Knowledge Representation Approaches for Educational Question Generation

Anonymous ACL submission

Abstract

Teachers are increasingly using prompted LLMs to generate exam questions, and students can use generated questions for self-assessment. When generating questions from a given educational text—rather than relying solely on the LLM's internal knowledge—handling long textual content, such as a textbook spanning hundreds of pages, presents a challenge. In this paper, we experiment with three knowledge representation approaches tailored for educational question generation using LLMs. As a novel contribution among these alternatives, we adapt the atomic fact decomposition method from fact-checking research to the educational domain. We manually evaluate the generated questions based on various criteria. Our empirical results indicate that a list of atomic facts provides a better foundation for question generation than long plain text and that LLM-based question generation from Knowledge Graph triplets outperforms rule-based question generation from Knowledge Graphs.

1 Introduction

The automatic generation of educational questions (EQG) will play a key role in scaling education (Baidoo-Anu and Ansah, 2023). It can significantly decrease the workload of teachers along with enabling student self-assessment at scale in a personalized way. Large language models (LLMs) have demonstrated great performance in various tasks and applications. Teachers also employ LLMs to generate assessment questions for their exams by prompting general models (Walter, 2024).

In this paper, we focus on EQG given textual educational material (textbooks, lecture slides, transcripts, etc.). Questions are generated faithfully to the educational material and not from the latent knowledge of pre-trained general LLMs. This use case is important in rapidly evolving fields, as it can be applied even to web content like blog posts. Onboarding processes in organizations can also exploit automated assessments against their own document base. Lastly, a teacher might want to teach special topics that should be included in assessment questions. We note that this approach seems to be very similar to RAG, but in EQG there is no query, thus instead of the retriever step it requires special techniques to gather relevant contexts from the educational material for the generation step.

Although LLMs have great potential in EQG, their use is not unproblematic, as they suffer from hallucinations and misinformation. Chen et al. (2024) and our preliminary experiments also report that LLMs can generate close to perfect simple questions. Simple questions are remember-type questions (according to the Bloom's Taxonomy of educational science (Anderson et al., 2001)) which can be generated from a short chunk of text as relevant content. Increasing the complexity level of questions requires the LLM to get to a deeper semantic understanding of the text; thus, it leads to hallucinations. Similarly EQG from long contents, for instance a textbook, also introduces more mistakes as distant relationships in text are more difficult to detect in LLMs. In this paper, we propose four methods for generating multiple-choice questions (MCQs) from multiple chapters of textbooks. These types of items are still remember-type questions, but the generation of good-quality questions and answer options requires comprehensive memorization of all of the educational material.

There are various opportunities to represent the knowledge of a larger educational material that provides the knowledge context for the generation step of the LLM. Knowledge Graphs (KGs) are structured representation formats that represent knowledge in entity-relation-entity triplets and can store unlimited amounts of information. The key practical disadvantage of KGs is that they are rigid, and building a KG from text is still not precise enough with moderate coverage (despite the vast amount

of research on KG extraction (Kertkeidkachorn and Ichise, 2017; Koncel-Kedziorski et al., 2019; Melnyk et al., 2022)). On the other end, we can consider the original long-form texts as a knowledge representation format, having perfect precision and coverage. The problem is that LLMs are not accurate enough to understand deeper and long-distance semantic/pragmatic relationships in long texts. We propose to use a solution that lies between KGs and long texts. Inspired by fact-checking research (Min et al., 2023; Chung et al., 2025), we extract atomic facts from the original text and use the set of atomic facts as the knowledge representation format for EQG. The definition of atomic fact is EQG-driven, it can be any simple and clear statement, i.e. whose knowledge can be assessed. Our assumption is that the set of atomic facts, which are simple and clear sentences, are considerably easier to understand by LLMs than long texts and provide a less rigid and better coverage representation than KGs could.

We introduce four EQG methods grounded on the three knowledge representation formats (two methods for the KGs). Each of the methods employs GPT-4o (Hurst et al., 2024) with few-shot prompting. We carried out comparative experiments on the methods and generated compare-type MCQs for sorting algorithms. The generated MCQs were carefully evaluated by human experts (teachers of algorithms subject) according to seven evaluation metrics. The evaluation metrics were designed for real-world usability, i.e. whether they are usable by a teacher or for student self-assessment. The metrics cover areas from factual correctness through clarity to creativity and engagement. The conclusion of these experiments is that there is no clear winner, but there is a considerable tradeoff between question correctness.

In summary, our contributions are threefold in this paper:

- We adapt the atomic fact text decomposition method to the EQG task.
- We propose and comperatively evaluate four methods for educational MCQ generation based on three different types of knowledge representation.
- Our human evaluation methodology for generated MCQs consists of various aspects that are important in their real-world usability, including factuality, clarity, creativity, and engagement metrics.

2 Related Work

Multiple-choice question generation Multiple-choice question (MCQ) generation has been extensively studied in educational technology.

Early research focused on template-based or rule-based techniques. Papasalouros et al. (2008) developed algorithms for MCQ generation based on domain ontologies, introducing eleven strategies. Of these eleven, Strategy 6 and Strategy 7 are very similar to our rule-based KG approach.

LLMs have been disrupting the EQG field in the last two years. For instance, Maity et al. (2024) introduced multi-stage prompting with GPT models, showing improved performance across multiple languages. Scaria et al. (2024) reported that LLMs are able to generate MCQs keeping the Bloom's Taxonomy levels, while Yao et al. (2024) and Wang et al. (2024) developed a self-refine framework for professional exam questions using iterative self-critique.

Recent work demonstrates, that hybrid methods are still relevant. Kumar et al. (2024) combined ontology-based and machine learning techniques for MCQ generation. Their approach also specifically addresses generating questions for different cognitive levels.

Prior EQG work focuses on generating questions either from localized contexts or without any grounding in a zero-shot manner. Our work addresses the challenge of generating questions that require information scattered across multiple chapters or sections of educational material.

Atomic fact extraction Atomic fact extraction is a recently popularized technique mainly used in claim-verification. Min et al. (2023) argues that complex claims could contain both valid and invalid information, hence the need for decomposing these complex sentences into statements that each contain a single piece of information, that is unambiguous in their factuality.

Min et al. (2023) proposes a complete framework for evaluating LLM-generated text by breaking it down to atomic statements and estimating the ratio of such statements that were supported by a trusted source. Chung et al. (2025) presents a complex system, leveraging a similar approach in the medical domain to verify LLM-generated text based on the patient's medical history. Wanner et al. (2024) compares different methods for breaking down claims into atomic facts and shows that the decomposition method significantly affects

downstream results on factual precision measurements, such as FACTSCORE (Min et al., 2023).

Atomic fact extraction is useful in other domains besides claim verification. Chen et al. (2023) found that breaking down complex sentences into self-contained factual statements, which they refer to as "propositions" significantly improves retrieval performance in a dense retrieval setting. They work with various open-domain QA datasets and English Wikipedia text, presenting FACTOIDWIKI, English Wikipedia broken down into various levels of granularity (passage, sentence, and proposition). Kamoi et al. (2023) uses GPT-3.5 to automatically decompose Wikipedia claims into subclaims, showing improved performance on entailment tasks across multiple datasets.

To the best of our knowledge, we are the first to employ atomic facts as educational knowledge representation. Our definition of atomic facts is slightly different from the claim-checking definition as we focus on small self-contained knowledge statements that are suitable for evaluation. The educational use case requires that the atomic facts be both verifiable and pedagogically relevant.

3 Question Generation Approaches

3.1 MCQ for Comparing Skills

We automatically generate MCQs, which is a common assignment format in education, as they can test various levels of complexity and cognition of students (Masters et al., 2001; Brady, 2005; Scaria et al., 2024). We address monodisciplinary EQG (Chen et al., 2024) and in this study, we experiment with comparison questions for sorting algorithms. To answer these questions, the test-taker has to remember the concepts in algorithm studies (e.g. time complexity), concepts that are specific to sorting algorithms (e.g. stable sorting), and has to remember the sorting algorithms' properties. According to Bloom's Taxonomy (Anderson et al., 2001), these questions belong to the remember level, as these MCQs can be answered with perfect remembering, while comprehensive understanding is not necessary.

On the other hand, remembering many pieces of information is necessary to answer the comparison-type MCQs, and this information is scattered in the educational material, requiring the test-taker to keep in mind dozens of pages. The key difference between this paper and related EQG work (Chen et al., 2024; Elkins et al., 2023) is that we generate questions from very long texts of educational material and a single MCQ can ask for facts mentioned in the texts far from each other. Precise and full coverage understanding of long texts is still a challenge for LLMs. We describe three knowledge representation forms for EQG, designed to overcome this issue.

3.2 Knowledge Representaion Alternatives

We explored three approaches to represent the educational knowledge. A comparative example of the alternatives is presented in Figure 1.

Plain Text As LLMs have the ability to make sense of plain textual information, the first natural choice was to represent the knowledge as text. Our first approach utilized text descriptions, sourced from Wikipedia entries. In educational deployment, this could be extended to textbooks, lecture notes, and other trusted materials. This format maintains natural language context but requires LLMs to identify and extract relevant information.

Knowledge Graph KGs offer a structured representation through entities and relationships. The structure of KGs makes programmatic manipulation possible while meaningful edge types and concept nodes could make them interpretable for humans. A large number of KGs are available, and constructing new ones for specialized domains is also manageable.

It is important to note that KGs and knowledge spaces (Doignon and Falmagne, 1985, 2015), which are often used in the education literature, are separate approaches to organizing educational content. While knowledge spaces provide manageable chunks for direct instruction and assessment and focus on prerequisites, KGs are a general representation of data, making use of concepts and the relationships between them.

Factual Statements This approach transforms natural text into atomic fact statements, where each statement captures a single, self-contained piece of information.

Each factual statement must satisfy three criteria:

1. Non-triviality: The statement should convey meaningful domain knowledge
2. Self-containment: The statement should be comprehensible without additional context
3. Verifiability: The statement should be clearly true or false within the domain

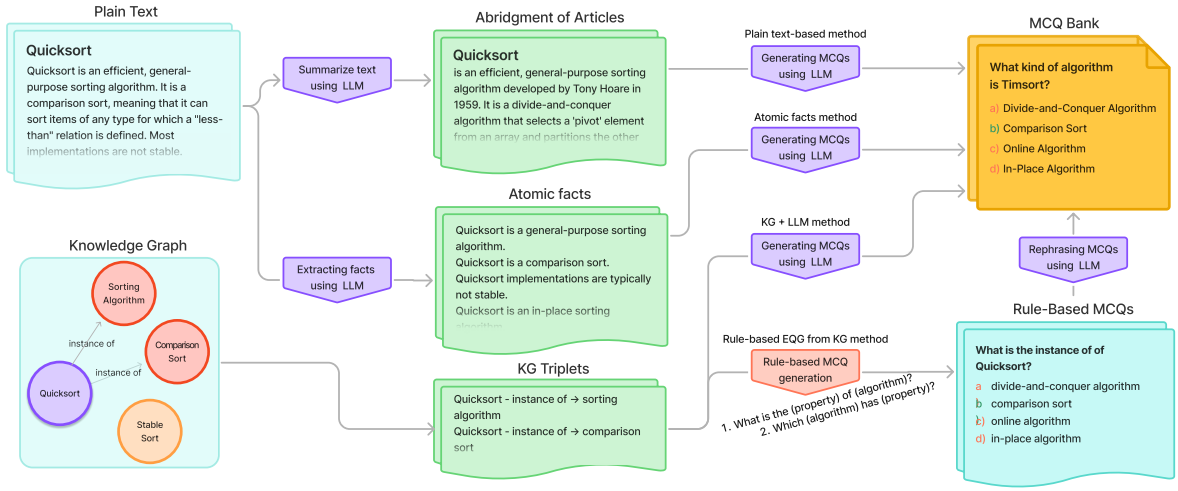

Figure 1: Overview of four EQG methods over three knowledge representation forms.

This representation combines the accessibility of natural language with some of the structural benefits of KGs, as statements often follow implicit subject-predicate-object patterns.

3.3 MCQ Generation Methods

Building upon the previously described knowledge representations, we developed four distinct methods for generating multiple-choice questions, see Figure 1 for an overview. Each method uses LLMs and we engineered the EQG prompts to be as similar to each other as possible.

Plain Text-Based Method A straightforward approach would be to give the entire educational material to the LLM to generate questions. However, due to the limited context window of LLMs, this approach quickly becomes infeasible with large volumes of content. Splitting the data is not an option either, as it would prevent generating questions that consider the entire document. To address these challenges, we partitioned the source material into manageable segments (e.g. textbook chapters) and first applied a summarization process to each segment individually. This process removes less important details while still allowing the LLM to grasp the big picture. The resulting summaries were then used as input for the few-shot question generation prompt.

Rule-based EQG from Knowledge Graphs To exploit the structured information inherent in KGs, we created a rule-based question generation methodology that utilizes predefined templates corresponding to the graph's nodes and edges. Two primary templates were constructed. In the first template, the inquiry focuses on identifying which algorithm exhibits a specified property. Let knowledge graph $G = \{(h, r, t)|h, t \in V, r \in R\}$ is a set of triplets, where V is the set nodes and R is the set of relation types. Let $p \in V$ a selected property and $r \in R$ a selected relation type, and we have four sorting algorithms $s_a \in S$ the answer and $s_{1-3} \in S$ distractors where $S \subset V$.

Which algorithm has $\{r\}$ of $\{p\}$?
- s_a, where $(s_a, r, p) \in G$
- s_1, where $(s_1, r, p) \notin G$
- s_2, where $(s_2, r, p) \notin G$
- s_3, where $(s_3, r, p) \notin G$

In the second template, the focus is on a specific algorithm $s \in S$ by inquiring about one of its properties. In this case, the question is formulated as:

What is the $\{r\}$ of $\{s\}$?
- p_a, where $(s, r, p_a) \in G$
- p_1, where $(s, r, p_1) \notin G$
- p_2, where $(s, r, p_2) \notin G$
- p_3, where $(s, r, p_3) \notin G$

Where $p_a \in V$ the answer and the $p_{1-3} \in V$ distractors.

Because the generated questions initially relied exclusively on the formal structure of the KG, this approach led to many unnatural questions for example, *"Which algorithm has instance of of adaptive sort?"*. We prompt an LLM to rephrase these questions to make them more understandable for students.

Knowledge Graph + LLM In this method, instead of relying on predefined templates, we use the KG – or a selected subset thereof – as the context of an LLM. The KG is represented in a list of textual (h, r, s) triplets. The LLM few-shot prompted for MCQ generation based on this context can infer the relationships necessary for coherent question generation.

Atomic Facts The last method employs atomic facts as context. These factual statements have a structure that closely resembles how knowledge is organized in a KG, while it is more flexible as there is no ontology schema. Atomic facts are more concise than presenting the same information as lengthy paragraphs of plain text. The list of atomic facts provides the context for the LLM employing a few-shot prompt very similarly to the previous method.

4 Experimental Results

4.1 Datasets

Our goal is to generate MCQs that require having information from different parts of educational materials, for instance, connecting concepts from different chapters of a textbook. Let $S =$ *Merge sort*, *Selection sort*, *Heapsort*, *Timsort*, *Quicksort*, *Insertion sort*, *Bubble sort* represent selected sorting algorithms.

The set S of sorting algorithms represents a case of this general problem, where each algorithm can be considered a separate chapter within a textbook on computer algorithms. What makes sorting algorithms a good use case is that they share the same properties (time and space complexity, stability, etc.) while being different in important aspects. However, we believe that our approach can be generalized to most educational content. For each representation approach, we constructed specialized datasets:

Plain Text Corpus We extracted and processed the English Wikipedia articles corresponding to each algorithm in S. Original long textual content was summarized by prompting an LLM, in order to fit the most amount of text possible into the context window of the LLM while maintaining essential algorithmic concepts. The resulting corpus had a total of 2,060 words.

Knowledge Graph Dataset We used Wikidata[1] as our source of KG, as it is naturally aligned with Wikipedia, thus trying to align our structured and unstructured knowledge representations. Due to the amount of information represented in the text, as opposed to any KG, this is not fully achievable. We extracted a subgraph from Wikidata centered on the algorithms in S. The graph was pruned to remove technical edges (e.g., entity IDs) while preserving meaningful relationships. The resulting graph contained 99 triplets.

Factual Statement Dataset We prompt an LLM to transform the plain text corpus into atomic factual statements. Each paragraph was sequentially processed by the model, which had access to the requirements and factual statement criteria. The final dataset comprised of 1,471 sentences. We note that the extraction of atomic facts is not a summarization or text simplification process. Instead, it is aimed at decomposing rather complex sentences into a pedagogically relevant set of simple, self-contained statements.

Topic-Oriented Filtering Topic-oriented filtering is an opportunity for the educator to customize the underlying dataset to contain only knowledge that the educator wants to assess. Filtering is the most convenient with KGs through removing triplets with certain relation types and atomic fact sentences can be simply deleted. On the other hand, modifying the source plain text to keep only pedagogically relevant content requires a lot of effort and uncertainty.

To align with educational objectives, we applied additional filtering across all representations to emphasize the technological aspects of sorting algorithms. This included removing historical development information, and focusing instead on operational characteristics and complexity analysis. To achieve this, in the case of Wikipedia, a summarization LLM was prompted to remove undesired content, reducing the corpus from 23,643 words to 2,060 words. The factual statements were reduced from 1,471 to 1,435 sentences using heuristical techniques to locate and remove this information. In the case of the KG, we collected certain relation types that indicate the presence of this type of information, such as $WikidataProperty : P61$ ("*discoverer or inventor*") and deleted all triples from the KG with these relation types, reducing the

[1] https://www.wikidata.org/wiki/Wikidata

number of triplets from 99 to 85.

For the sake of reproducibility, the cleaned plain texts, the KG triplets, and the list of atomic facts are available at github.com/anonym.

4.2 Experimental Details

We used GPT-4o (version 2024-08-06) with a temperature setting of 0.4 for all experiments, including summarization, atomic fact decomposition, and rephrasing tasks, accessing the model through OpenAI's API[2] from a local machine (a total API cost of $10 USD). In preliminary experiments, we found that a lower temperature option (0.4) produced higher quality questions compared to a higher setting (0.7). No additional computational resources were required for the experiments.

The whole knowledge base fits into the context window of the prompts in all of the EQG prompts. The summarized Wikipedia articles consist of 2,060 words, while the atomic facts 18,021 words. In the Knowledge Graph + LLM method, the triplets were formatted as "*subject – predicate → object*", allowing the model to access all relevant relationships during question generation.

At the KG rules method, when generating questions using the first template type, if a randomly selected property was not shared by four different algorithms (which was needed for one correct answer and three distractors), we randomly sampled algorithms as distractors, under the assumption that the absence of a property in the KG indicates the algorithm lacks that characteristic. This enabled us to generate questions about properties not shared by at least four algorithms in the dataset, increasing the pool of possible questions despite the number of triplets available.

All approaches used prompts that enforced: (1) inference across multiple facts for comparison-based questions, (2) verification of answer choices by prompting to include evidence for both correct answers and distractors, and (3) grounding in explicitly stated information only. All prompts are provided in Appendix A.

4.3 Evaluation methodology

We applied seven evaluation metrics to compare the questions generated by different methods. These metrics were inspired by the evaluation methodologies of Chen et al. (2024); Elkins et al. (2023); Luo et al. (2024); Scaria et al. (2024) and were further refined based on insights gained during the annotation process.

Answer Correctness (Yes/No) Yes, if the provided answer is correct for the given question.

Distractor Incorrectness (Yes/No) Evaluates whether the distractors are valid, ensuring that none of the distractors constitute correct answers.

Option Quality (Good/Bad) Option quality is good if the correct answer and the distractors are non-trivial to distinguish and the MCQ is sufficiently challenging. Distractors should also avoid redundancy among them and trivial elimination. For example, the question should not implicitly reveal the answer, as in the case of *"Which sorting algorithm uses a heap?"*

Text Quality (1-5) Measures the linguistic quality of the question, assessing grammatical correctness, clarity, and readability. A score of 1-2 corresponds to grammatically incorrect sentences, a score of 3 indicates grammatically correct but not entirely clear questions, while a score of 4 represents understandable phrasing that could still be improved.

Compactness (Yes/No) Early experiments revealed generated MCQs with unnecessary details, such as asking about multiple concepts while the answer options can be chosen by knowing only one concept. This metric is yes if the question contains only essential information.

Template-Based vs. Creative (1-5) A very subjective metric, to assess the extent to which the question follows a template-based structure, i.e. boring (1) or diverse and exhibits creativity (5).

Would You Use It? (1-5) Evaluates whether a teacher would consider using the question in an exam. This is the most subjective metric, as it depends entirely on the preferences of the educator.

Prior studies have incorporated relevance-based metrics (Chen et al., 2024; Elkins et al., 2023; Luo et al., 2024; Scaria et al., 2024); however, we leave out this category because our preliminary findings indicated that all generated MCQs consistently met relevance criteria. Similarly, a frequently used metric evaluated whether a question aligned with a predefined category of the Bloom's Taxonomy (Bloom et al., 1956). Since our initial annotation process confirmed that all questions adhered to their respective categories, we excluded this metric as well.

[2] https://platform.openai.com/

	Answer Corr.	Distractor Incorr.	Compact.	Option Quality	Text quality	Template v. Creative	Would You Use It?
Plain Text	0.85	0.80	0.55	1.00	4.55	4.04	3.50
Atomic Facts	0.95	0.80	0.95	0.98	4.73	3.74	3.90
KG + LLM	0.90	0.95	1.00	0.95	4.71	2.65	3.78
KG Rules	0.90	0.75	1.00	0.79	4.70	2.51	3.36

Table 1: The four annotators' average scores on three factual correctness and four engagement metrics.

The evaluation of the generated MCQs was conducted by four annotators, all of whom are either current or former teachers of the Algorithm and Data Structures undergraduate course. Each annotator reviewed and annotated every example in the dataset.

4.4 Results

The performance of the methods was compared over an MCQ bank containing 20 generated questions per method (listed in Appendix B). Every MCQ was independently evaluated by all four annotators. To ensure unbiased evaluation, the MCQs were presented in a randomized order, and the annotators were unaware of their origin.

Answer Correctness, *Distractor Incorrectness*, and *Compactness* metrics are objective ones. The cases of disagreement among annotators were further examined by a designated annotator, who made the final decision. Table 1 presents the average scores of the annotators.

For the metrics where disagreement was not resolved, we conducted an analysis of inter-annotator agreement. In the case of the categorical *Option Quality* metric, we computed Fleiss' kappa (κ) across the four annotators, where the $\kappa = 0.89$ indicates a high level of agreement.

For the remaining nominal metrics, Spearman's rank correlation coefficient was employed to compare the decisions of the annotators. We report here the average of the pairwise correlations. Among the three nominal categories, the *Template-Based vs. Creative* metric achieved the highest correlation, with a coefficient of 0.61. In contrast, the *Text Quality* and *Would You Use It?* metrics exhibited lower agreement levels, with correlation coefficients of 0.31 and 0.22, respectively. The low correlation for *Would You Use It?* is understandable because this subjective measure is highly dependent on personal preferences. *Text quality* ratings received mostly 4 and 5 ratings. Annotators judged only nuances, thus the low agreement here can be attributed here to personal preferences also.

Our results reveal significant differences in MCQ factual correctness across the four approaches, with structured and semi-structured knowledge representations showing more favorable results. While the `Plain Text` approach achieved respectable scores, the aggregate performance across all factual metrics (*Answer Correctness*, *Distractor incorrectness* and *Option Quality*) shows that both `Atomic Facts` and `KG + LLM` achieved higher overall factual accuracy. This suggests that structured and semi-structured representations provide a more reliable grounding for EQG. On the other hand, the more freedom the method gets, the more creative MCQs are generated.

5 Discussion

5.1 Error analysis

The *Would You Use It?* metric indicates a preference for MCQs generated using either atomic facts or KG triplets as LLM context. This can be partly due to the conciseness of the MCQs generated by these approaches, illustrated by the *Compactness* score. The plain text-based approach produced many questions with redundant content, this being the reason for its low score on the *Compactness* metric. For instance, the question *"Which sorting algorithm is described as a stable, hybrid sorting algorithm that combines merge sort and insertion sort?"* provides multiple attributes when a smaller subset of information would uniquely identify Timsort as the correct answer for the question.

The rule-based KG approach, despite having the least amount of LLM influence, did not achieve the high factual accuracy we expected. KG incompleteness is one source of these errors, but rephrasing can also introduce mistakes due to ambiguities in edge labels. For instance, the ambiguity of the triplet "*insertion sort – derivative work → Timsort*" led to the following rephrased question: *"Which algorithm is a derivative of Timsort?"* where the direction of the *"derivative"* is changed.

Another common issue was the rigidity of the rule-based approach, which struggled with incon-

sistencies in the KG. For instance, the interchangeable use of *"subclass of"* and *"instance of"* relationships in Wikidata (*"bubble sort – instance of' → sorting algorithm"* vs. *"merge sort – subclass of' → sorting algorithm"*) created problems for rule-based generation. The KG + LLM approach overcame these issues by leveraging the LLM's ability to understand semantic similarities between edge types and node names, leading to better performance in distractor validity.

5.2 Hallucination

We address EQG's faithfulness to the given educational material. As the topic of computer algorithms is well known, the basic concepts are probably learnt by the LLMs. To measure the methods' hallucination, we designed a standalone experiment in which deliberately false information was injected into the data sources. The objective was to determine whether the generated MCQs would rely on the falsified data or the pretrained knowledge of the LLM.

Initially, we permuted the names of the sorting algorithms present in the preprocessed texts (summarized texts and atomic facts) and in the KG. Specifically, we applied a permutation where every occurrence of the algorithms' names was replaced, carefully addressing variations such as *"mergesort"* versus *"Merge sort"*. The below mapping is applied to the original algorithm identifiers:

selection sort → heapsort → insertion sort →
→ Timsort → bubble sort →
→ quicksort → merge sort → selection sort

After the replacement, the four EQG methods were executed on the permuted dataset, yielding new MCQs derived from the intentionally corrupted data. To ensure that the generated MCQs remained comparable, we subsequently revered the replacement over the new MCQs and options to restore the original algorithm names. The resulting MCQ bank was then subjected to an annotation process using only the *Answer Correctness* and *Distractor Incorrectness* metrics. For each method, 20 MCQs were evaluated and the outcomes for the corrupted MCQs are presented in Table 2.

The results show that the `KG Rules` method, which depends least on the capabilities of the LLM, exhibited the lowest degree of hallucination. In this method, the LLM's primary task was to rephrase the MCQ, ensuring that the answers remained consistently aligned with the KG. The mistakes of `KG Rules` in the corrupted data has the same source as the original data, i.e. relation direction ambiguity. In contrast, for the other three methods, wherein the LLM was responsible for constructing the MCQ, the incidence of errors was much higher and the three methods suffered from halucination at the same level.

	Answer Corr.		Distractor Incorr.	
	Original	Corrupted	Original	Corrupted
Plain Text	0.85	0.60	0.80	0.50
A. Facts	0.95	0.60	0.80	0.20
KG + LLM	0.90	0.65	0.95	0.20
KG Rules	0.90	0.95	0.75	0.60

Table 2: Hallucination experiments: Scores on original and corrupted datasets.

The `Plain Text` method achieved considerably better scores on *Distractor Incorrectness* compared to other methods. Analysing the MCQs, this can mainly attributed to behaviour of the method generating distractors that were specific to algorithmic contexts, such as *"Using a heap data structure."*

6 Conclusions

In this paper, we analysed the knowledge representation formats of long educational textual materials to automatically generate comparison-type MCQs. Our experiments show that generating EQG directly from long, plain texts gives significantly more factually incorrect answers than other representations, while rule-based MCQ generation from KG yields boring MCQs and question quality highly depends on the coverage and completeness of the KG. Instead of these methods, we recommend to list facts as the context of an EQG prompt and exploit LLMs to generate questions. The list of facts can be triplets from a KG or simple statement sentences (atomic facts) decomposed from the original long, plain text. The choice between these methods should consider two factors: (1) the availability of a high-quality KG for the domain, and (2) the desired trade-off between factual accuracy and engagement. While both approaches demonstrate overall good factual correctness, KG + LLM scoring slightly higher, the atomic facts method produces more creative and engaging questions, as indicated by its higher *"Template-Based vs. Creative"* metric compared to KG + LLM.

Limitations

In this study, we experimented exclusively with a narrow domain of sorting algorithms. A crucial limitation of our findings is that we do not know whether they hold on characteristically different other domains, like arts.

Similarly, we only generate comparison-type MCQs. We can only guess that other types of questions requiring remembering distant facts would behave in the same way.

We employed hand-crafted and manually cleaned KG as a knowledge representation option. In the real world, KG has to be constructed from text and the text-to-KG process is far from perfect (see the review of Pan et al. (2024)). The errors introduced by the text-to-KG process might decrease the efficiency of the KG+LLM method.

In this study, we assumed that the subset of the educational material – either in the format of plain text or a list of atomic facts or KG triplets – from which comparison-type MCQs can be generated is given. This is not available in real-world situations where you can have hundreds of textbook pages. As a future work, we are proposing methods to identify a reduced number of subsets of distant text chapters or subsets of atomic facts which gives an appropriate grounding for EQG.

Our evaluation setup is human labor intensive. As a future work, it would be interesting to investigate whether some of our metrics can be replaced by automatic evaluation in an LLM as a judge approach. The annotations of the four domain experts for the 160 generated MCQs provide a useful benchmark for the comparative evaluation of human and LLM as a judge evaluations.

As Section 5.2 shows, hallucination is a serious issue with the proposed methods. Improving faithfulness is one of our most important research challenges.

Acknowledgments

anonym

References

LW Anderson, DR Krathwohl, PW Airasian, PA Cruikshank, Richard Mayer, RJ Pintrich, J. Raths, and MC Wittrock. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*.

David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (ai): Understanding the potential benefits of chatgpt in promoting teaching and learning. *Journal of AI*, 7(1):52–62.

B. S. Bloom, M. B. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. 1956. *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain*. Longmans Green, New York.

Anne-Marie Brady. 2005. Assessment of learning with multiple-choice questions. *Nurse Education in Practice*, 5(4):238–242.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2023. Dense x retrieval: What retrieval granularity should we use? *arXiv preprint arXiv:2312.06648*.

Yuyan Chen, Chenwei Wu, Songzhou Yan, Panjun Liu, and Yanghua Xiao. 2024. Dr. academy: A benchmark for evaluating questioning capability in education for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3138–3167.

Philip Chung, Akshay Swaminathan, Alex J Goodell, Yeasul Kim, S Momsen Reincke, Lichy Han, Ben Deverett, Mohammad Amin Sadeghi, Abdel-Badih Ariss, Marc Ghanem, et al. 2025. Verifact: Verifying facts in llm-generated clinical text with electronic health records. *arXiv preprint arXiv:2501.16672*.

Jean-Paul Doignon and Jean-Claude Falmagne. 1985. Spaces for the assessment of knowledge. *International journal of man-machine studies*, 23(2):175–196.

Jean-Paul Doignon and Jean-Claude Falmagne. 2015. Knowledge spaces and learning spaces. *arXiv preprint arXiv:1511.06757*.

Sabina Elkins, Ekaterina Kochmar, Iulian Serban, and Jackie CK Cheung. 2023. How useful are educational questions generated by large language models? In *International Conference on Artificial Intelligence in Education*, pages 536–542. Springer.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Ryo Kamoi, Tanya Goyal, Juan Diego Rodriguez, and Greg Durrett. 2023. Wice: Real-world entailment for claims in wikipedia. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7561–7583.

Natthawut Kertkeidkachorn and Ryutaro Ichise. 2017. T2kg: An end-to-end system for creating knowledge graph from unstructured text. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text generation from knowledge graphs with graph transformers. *arXiv preprint arXiv:1904.02342*.

Archana Praveen Kumar, Ashalatha Nayak, Manjula Shenoy K, Chaitanya, and Kaustav Ghosh. 2024. A novel framework for the generation of multiple choice question stems using semantic and machine-learning techniques. *International Journal of Artificial Intelligence in Education*, 34(2):332–375.

Haohao Luo, Yang Deng, Ying Shen, See Kiong Ng, and Tat-Seng Chua. 2024. Chain-of-exemplar: Enhancing distractor generation for multimodal educational question generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7978–7993.

Subhankar Maity, Aniket Deroy, and Sudeshna Sarkar. 2024. A novel multi-stage prompting approach for language agnostic mcq generation using gpt. In *European Conference on Information Retrieval*, pages 268–277. Springer.

Joan C Masters, Barbara S Hulsmeyer, Mary E Pike, Kathy Leichty, Margaret T Miller, and Amy L Verst. 2001. Assessment of multiple-choice questions in selected test banks accompanying text books used in nursing education.

Igor Melnyk, Pierre Dognin, and Payel Das. 2022. Knowledge graph generation from text. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1610–1622.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. pages 12076–12100.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*.

Andreas Papasalouros, Konstantinos Kanaris, and Konstantinos Kotis. 2008. Automatic generation of multiple choice questions from domain ontologies. *e-Learning*, 1:427–434.

Nicy Scaria, Suma Dharani Chenna, and Deepak Subramani. 2024. Automated educational question generation at different bloom's skill levels using large language models: Strategies and evaluation. In *International Conference on Artificial Intelligence in Education*, pages 165–179. Springer.

Yoshija Walter. 2024. Embracing the future of artificial intelligence in the classroom: the relevance of ai literacy, prompt engineering, and critical thinking in modern education. *International Journal of Educational Technology in Higher Education*, 21(1):15.

Jiayi Wang, Ruiwei Xiao, and Ying-Jui Tseng. 2024. Generating ai literacy mcqs: A multi-agent llm approach. *arXiv e-prints*, pages arXiv–2412.

Miriam Wanner, Seth Ebner, Zhengping Jiang, Mark Dredze, and Benjamin Van Durme. 2024. A closer look at claim decomposition. *arXiv preprint arXiv:2403.11903*.

Zonghai Yao, Aditya Parashar, Huixue Zhou, Won Seok Jang, Feiyun Ouyang, Zhichao Yang, and Hong Yu. 2024. Mcqg-srefine: Multiple choice question generation and evaluation with iterative self-critique, correction, and comparison feedback. *arXiv preprint arXiv:2410.13191*.

A Prompts

This section shows the prompts that were applied to the results of the paper.

A.1 Plain Text-Based Method

The following prompt was used to summarize Wikipedia articles.

```
Remove any unnecessary text from the given passage, retaining only the essential information needed to understand the main topic, while ensuring individuals familiar with the topic gain the same insights as before.

# Steps

1. **Read the Text**: Carefully review the entire passage to comprehend its main topic and context.
2. **Identify Key Points**: Determine the primary ideas, statements, or data that are crucial for understanding the topic, ensuring all necessary insights remain for those familiar with the subject.
3. **Filter Out Extraneous Information**: Remove any superfluous text such as filler words, repetitive phrases, tangential details, and unrelated anecdotes that do not contribute to the core understanding of the topic.
4. **Maintain Coherence**: Ensure the remaining text is clear and coherent without compromising the primary message or understanding of the topic, retaining essential insights for knowledgeable readers.

# Output Format

A concise and focused text that contains only the necessary information needed to understand the main topic, ensuring comprehensive understanding by all audiences.

# Notes

- Consider the audience for which the text is being tailored to ensure that all retained information is relevant and maintains the knowledge depth suitable for those familiar with the topic.
- Be mindful of retaining context and logical flow even as you remove extraneous text, ensuring experts continue to learn at the same level of depth.
```

The following prompt was applied to generate MCQs from the summarized texts.

```
Create multiple choice questions from the provided documents frathat assess Understanding" according to Anderson's taxonomy.

Focus on generating questions that require learners to infer, and compare content from the documents.

# Steps

1. **Analyze the Documents:** Thoroughly read the attached documents to grasp key concepts and details necessary for creating insightful questions.
2. **Determine Key Concepts:** Identify the main ideas and important supporting details.
3. **Generate Questions:** Formulate clear and easy to understand questions that require the application of understanding skills, such as interpretation or summarization.
4. **Create Options:** Develop plausible distractors for each question along with the correct answer.
5. **Provide Context:** Include a context from the sources for each question to guide learners, based on the attached documents.

# Output Format

The output should be formatted as a JSON array of objects. Each object should represent a question and include:
- **question**: The text of the question.
- **options**: An array of answer choices.
- **correct_answer**: The position or index of the correct answer in the options array. Starting from 0.
- **grounding**: A context from the sources for the question, based on the attached documents.

```json
[
  {
    "question": "What is the main concept addressed in [Section/Paragraph]?",
    "options": ["Option A", "Option B", "Option C", "Option D"],
    "correct_answer": 1,
    "grounding": "Provide a context from the sources for the question."
  },
]
```

Examples

Example 1:

Input:
An article about an algorithm.
An article about an another algorithm.

Output:
```json
[
  {
    "question": "Which algorithm has instance of of divide-and-conquer algorithm?",
```
```

```
    "options": ["heapsort", "quicksort", "Timsort", "selection sort"],
    "correct_answer": 1,
    "grounding": "The divide-and-conquer algorithm is a key concept in quicksort."
  }
]
```

(Real examples should be longer and customized to the attached documents, focusing on understanding concepts presented.)

Notes

- Ensure each question is designed to test understanding within Anderson's taxonomy.
- Verify that distractors are plausible to encourage critical thinking.
- Tailor questions to reflect the specific content and style of the attached documents.
- Provide context from the sources to guide learners in answering the questions.
```

## A.2 Atomic Facts

To generate questions from the atomic facts we used the following prompt:

```
Given a knowledge base of factual statements about a topic, generate 10 multiple choice questions that test understanding of the concepts and relationships described in these facts.

Focus on generating questions that require learners to infer, and compare content from the triplets.

Steps

1. First, **identify clusters of related facts** by either:
   a) Finding predicates that connect multiple subjects to the same type of object
      (e.g., all algorithms with their space complexity)
   OR
   b) Finding predicates that connect one subject to multiple objects
      (e.g., all properties of one algorithm)

2. **Before formulating a question, verify**:
   - You have enough distinct options for 4 meaningful choices
   - The relationships are unambiguous (one clear correct answer)
   - Supporting evidence exists for both the correct answer and distractors
   - For property questions: at least 4 different algorithms have this property
   - For value questions: at least 4 different possible values exist across algorithms

Guidelines for questions

1. Answers must be definitively provable from the given facts
2. All 4 options must be plausible and related to the topic
3. Incorrect options (distractors) should be based on facts about other related concepts
4. The distractor answers should not be too trivial
5. Questions should test relationships between concepts or comparative properties

Output Format

The output should be formatted as a JSON array of objects. Return nothing but the JSON response, pay attention to the format so it can be loaded by Python's `json.loads` without any modifications. Each object should represent a question and include:
- **question**: The text of the question.
- **choices**: An array of answer choices.
- **correct**: The position or index of the correct answer in the options array. Starting from 0.
- **grounding**: A list of factual statements from the sources for the question, based on the attached documents.

```json
[
   {
       "question": "...",
       "choices": [
           "...",
           "...",
           "...",
           "..."
       ],
       "correct": 0-3,  // index of the correct answer
       "grounding": ["...", "...", "..."] // list the relevant facts supporting the correctness of the correct answer option and any relevant information to the distractors
   }
]
```

Examples

Example 1:

Input:
List of factual statements about algorithms and their properties.

Output:
```json
[
```

````
    {
        "question": "Which algorithm is not a stable sorting algorithm?",
        "choices": [
            "Timsort",
            "Heapsort",
            "Insertion Sort",
            "Bubble Sort"
        ],
        "correct": 1,
        "grounding": [
            "Timsort is a stable sorting algorithm",
            "Heapsort is not stable",
            "Insertion sort is a stable sorting algorithm",
            "Bubble sort is a stable sorting algorithm"
        ]
    }
]
```

**Example 2:**

_Input:_
List of factual statements about algorithms and their properties.

_Output:_
```json
[
    {
        "question": "What is the worst-case space complexity of merge sort?",
        "choices": [
            "O(n)",
            "O(1)",
            "O(n log n)",
            "O(n^2)"
        ],
        "correct": 0,
        "grounding": [
            "Merge sort has a worst-case space complexity of O(n).",
            "Merge sort has space complexity of O(n).",
            "Quicksort has a worst-case space complexity of O(n).",
            "Heapsort has a worst-case space complexity of O(1)."
        ]
    }
]
```

# Rules

1. Only use information explicitly stated in the fact statements
2. Only reference concepts mentioned in the knowledge base
3. Treat the facts as the sole source of truth
4. If something isn't explicitly stated in a fact, don't assume it
5. Consider different ways the same concept might be expressed

# Notes

- Verify that distractors are plausible to encourage critical thinking.
- Provide triplets from the sources to guide learners in answering the questions.

Return nothing but the json response, formatted to be loaded by Python's json.loads without any modifications.
````

The facts were extracted from the plain texts by the following prompt:

```
You are tasked with extracting fact-based statements from a text.

Guidelines:
1. Break down complex sentences into simple, atomic facts. Each fact must clearly indicate its topic and be completely self-contained.
2. Only use information present in the source text
3. Only return pieces of information which are relevant for later testing knowledge on the topic in an educational setting (as the facts will be used for constructing test questions)
4. Focus on technical, definitional, and functional aspects that demonstrate understanding of the core concept
5. Each statement must clearly indicate what topic or concept it's describing, as if it could appear in a random set of facts about different topics
   BAD: "Compares each element with the next one"
   GOOD: "Bubble sort compares each element with the next element in the list"

Input Format:
<INPUT>
Your text goes here...
</INPUT>

Output Format:
[
    "Fact statement 1",
```
```

```
    "Fact statement 2",
    ...
]

Example:

<INPUT>
The number ***π*** (/paɪ/; spelled out as "**pi**") is a mathematical constant, approximately equal to 3.14159, that is the ratio of a circle's circumference to its diameter. In addition to being irrational, *π* is also a transcendental number, which means that it is not the solution of any non-constant polynomial equation with rational coefficients.
</INPUT>

Output:
[
    "π is approximately equal to 3.14159",
    "π equals the ratio of a circle's circumference to its diameter",
    "π is an irrational number",
    "π is a transcendental number",
    "A transcendental number cannot be the solution of any non-constant polynomial equation with rational coefficients"
]

Possible statements left out due to their lack of relevancy:
- "The number π is a mathematical constant", --> as this is a very trivial statement
- "π is pronounced as 'paɪ'", --> this can't be effectively used when constructing a written test on π
- "π is spelled out as 'pi'", --> as this is a very trivial statement
```

A.3 Knowledge Graph + LLM

The below system prompt was applied to generate questions from the KG:

```
Given a knowledge base of facts in the form of triplets (subject--predicate->object), generate multiple choice questions that test understanding of relationships and properties in the knowledge graph.

Focus on generating questions that require learners to infer, and compare content from the triplets.

# Steps

1. First, **identify clusters of related facts** by either:
   a) Finding predicates that connect multiple subjects to the same type of object
      (e.g., all algorithms with their space complexity)
   OR
   b) Finding predicates that connect one subject to multiple objects
      (e.g., all properties of one algorithm)

2. **Before formulating a question, verify**:
   - You have enough distinct options for 4 meaningful choices
   - The relationships are unambiguous (one clear correct answer)
   - Supporting evidence exists for both the correct answer and distractors
   - For property questions: at least 4 different algorithms have this property
   - For value questions: at least 4 different possible values exist across algorithms

3. **Document all relevant supporting triplets** that:
   - Prove the correct answer
   - Disprove incorrect options
   - Use semantically similar predicates
   - Validate the complete set of choices

# Output Specifications

1. Generate 10 questions
2. Each question must have exactly 4 options
3. Only ONE option should be correct based on the knowledge base
4. All answers must be supported by explicit triplets
5. The distractor answers should not be too trivial

# Output Format

The output should be formatted as a JSON array of objects. Return nothing but the JSON response, pay attention to the format so it can be loaded by Python's `json.loads` without any modifications. Each object should represent a question and include:
- **question**: The text of the question.
- **choices**: An array of answer choices.
- **correct**: The position or index of the correct answer in the choices array. Starting from 0.
- **supporting_triplets**: Supporting triplets from the input, keep the format. Do not modify the triplets.

```json
[
    {
        "question": "...",
        "choices": [
            "...",
            "...",
            "...",
            "..."
        ],
        "correct": 0-3,  // index of the correct answer
```
```

```
        "supporting_triplets": [
            // format: "subject--predicate->object, do not include any comments as it makes the result unparsable
        ]
    }
]
```

Examples

Example 1:

Input:
Triplets from a knowledge graph, about algorithms and their properties.

Output:
```json
[
    {
        "question": "Which algorithm is not a stable sorting algorithm?",
        "choices": [
            "Timsort",
            "Heapsort",
            "Insertion Sort",
            "Bubble Sort"
        ],
        "correct": 1,
        "supporting_triplets": [
            "Timsort--instance of->stable sorting algorithm",
            "heapsort--has property->not stable",
            "insertion sort--instance of->stable sorting algorithm",
            "insertion sort--is a type of->stable sorting algorithm",
            "bubble sort--instance of->stable sorting algorithm",
            "bubble sort--is a type of->stable sorting algorithm",
            "bubble sort--is->stable sorting algorithm"
        ]
    }
]
```

Note how the same concept "being stable" is expressed through different predicates:
- "instance of->stable sorting algorithm"
- "is a type of->stable sorting algorithm"
- "is->stable sorting algorithm"
- "has property->not stable"

Example 2:

Input:
Triplets from a knowledge graph, about algorithms and their properties.

Output:
```json
[
    {
        "question": "What is the worst-case space complexity of merge sort?",
        "choices": [
            "O(n)",
            "O(1)",
            "O(n log n)",
            "O(n^2)"
        ],
        "correct": 0,
        "supporting_triplets": [
            "merge sort--worst-case space complexity->O(n)",
            "merge sort--has space complexity->O(n)",
            "merge sort--space complexity->O(n)",
            "quicksort--worst-case space complexity->O(n)",
            "heapsort--worst-case space complexity->O(1)"
        ]
    }
]
```

Note how space complexity can be expressed through variations like:
- "worst-case space complexity"
- "has space complexity"
- "space complexity"

Rules

1. Only use information explicitly stated in the triplets
2. Only reference entities mentioned in the knowledge base
3. Treat the knowledge base as the sole source of truth
4. If a fact isn't explicitly stated in a triplet, don't assume it
5. Consider semantic equivalence of different predicates

Notes

```
- Verify that distractors are plausible to encourage critical thinking.
- Provide triplets from the sources to guide learners in answering the questions.

Return nothing but the json response, formatted to be loaded by Python's json.loads without any modifications.
```

Where the edges were added to the message with *User* role in the *subject –predicate → object* format:

```
Knowledge base:
{edges}
```

A.4 Rule-based EQG from Knowledge Graphs

We used the following prompt to rephrase the rule-based MCQs:

```
Rewrite the following JSON questions and options to make them more readable and user-friendly by:
Simplifying and improving readability.
Correcting any grammatically incorrect sentences.
Replacing unnecessarily long date formats with concise and clear ones (e.g.: in the case of Jan 01 dates show only the year).
Phrasing the question in such a way that it is less like it was generated using a template.
Make the questions more creative, but preserve the original meaning.

Return nothing but the json response, formatted to be loaded by Python's json.loads without any modifications.
```

B Generated Questions

Atomic Facts

1. Which algorithm is a hybrid of merge sort and insertion sort?
   - Heapsort
   - Timsort
   - Quicksort
   - Bubble Sort
2. Which algorithm is most suitable for sorting linked lists?
   - Merge Sort
   - Quicksort
   - Heapsort
   - Selection Sort
3. Which algorithm is particularly efficient for sorting linked lists?
   - Merge Sort
   - Quicksort
   - Heapsort
   - Bubble Sort
4. Which sorting algorithm has a worst-case time complexity of O(n$\hat{2}$)?
   - Bubble Sort
   - Merge Sort
   - Heapsort
   - Timsort
5. Which sorting algorithm is known for its efficiency in sorting arrays with many equal elements?
   - Quicksort
   - Merge Sort
   - Heapsort
   - Selection Sort
6. Which sorting algorithm is known for its poor locality of reference?
   - Heapsort
   - Quicksort
   - Merge Sort
   - Insertion Sort
7. Which sorting algorithm is known for its simple implementation but poor performance on large datasets?
   - Merge Sort
   - Quicksort
   - Bubble Sort
   - Timsort
8. Which sorting algorithm is known to be stable?
   - Merge Sort
   - Quicksort
   - Heapsort
   - Selection Sort

9. Which sorting algorithm is known to perform poorly on already sorted data?
    - Quicksort with the first element as the pivot
    - Merge Sort
    - Heapsort
    - Insertion Sort

10. Which sorting algorithm is known to perform poorly on large lists due to its quadratic time complexity?
    - Merge Sort
    - Quicksort
    - Bubble Sort
    - Heapsort

11. Which sorting algorithm is particularly beneficial when sorting data stored on slow-to-access media?
    - Quicksort
    - Bubble Sort
    - Merge Sort
    - Insertion Sort

12. Which sorting algorithm is particularly inefficient for large lists?
    - Bubble Sort
    - Merge Sort
    - Quicksort
    - Heapsort

13. What is the main advantage of Timsort over Quicksort for sorting object references or pointers?
    - Less memory usage
    - Better locality of reference
    - Stability
    - Faster execution time

14. Which sorting algorithm is specifically designed to handle large datasets stored on slow-to-access media?
    - Merge Sort
    - Quicksort
    - Heapsort
    - Insertion Sort

15. Which sorting algorithm is typically less efficient on large lists compared to merge sort?
    - Insertion Sort
    - Quicksort
    - Heapsort
    - Bubble Sort

16. Which sorting algorithm is typically more efficient than bubble sort on average?
    - Insertion Sort
    - Selection Sort
    - Merge Sort
    - Quicksort

17. Which sorting algorithm is typically used as a fallback when quicksort becomes inefficient?
    - Heapsort
    - Merge Sort
    - Bubble Sort
    - Insertion Sort

18. Which sorting algorithm is used as the default in Python since version 2.3?
    - Quicksort
    - Heapsort
    - Timsort
    - Merge Sort

19. Which sorting algorithm uses a 'divide-and-conquer' approach?
    - Quicksort
    - Insertion Sort
    - Bubble Sort
    - Selection Sort

20. What is the primary advantage of using Timsort over Quicksort?
    - Stability
    - Lower space complexity
    - Faster average-case performance
    - Better worst-case time complexity

KG + LLM

1. Which algorithm has a best-case time complexity of O(n)?
   - Insertion Sort
   - Merge Sort
   - Heapsort
   - Quicksort
2. Which algorithm has a worst-case space complexity of O(1)?
   - Heapsort
   - Merge Sort
   - Quicksort
   - Timsort
3. Which algorithm has a worst-case time complexity of O(n$\hat{2}$)?
   - Bubble Sort
   - Heapsort
   - Merge Sort
   - Timsort
4. Which algorithm has a worst-case time complexity of O(n$\hat{2}$)?
   - Bubble Sort
   - Timsort
   - Heapsort
   - Merge Sort
5. Which algorithm has an average time complexity of O(n log n)?
   - Heapsort
   - Bubble Sort
   - Insertion Sort
   - Selection Sort
6. Which algorithm is a derivative work of Timsort?
   - Insertion Sort
   - Merge Sort
   - Heapsort
   - Bubble Sort
7. Which algorithm is an example of an adaptive sort?
   - Timsort
   - Heapsort
   - Quicksort
   - Selection Sort
8. Which algorithm is based on both merge sort and insertion sort?
   - Timsort
   - Heapsort
   - Quicksort
   - Bubble Sort
9. Which algorithm is based on both merge sort and insertion sort?
   - Timsort
   - Heapsort
   - Quicksort
   - Bubble Sort
10. Which algorithm is named after Tim Peters?
    - Timsort
    - Bubble Sort
    - Insertion Sort
    - Merge Sort
11. Which algorithm is not a comparison sort?
    - Heapsort
    - Timsort
    - Bubble Sort
    - Merge Sort
12. Which algorithm is not a stable sorting algorithm?
    - Heapsort
    - Bubble Sort
    - Insertion Sort
    - Merge Sort
13. Which algorithm is not a stable sorting algorithm?
    - Heapsort
    - Timsort
    - Merge Sort
    - Bubble Sort
14. Which algorithm is used by Python for sorting?
    - Timsort
    - Quicksort
    - Heapsort
    - Merge Sort
15. Which algorithm is used by Python?

- Timsort
- Bubble Sort
- Insertion Sort
- Heapsort

16. Which algorithm is used by the Java Platform, Standard Edition?

    - Timsort
    - Bubble Sort
    - Insertion Sort
    - Heapsort

17. Which sorting algorithm has a best-case time complexity of O(n)?

    - Insertion Sort
    - Heapsort
    - Merge Sort
    - Quicksort

18. Which sorting algorithm has a worst-case space complexity of O(1)?

    - Heapsort
    - Merge Sort
    - Timsort
    - Quicksort

19. Which sorting algorithm is an instance of a comparison sort?

    - Quicksort
    - Heapsort
    - Merge Sort
    - Bubble Sort

20. Which sorting algorithm is an instance of an adaptive sort?

    - Timsort
    - Heapsort
    - Quicksort
    - Merge Sort

KG rules

1. What kind of algorithm is Merge Sort?
   - Sorting Algorithm
   - Stable Sorting Algorithm
   - Online Algorithm
   - Divide-and-Conquer Algorithm
2. What kind of algorithm is Selection Sort?
   - Sorting Algorithm
   - Divide-and-Conquer Algorithm
   - Online Algorithm
   - Adaptive Sort
3. What kind of algorithm is Timsort?
   - Comparison Sort
   - Divide-and-Conquer Algorithm
   - Online Algorithm
   - In-Place Algorithm
4. What type of algorithm is Timsort?
   - Adaptive Sort
   - Divide-and-Conquer Algorithm
   - In-Place Algorithm
   - Online Algorithm
5. What type of algorithm is Timsort?
   - Online Algorithm
   - Divide-and-Conquer Algorithm
   - In-Place Algorithm
   - Stable Sorting Algorithm
6. Which algorithm has a best-case time complexity of O(n log n)?
   - Merge Sort
   - Bubble Sort
   - Heapsort
   - Selection Sort
7. Which algorithm has a worst-case space complexity of O(1) auxiliary?
   - Bubble Sort
   - Insertion Sort
   - Heapsort
   - Timsort
8. Which algorithm has a worst-case space complexity of O(1)?
   - Heapsort
   - Merge Sort
   - Quicksort
   - Bubble Sort
9. Which algorithm has a worst-case time complexity of O(n log n)?
   - Insertion Sort
   - Quicksort
   - Merge Sort
   - Selection Sort
10. Which algorithm has a worst-case time complexity of O(n log n)?
    - Selection Sort
    - Insertion Sort
    - Timsort
    - Bubble Sort
11. Which algorithm is a derivative of Timsort?
    - Insertion Sort
    - Selection Sort
    - Quicksort
    - Timsort
12. Which algorithm is an example of a divide-and-conquer algorithm?
    - Selection Sort
    - Quicksort
    - Heapsort
    - Insertion Sort
13. Which algorithm is an example of an online algorithm?
    - Insertion Sort
    - Timsort
    - Merge Sort
    - Bubble Sort
14. Which algorithm is based on Insertion Sort?
    - Bubble Sort
    - Timsort
    - Heapsort
    - Merge Sort
15. Which algorithm is based on Merge Sort?
    - Timsort
    - Selection Sort
    - Quicksort
    - Insertion Sort

16. Which algorithm is derived from Smoothsort?

    - Selection Sort
    - Insertion Sort
    - Merge Sort
    - Heapsort

17. Which algorithm is used by the V8 engine?

    - Insertion Sort
    - Bubble Sort
    - Timsort
    - Heapsort

18. Which algorithm uses the merge algorithm?

    - Quicksort
    - Merge Sort
    - Bubble Sort
    - Insertion Sort

19. Which sorting method is an example of an in-place algorithm?

    - Quicksort
    - Selection Sort
    - Insertion Sort
    - Heapsort

20. Which sorting method is named after bubbles?

    - Bubble Sort
    - Insertion Sort
    - Selection Sort
    - Timsort

Plain text

1. In what scenario is Merge Sort particularly advantageous?
   - Sorting small datasets
   - Sorting large datasets with external storage
   - Sorting data in-place
   - Sorting data with minimal memory
2. What is a common optimization for quicksort to avoid worst-case performance?
   - Using a fixed pivot
   - Using insertion sort for small arrays
   - Switching to bubble sort
   - Increasing recursion depth
3. What is a key advantage of Timsort over Quicksort?
   - Timsort is a stable sort
   - Timsort has better worst-case time complexity
   - Timsort uses less memory
   - Timsort is easier to implement
4. Which algorithm is described as using a 'pivot' element to partition the array into sub-arrays?
   - Insertion Sort
   - Merge Sort
   - Quicksort
   - Selection Sort
5. What is a key disadvantage of heapsort compared to quicksort?
   - Heapsort is not stable.
   - Heapsort has a higher worst-case time complexity.
   - Heapsort is not an in-place algorithm.
   - Heapsort is recursive.
6. What is a primary disadvantage of Selection Sort compared to Insertion Sort?
   - Higher time complexity
   - More memory usage
   - Less efficient for small datasets
   - More write operations
7. Which sorting algorithm is described as a stable, hybrid sorting algorithm that combines merge sort and insertion sort?
   - Heapsort
   - Timsort
   - Quicksort
   - Bubble Sort
8. Which sorting algorithm is described as having a worst-case time complexity of $O(n\hat{2})$ but is simple and can be advantageous when auxiliary memory is limited?
   - Selection Sort
   - Merge Sort
   - Heapsort
   - Timsort
9. Which sorting algorithm is generally more efficient for partially sorted data?
   - Heapsort
   - Bubble Sort
   - Insertion Sort
   - Selection Sort
10. Which sorting algorithm is known for its efficient performance on small datasets and partially sorted data?
    - Insertion Sort
    - Bubble Sort
    - Merge Sort
    - Quicksort
11. Which sorting algorithm is known for its poor locality of reference?
    - Heapsort
    - Merge Sort
    - Quicksort
    - Timsort
12. What is a significant disadvantage of heapsort compared to quicksort?
    - It is not stable
    - It has a higher worst-case time complexity
    - It uses more memory
    - It is harder to implement
13. Which sorting algorithm is known for its stable sorting and efficient performance on sequentially accessed data?
    - Heapsort
    - Quicksort
    - Merge Sort

- Selection Sort

14. Which sorting algorithm is noted for its inefficiency on large datasets but is simple and often used for educational purposes?
    - Merge Sort
    - Bubble Sort
    - Quicksort
    - Insertion Sort

15. Which sorting algorithm is particularly inefficient for large datasets due to its (O(n$\hat{2}$)) complexity?
    - Bubble Sort
    - Heapsort
    - Timsort
    - Merge Sort

16. Which sorting algorithm is particularly noted for its use in educational contexts due to its simplicity, despite its inefficiency?
    - Heapsort
    - Selection Sort
    - Bubble Sort
    - Insertion Sort

17. Which sorting algorithm is typically faster for randomized data due to better cache coherence?
    - Heapsort
    - Merge Sort
    - Quicksort
    - Bubble Sort

18. Which sorting algorithm is typically used in hybrid forms like Timsort due to its stability and efficiency for sequential media?
    - Merge Sort
    - Quicksort
    - Heapsort
    - Selection Sort

19. Which sorting algorithm is used in Python due to its efficiency on real-world data?
    - Heapsort
    - Merge Sort
    - Timsort
    - Bubble Sort

20. What is the primary advantage of merge sort over quicksort?
    - Merge sort is in-place.
    - Merge sort is stable.
    - Merge sort has better average time complexity.
    - Merge sort requires less space.