1 Appendix

1.1 Knol-ML: Core Layer

```
<KnowledgeData Type="text" Id="1">
    <Title>This is an example for sequential knowledge data</Title>

    <Instance Id="1" InstanceType="Revision">
        <TimeStamp><CreationDate>t1</CreationDate></TimeStamp>
        <Contributors>
            <OwnerUserName>Editor 1</OwnerUserName>
            <OwnerUserId>1</OwnerUserId>
        </Contributors>
        <Body><Text Type="text">Edits of Editor 1</Text></Body>
    </Instance>

    <Instance Id="2" InstanceType="Revision">
        <TimeStamp><CreationDate>t2</CreationDate></TimeStamp>
        <Contributors>
            <OwnerUserName>Editor 2</OwnerUserName>
            <OwnerUserId>2</OwnerUserId>
        </Contributors>
        <Body><Text Type="text">Edits of Editor 2</Text></Body>
    </Instance>
        ⋮
</KnowledgeData>
```

Figure 1. Representation of sequential knowledge data in Knol-ML format.

Figure 1 depicts the Knol-ML format for sequential knowledge data. A Knol-ML document may contain multiple `<KnowledgeData>`, each representing different document. The topic of a document is described using the `<Title>` tag. The information regarding a particular edit performed by a contributor is described within the `<Instance>` tag. It contains three mandatory tags explained below:

- A `<TimeStamp>` tag contains a sub-tag `<CreationDate>`, which stores the commit date and time of this edit.
- `<Contributors>` tag stores the details regarding editor of this instance. It has two further sub-tags, `<OwnerUserName>`, which stores the name of the editor and `<OwnerUserId>`, which stores the unique Id assigned to the editor. The former is optional, but the latter is mandatory.
- The `<Body>` tag contains a sub-tag `<Text>`, which stores the actual edits of this instance.

The sub-tags of `<Contributors>` and `<Body>` tags ensure that additional information can be included within them appropriately.

1.2 Wiki Based Data

Under the umbrella of *Wikimedia Foundation*, Wikipedia[1] is a free encyclopedia, where anyone can edit and contribute knowledge. As of March 2019, English Wikipedia contains

[1]https://www.wikipedia.org/

```
<KnowledgeData Type="text" Id="1">
    <Title>Title of the article</Title>

    <Instance Id="1" InstanceType="Revision">
        <TimeStamp><CreationDate>t1</CreationDate></TimeStamp>
        <Contributors>
            <OwnerUserName>Editor 1</OwnerUserName>
            <OwnerUserId>1</OwnerUserId>
        </Contributors>
        <EditDetails>
             <EditType>details of edit</EditType>
        </EditDetails>
        <Body><Text Type="text">Edits of Editor 1</Text></Body>
    </Instance>
          ⋮
</KnowledgeData>
```

Figure 2. Representation of Wikipedia data in Knol-ML.

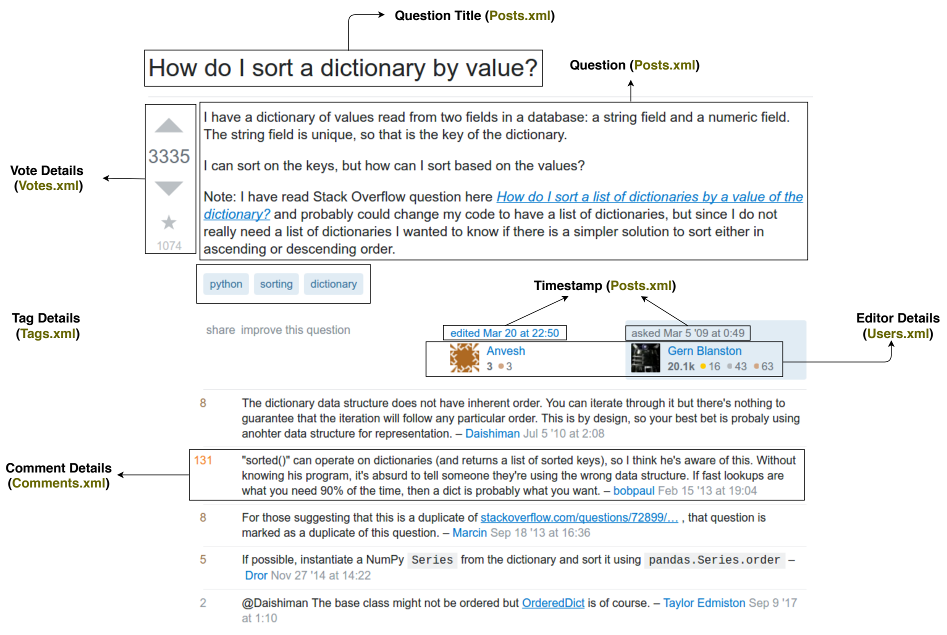
5,817,607 articles. Wikimedia foundation release data dumps of Wikipedia and all of its projects on a regular basis. The data dump comprises of the following:

- Text of the current and all the previous revisions as XML files with defined wrapper format[2].
- Information about other Wikipedia pages like *Talk*, *User Talk*, *Templates*, etc.

Our focus is on Wikipedia articles, which are the main source of the referred knowledge. The XML dump of articles contains information like, *title* and edits in the form of *revisions*. Each revision has details like, *date and time* of the edit, *editors' information*, *comments*, if any and the actual *edits* in text form. For every such detail, a tag has been defined. The XML dump may contain multiple pages, each representing a different article. Each article is considered as `<KnowledgeData>` in Knol-ML format. Figure 2 depicts the Knol-ML format for Wikipedia data. The details of the tags are as follows:

- `<Title>` stores the title of the article.
- `<Instance>` stores all the information related to a particular edit, containing the following tags:
    - `<TimeStamp>` stores the date and time of the edit.
    - `<Contributors>` stores the editors' information using the subtags `<OwnerUserId>` and `<OwnerUserName>`.
    - `<EditDetails>` stores the comment related to the edit.
    - `<Body>` stores the actual edit in the `<Text>` subtag.

[2]https://www.mediawiki.org/xml/export-0.8.xsd



How do I sort a dictionary by value?

I have a dictionary of values read from two fields in a database: a string field and a numeric field. The string field is unique, so that is the key of the dictionary.

I can sort on the keys, but how can I sort based on the values?

Note: I have read Stack Overflow question here How do I sort a list of dictionaries by a value of the dictionary? and probably could change my code to have a list of dictionaries, but since I do not really need a list of dictionaries I wanted to know if there is a simpler solution to sort either in ascending or descending order.

8 The dictionary data structure does not have inherent order. You can iterate through it but there's nothing to guarantee that the iteration will follow any particular order. This is by design, so your best bet is probaly using anohter data structure for representation. – Daishiman Jul 5 '10 at 2:08

131 "sorted()" can operate on dictionaries (and returns a list of sorted keys), so I think he's aware of this. Without knowing his program, it's absurd to tell someone they're using the wrong data structure. If fast lookups are what you need 90% of the time, then a dict is probably what you want. – bobpaul Feb 15 '13 at 19:04

8 For those suggesting that this is a duplicate of stackoverflow.com/questions/72899/... , that question is marked as a duplicate of this question. – Marcin Sep 18 '13 at 16:36

5 If possible, instantiate a NumPy `Series` from the dictionary and sort it using `pandas.Series.order` – Dror Nov 27 '14 at 14:22

2 @Daishiman The base class might not be ordered but OrderedDict is of course. – Taylor Edmiston Sep 9 '17 at 1:10

Figure 3. An illustration of Stack Overflow data storage.

1.3 QnA Based Data

Unlike wiki-based portals, knowledge building in QnA-based portals like, Stack overflow[3] and Quora[4], develops through a series of discussions on a particular topic. The discussions may involve series of questions and answers, but not necessarily. The highlight of these portals is the explicitness of knowledge as the crowd provides question specific answers.

Portals like *Stack Overflow* and *Ask Ubuntu* are part of *Stack Exchange Network*. Currently, there are 185 QnA based portals maintained by Stack Exchange. Domain of these portals varies from programming to music. Similar to Wikipedia, Stack Exchange provides data dump of its websites in XML format. Data dump of each portal contains eight XML files, namely, *Posts.xml*, *PostHistory.xml*, *Users.xml*, *Comments.xml*, *Tags.xml*, *Votes.xml*, *Badges.xml* and *PostLinks.xml*, each of which contains different set of information. For example, *Posts.xml* contains all the questions and answers. *Users.xml* contains information related to all the users, *Comments.xml* contains comments of posts and related information.

Based on the design of QnA portal, every question sequentially accumulates posts like answers, comments, votes, etc. Our aim is to combine the data of these portals into sequential knowledge building format. A *thread* is defined as a question and all the posts related to it (answers, comments, votes, scores, etc.) [?]. We represent the information of each thread in the form of knowledge data. A question, an answer or a comment associated with a time-stamp is considered as an instance of Knol-ML. For example, a question q_1 may arrive at time t_1, followed by an answer a_1 at time t_2 ($t_i < t_{i+1}$). The answer a_1 may get multiple comments, hence, a post (question, answer or comment) arriving at time-stamp

[3]https://stackoverflow.com/
[4]https://www.quora.com/

```xml
<KnowledgeData Type="text" Id="1">
    <Title>Question Title</Title>


    <Instance Id="1" InstanceType="Question">
        <TimeStamp>
            <CreationDate>t1</CreationDate>
            <LastActivityDate>t2</LastActivityDate>
        </TimeStamp>
        <Contributors>
            <OwnerUserName>Editor 1</OwnerUserName>
            <ownerUserId>1</OwnerUserId>
            <LastEditorUserName>Editor 3</LastEditorUserName>
            <LastEditorUserId>3</LastEditorUserId>
        </Contributors>
        <Body><Text Type="text">Question by Editor 1</Text></Body>
        <Tags>tags</Tags>
        <Credit><Score>score value</Score></Credit>
    </Instance>


    <Instance Id="2" InstanceType="Comment">
        <TimeStamp>
            <CreationDate>t3</CreationDate>
        </TimeStamp>
        <Contributors>
            <OwnerUserName>Editor 4</OwnerUserName>
            <OwnerUserId>4</OwnerUserId>
        </Contributors>
        <Body><Text Type="text">Comment by Editor 4</Text></Body>
        <Credit><Score>score value</Score></Credit>
    </Instance>
</KnowledgeData>
```

Figure 4. Representation of Stack Overflow data in Knol-ML.

t_i will be considered as i^{th} instance. There could be multiple information associated with each instance like *upvotes*, *downvotes*, *editor's detail*, etc. All such information will be represented within the corresponding instance.

Figure 4 is an example representing a Stack Overflow thread. A single Knol-ML file can have multiple threads, each described within `<KnowledgeData>` tag. The tags `<Tag>` and `<Credit>` store the category of the topic discussed in a thread and reputation for a post, respectively. Extra information like, editor's biography, badge details, etc., can be represented using the extension mechanism. All the QnA based knowledge building portals maintain the basic structure of threaded questions and answers, making it feasible to represent in Knol-ML format.

1.4 `cElementTree` comparision

We have used C programming language based `cElementTree`[5] library as the basic tool to parse Knol-ML based knowledge data. `cElementTree` is a C implementation of `ElementTree` module, which makes it more efficient in terms of parsing time and memory usage. It outperforms most of the libraries in both the parameters (see supplimentry material Appendix 1.3).

We used popular XML toolkit to parse a 3119 KB Wikipedia XML dump of the article *1957 Pacific hurricane season* and compared the results. Table 1 shows the benchmark figures.

`cElementTree` represents the XML document as a tree, where each element is a node. Sub-elements of an element are represented as children of the node in the tree. Each node of the tree also maintains a hash table to store the attributes of this element, where key is

[5]https://docs.python.org/2/library/xml.etree.elementtree.html

Table 1. Execution Time of Various Parsers

Libraries	**Time**	**Space**
xml.dom.minidom	0.06 s	7768 KB
lxml.etree	0.09 s	16488 KB
ElementTree 1.2.4/1.3	0.049 s	8128 KB
PyRXPU	0.1 s	19936 KB
cElementTree	**0.040 s**	**4018 KB**

the attribute name and value is the data stored within this attribute. This allows KDAP methods to access the attributes associated with each element in constant time.