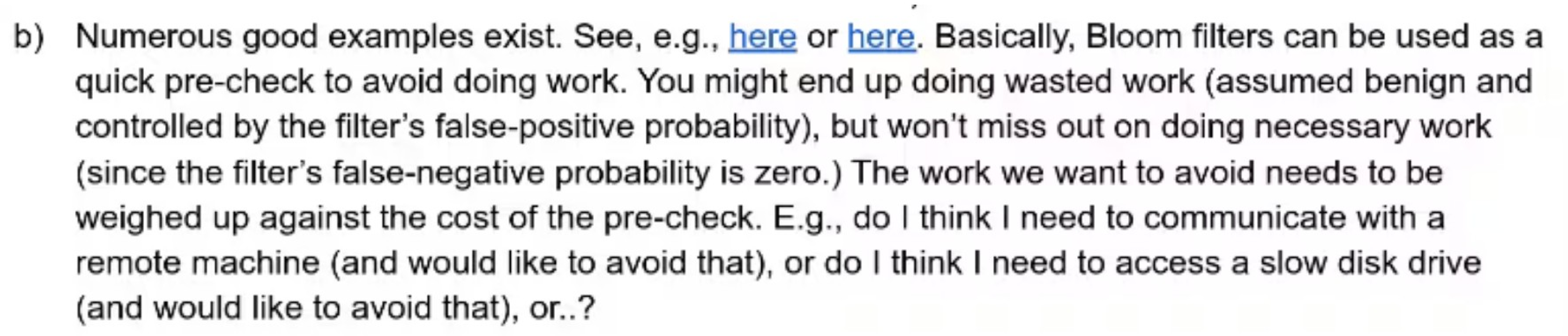
Oppgave 3 LF



Oppgave 3 LF

b) [4p] Provide an example of where using a Bloom filter can improve a search engine's performance.

Oppgave 4

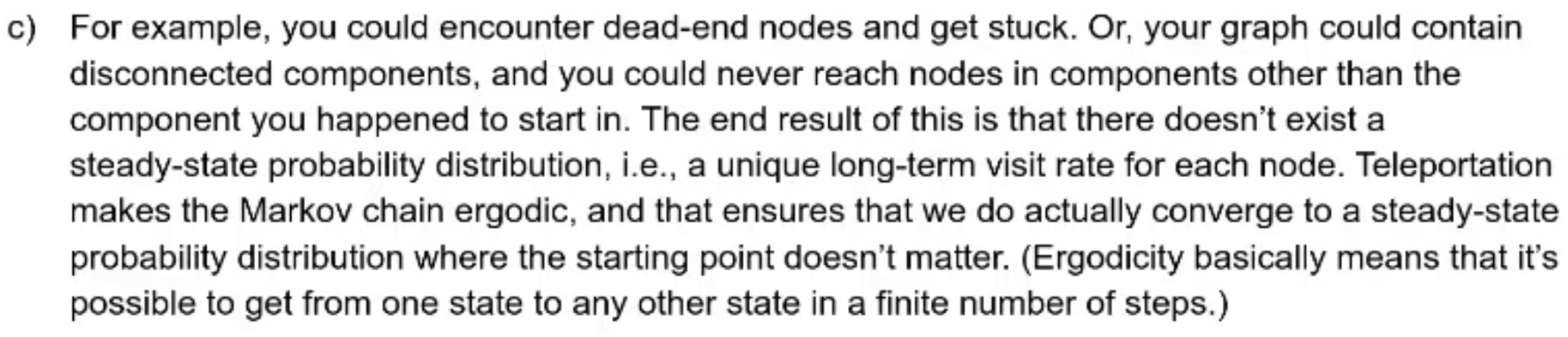


b) Numerous good examples exist. See, e.g., here or here. Basically, Bloom filters can be used as a quick pre-check to avoid doing work. You might end up doing wasted work (assumed benign and controlled by the filter's false-positive probability), but won't miss out on doing necessary work (since the filter's false-negative probability is zero.) The work we want to avoid needs to be weighed up against the cost of the pre-check. E.g., do I think I need to communicate with a remote machine (and would like to avoid that), or do I think I need to access a slow disk drive (and would like to avoid that), or..?

Oppgave 4 LF

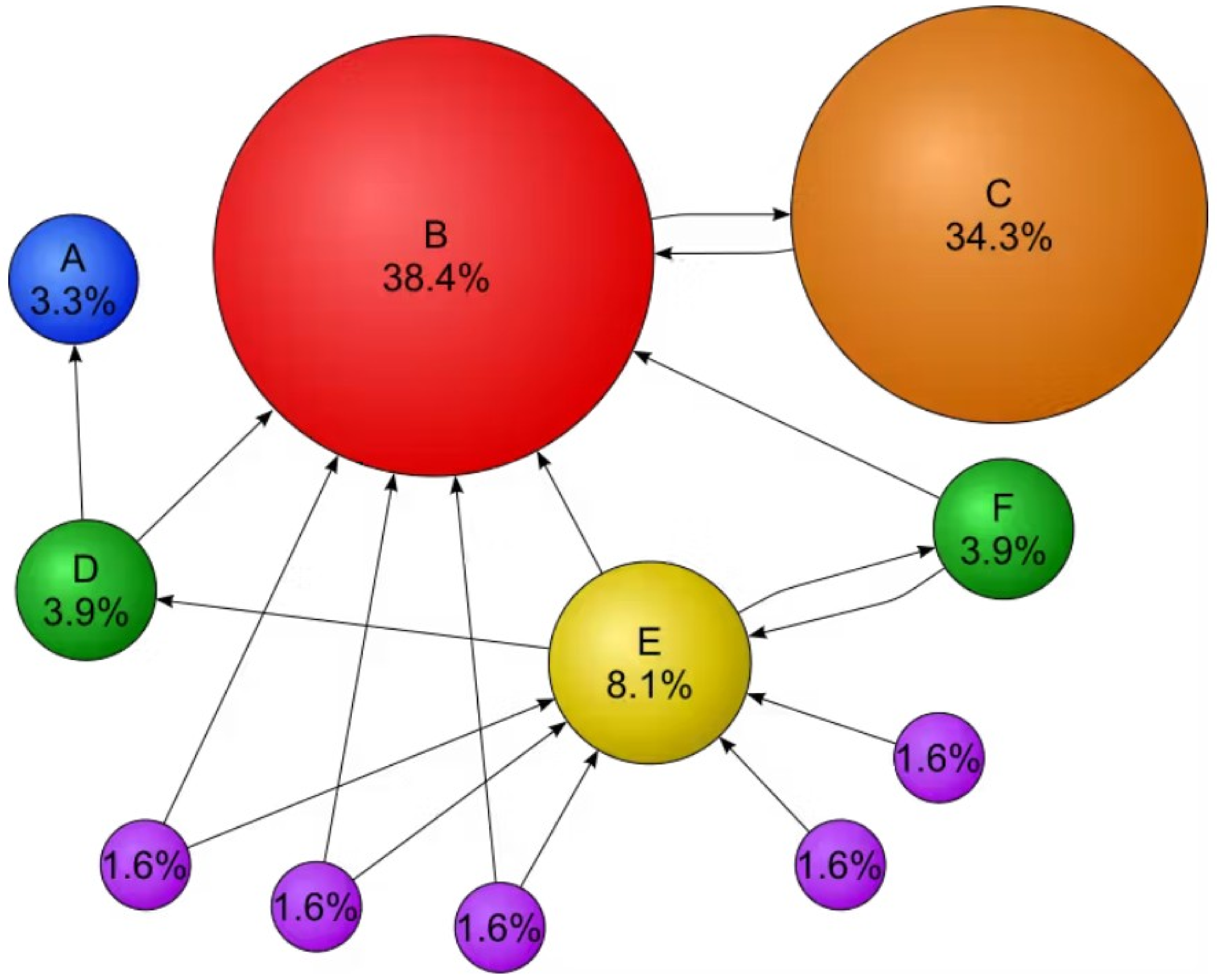
c) [4p] In the random surfer model the surfer is allowed to teleport with some probability greater than zero. If you assume that teleportation is not allowed, discuss some of the problems that can arise and their implications.

Oppgave 5



c) For example, you could encounter dead-end nodes and get stuck. Or, your graph could contain disconnected components, and you could never reach nodes in components other than the component you happened to start in. The end result of this is that there doesn't exist a steady-state probability distribution, i.e., a unique long-term visit rate for each node. Teleportation makes the Markov chain ergodic, and that ensures that we do actually converge to a steady-state probability distribution where the starting point doesn't matter. (Ergodicity basically means that it's possible to get from one state to any other state in a finite number of steps.)

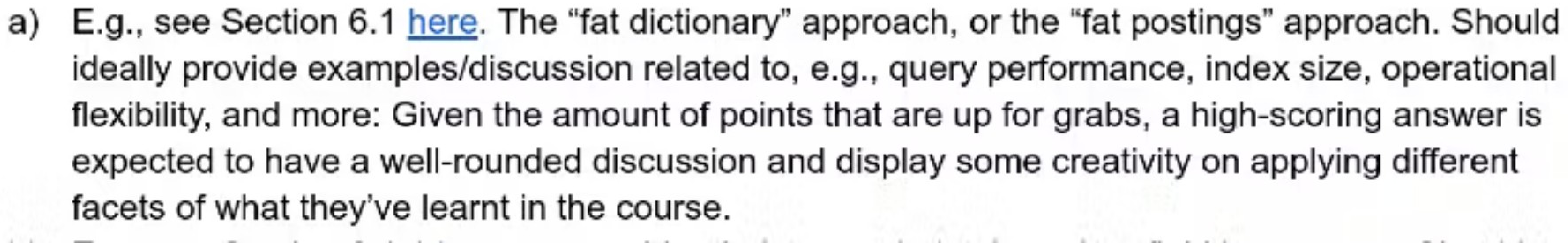
Oppgave 5 LF



Oppgave 5 LF

a) [10p] Using an inverted index, discuss at least two ways to support fielded search. What are their advantages and drawbacks?

Oppgave 6

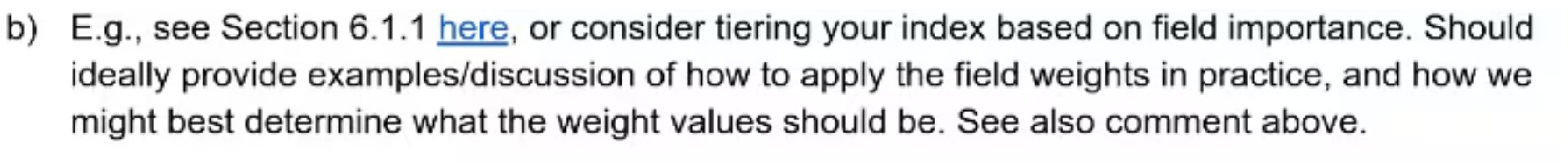


a) E.g., see Section 6.1 here. The “fat dictionary” approach, or the “fat postings” approach. Should ideally provide examples/discussion related to, e.g., query performance, index size, operational flexibility, and more: Given the amount of points that are up for grabs, a high-scoring answer is expected to have a well-rounded discussion and display some creativity on applying different facets of what they’ve learnt in the course.

Oppgave 6 LF

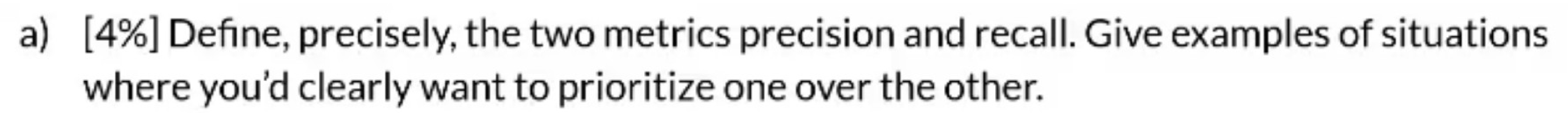
b) [10p] Some fields might intuitively be more important than others. For example, if your query matches the content of the *title* field for a document that might be better than a match in the *footnotes* field. Discuss how you could design a relevance function that takes this into account.

Oppgave 7



b) E.g., see Section 6.1.1 here, or consider tiering your index based on field importance. Should ideally provide examples/discussion of how to apply the field weights in practice, and how we might best determine what the weight values should be. See also comment above.

Oppgave 7 LF

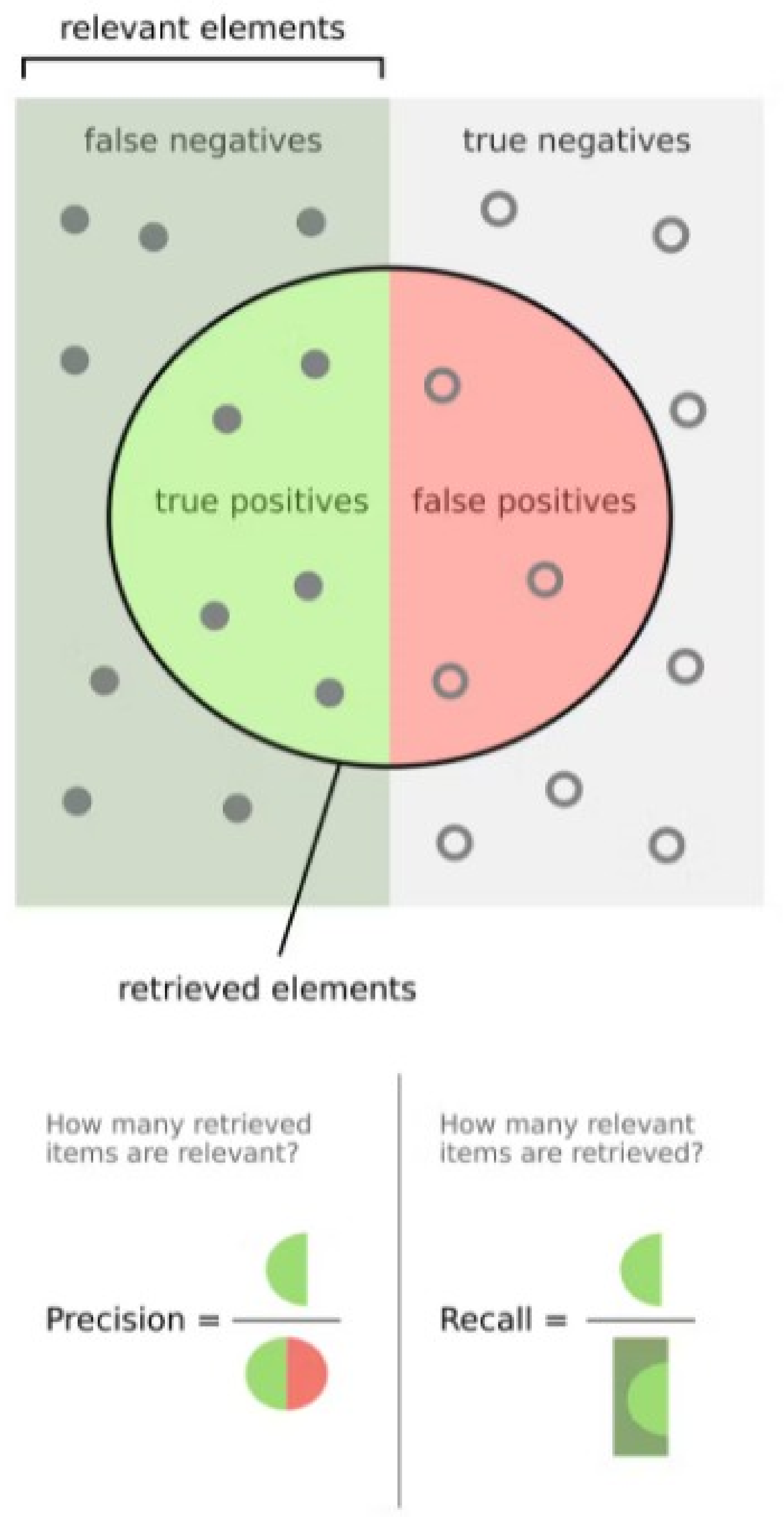

a) [4%] Define, precisely, the two metrics precision and recall. Give examples of situations where you'd clearly want to prioritize one over the other.

Oppgave 8

a) Assuming binary relevance, let RET and REL denote the sets of retrieved and relevant documents, respectively. Precision is defined as the ratio |RET∩REL| / |RET|, i.e., the number of relevant documents that were retrieved divided by the total number of documents retrieved. Recall is defined as the ratio |RET∩REL| / |REL|, i.e., the number of relevant documents that were retrieved divided by the total number of relevant documents. A situation where you'd prioritize recall is if there's a big cost of missing out on a relevant document, e.g., imagine a lawyer doing research and preparing for a case that requires that he has a complete overview of all relevant case law. A situation where you'd prioritize precision is if the presence of an irrelevant document has a big cost, e.g., imagine

that there is only a single relevant document to your query and that any irrelevant document thus constitutes "noise".

Oppgave 8 LF


relevant elements
false negatives
true negatives
true positives
false positives
retrieved elements
How many retrieved items are relevant?
Precision =
How many relevant items are retrieved?
Recall =

Oppgave 8 LF

Neste gang:

- Mer av dette.
- ^men innspill mottas!!
- Husk science fair!

Assignment aid until 16

Assignment workshop. Write something here and we'll discuss it towards the end.

Break until 15:15 😄