

# EUE: Development of a Hybrid Course Infrastructure and its Application in CS340: Algorithms and Data Structures



Live at:

<http://edit.cs.siu.edu:3000/>

Mark McKenney [marmkce@cs.siu.edu]  
Department of Computer Science  
Southern Illinois University Edwardsville



## Goals:

1. Real time, collaborative editing.
2. Simple interface.
3. Web based
4. Usable on computers and mobile devices in real time in class to create and demonstrate code.

## Easy Editor provides:

1. Log in with Facebook credentials.
2. Write, compile, and execute python code in the browser.
3. Share files with students.
4. Collaboratively create/edit code in real time.
5. Debug code as a group in class.

The screenshot shows the Easy Editor web interface. At the top, there are tabs for 'test', 'Editor', 'Documents', and 'Group'. Below the tabs, there are controls for creating a new file, opening an existing file, creating a link, and sharing the document. The main editor area contains Python code for a loop that prints 'hi there' and 'this loop has run ' followed by a range of numbers, and then 'times!'. Below the code, there are 'Run' and 'Clear' buttons. The output area shows the execution results, including a NameError: name 'asdfsadg' is not defined on line 12.

The screenshot shows the Easy Editor web interface. At the top, there are tabs for 'test', 'Editor', and 'Documents'. Below the tabs, there are controls for creating a new file, opening an existing file, creating a link, and sharing the document. The main editor area contains Python code using the turtle module to draw a square. Below the code, there are 'Run' and 'Clear' buttons. The output area shows the execution results, including a drawing of a square with a blue cursor at the bottom-left corner.



# EUE: Development of a Hybrid Course Infrastructure and its Application in CS340: Algorithms and Data Structures

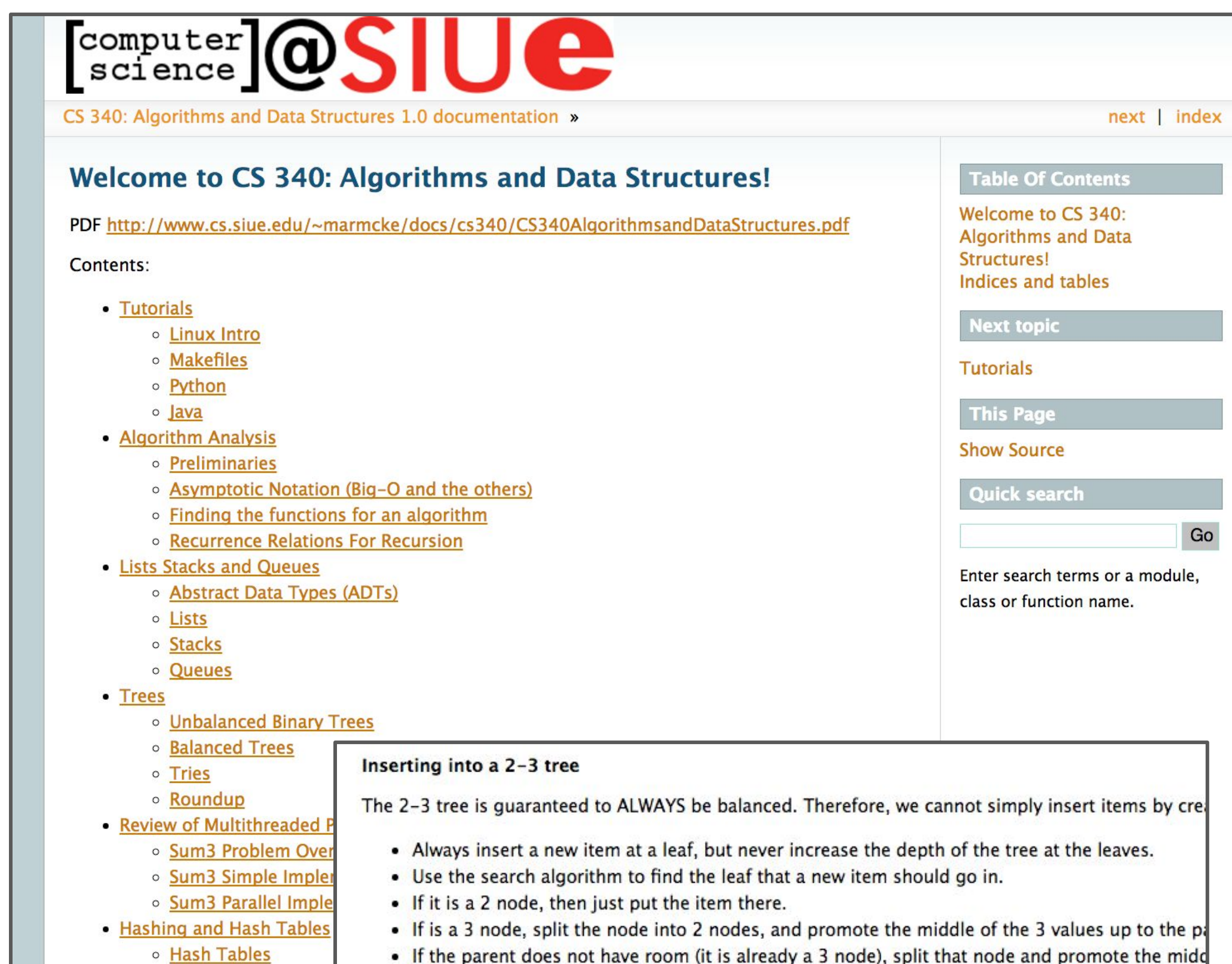
Mark McKenney [marmkce@cs.siu.edu]  
Department of Computer Science  
Southern Illinois University Edwardsville

## Goals:

1. Create course materials quickly.
2. Ability to include more in depth information than is appropriate on Powerpoint.
3. Searchable content. (its hard to find something specific in a folder full of Powerpoints)
4. Easily create Web and PDF formats.

## Sphinx provides:

1. Text based format format for content creation.
2. Multiple output formats:
  - a. Searchable website
  - b. Book in PDF
  - c. Ebook
3. Integrates easily with version control.



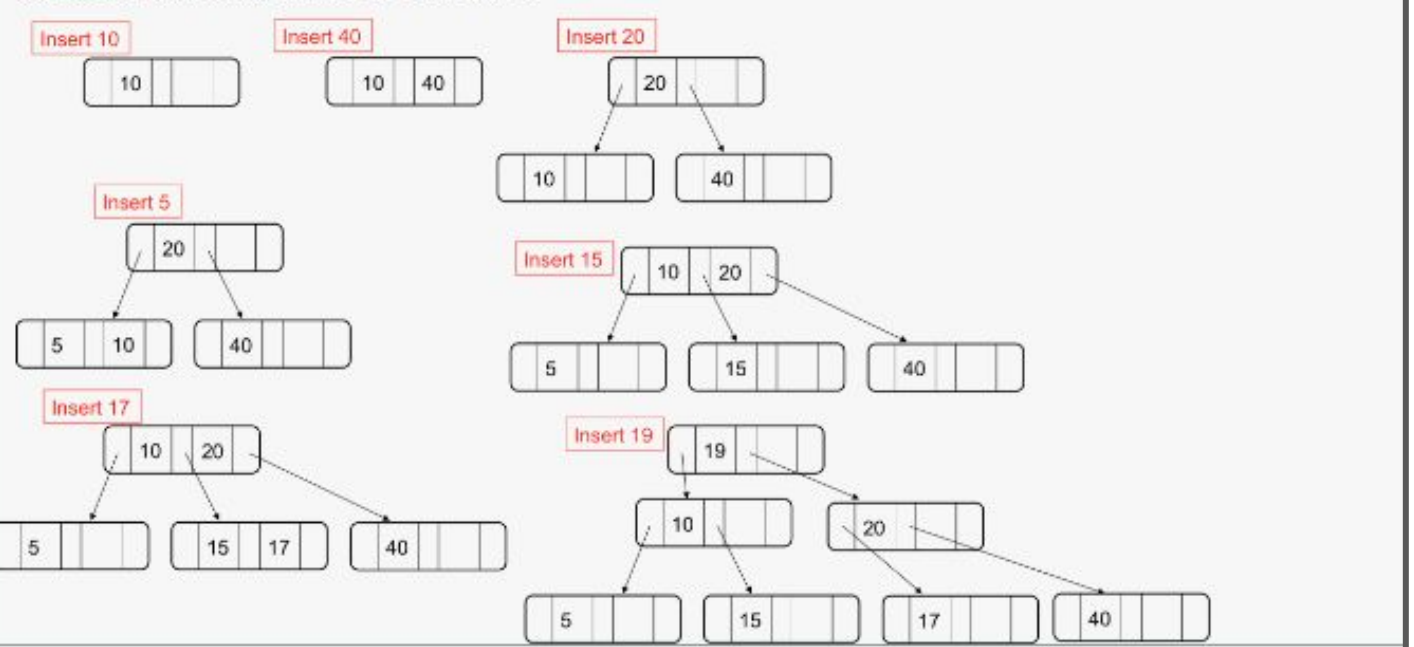
### Inserting into a 2-3 tree

The 2-3 tree is guaranteed to ALWAYS be balanced. Therefore, we cannot simply insert items by creating new leaves like we do with a BST. The process is as follows:

- Always insert a new item at a leaf, but never increase the depth of the tree at the leaves.
- Use the search algorithm to find the leaf that a new item should go in.
- If it is a 2 node, then just put the item there.
- If it is a 3 node, split the node into 2 nodes, and promote the middle of the 3 values up to the parent node.
- If the parent does not have room (it is already a 3 node), split that node and promote the middle value to its parent. Recursively repeat until no splitting occurs.

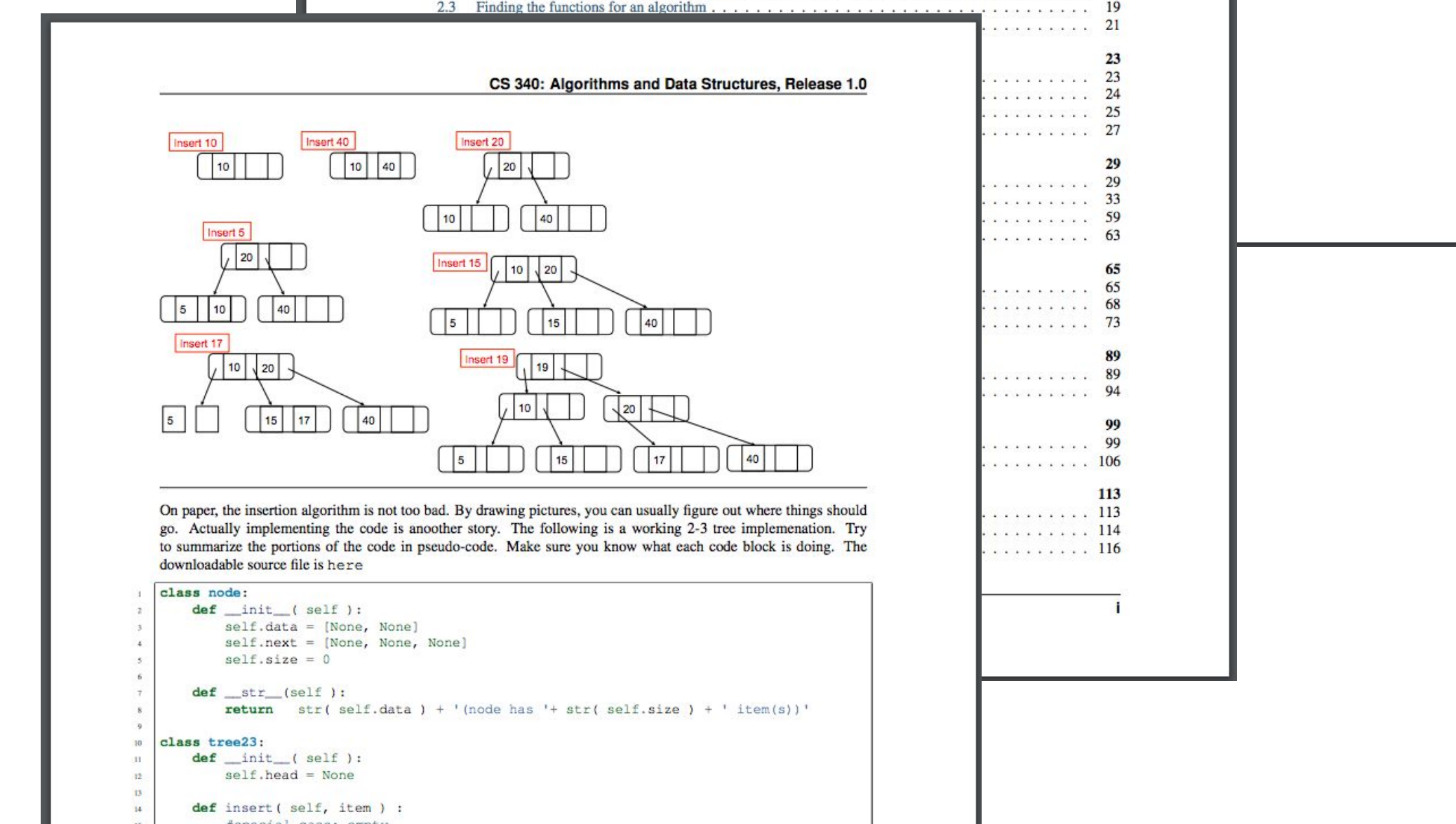
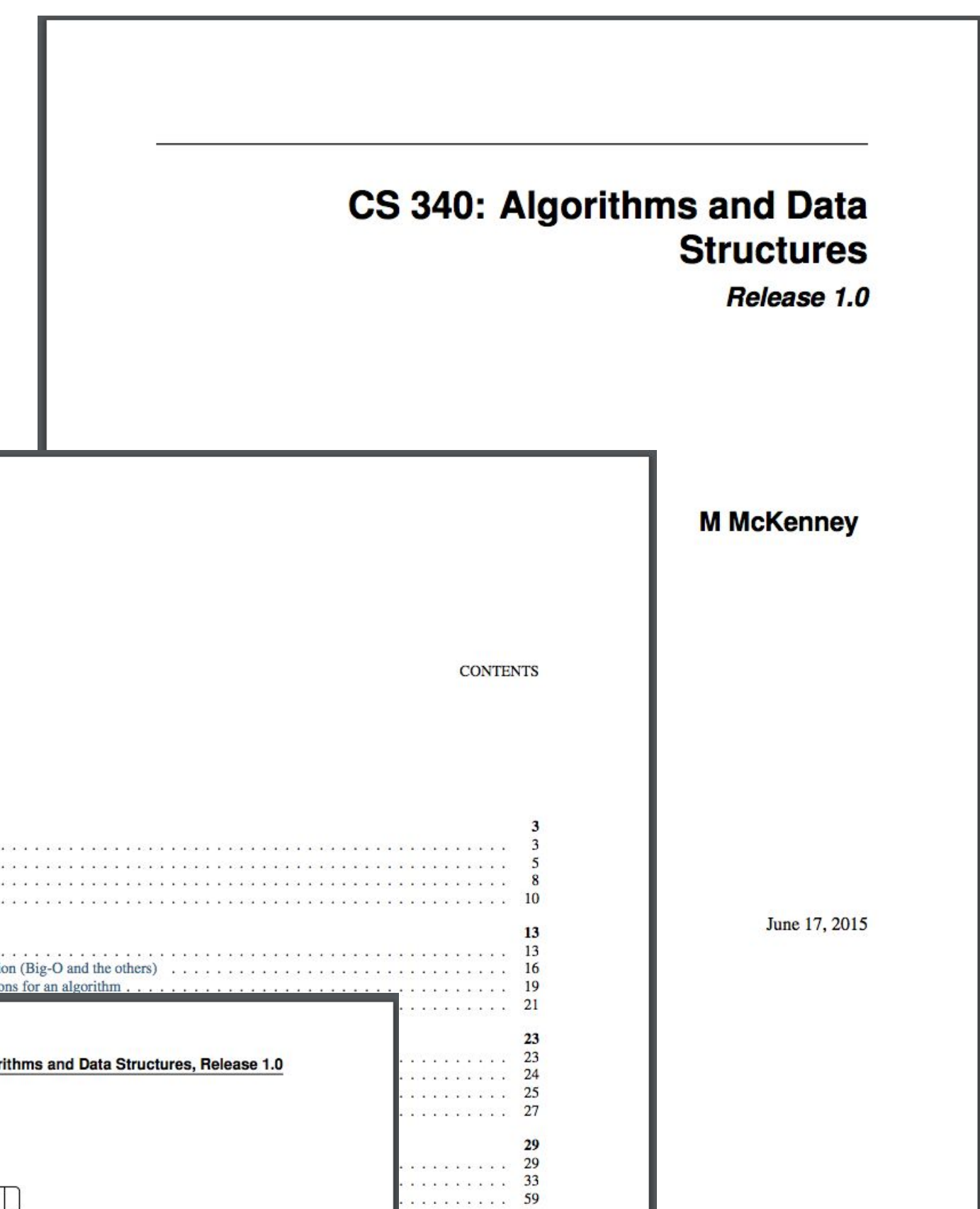
### Example 2-3 Tree Insertion

Insert the following: 10 40 20 5 15 17 19



On paper, the insertion algorithm is not too bad. By drawing pictures, you can usually figure out where things should go. Actually implementing the code is another story. The following is a working 2-3 tree implementation. Try to summarize the portions of the code in pseudo-code. The downloadable source file is [here](#)

```
1 class node:
2     def __init__( self ):
3         self.data = [None, None]
4         self.next = [None, None, None]
5         self.size = 0
6
7     def __str__(self):
8         return str( self.data ) + '(node has ' + str( self.size ) + ' item(s))'
```



```
227 **Inserting into a 2-3 tree**
228
229 The 2-3 tree is guaranteed to ALWAYS be balanced. Therefore, we cannot simply insert items by creating new leaves like we do with a BST. The process is as follows:
230
231 * Always insert a new item at a leaf, but never increase the depth of the tree at the leaves.
232
233 * Use the search algorithm to find the leaf that a new item should go in.
234
235 * If it is a 2 node, then just put the item there.
236
237 * If it is a 3 node, split the node into 2 nodes, and promote the middle of the 3 values up to the parent node.
238
239 * If the parent does not have room (it is already a 3 node), split that node and promote the middle value to its parent. Recursively repeat until no splitting occurs.
240
241 If this procedure is followed, the height of the tree will be increased at the TOP of the tree (the root). Basically, tree height will increase once a root is split. An example follows:
242
243 .. admonition:: Example 2-3 Tree Insertion
244
245     Insert the following: 10 40 20 5 15 17 19
246
247     .. image:: zstatic/t23Insert.*
248         :width: 6in
249
250
251 On paper, the insertion algorithm is not too bad. By drawing pictures, you can usually figure out where things should go. Actually implementing the code is another story. The following is a working 2-3 tree implementation. Try to summarize the portions of the code in pseudo-code. Make sure you know what each code block is doing. The downloadable source file is :download: `here <zstatic/t23.py>`
252
253 .. literalinclude:: zstatic/t23.py
254     :language: python
255     :linenos:
256
```