1. An "abstract" view of the stack:

Using an array implementation would look something like:

```
| items: | a | b | c | (a) |
| top:   | 2 |
| max:   | 100 |
```

Complete the big-oh notation for the following stack methods assuming an array implementation: ("n" is the # items)

<table>
<thead>
<tr>
<th>Big-oh</th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
<th>isFull()</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

2. Since Python does not have a (directly accessible) built-in array, we can use a list.

```python
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

Since Python uses an array of references (pointers) to list items in their implementation of a list.

"Abstract" Stack | Stack Object | list Object
|--------|------------|--------|

```
|        |          |        |
| a      | c        | 0 1 2 3 |
| b      |          | a b c d |
|        | bottom   |        |
```

a) Complete the big-oh notation for the stack methods assuming this Python list implementation: ("n" is the # items)

<table>
<thead>
<tr>
<th>Big-oh</th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td></td>
</tr>
</tbody>
</table>

b) Which operations should have what preconditions?

$\text{pop } \& \text{ peek: } \text{Stack cannot be empty}$
3. The text's alternative stack implementation also using a Python list is:

```python
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.insert(0, item)

    def pop(self):
        return self.items.pop(0)

    def peek(self):
        return self.items[0]

    def size(self):
        return len(self.items)
```

Since an array is used to implement a Python list, the alternate Stack implementation using a list:

<table>
<thead>
<tr>
<th>Abstract Stack</th>
<th>&quot;alternate&quot; Stack Object</th>
<th>list Object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>top</td>
<td>items: [ ]</td>
<td></td>
</tr>
<tr>
<td>bottom</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a) Complete the big-oh notation for the "alternate" Stack methods: ("n" is the # items)

<table>
<thead>
<tr>
<th>Big-oh</th>
<th>push(item)</th>
<th>pop()</th>
<th>peek()</th>
<th>size()</th>
<th>isEmpty()</th>
<th><strong>init</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

4. How could we use a stack to check if a word is a palindrome (e.g., radar, toot)?

5. How could we check to see if we have a balanced string of nested symbols? ("(()(()())())")
1. The Node class (in node.py) is used to dynamically create storage for a new item added to the stack. The LinkedStack class (in linked_stack.py) uses this Node class. Conceptually, a LinkedStack object would look like:

```
class Node:
    def __init__(self, initdata):
        self.data = initdata
        self.next = None
    def getData(self):
        return self.data
    def getNext(self):
        return self.next
    def setData(self, newdata):
        self.data = newdata
    def setNext(self, newnext):
        self.next = newnext
```

a) Complete the push, pop, and __str__ methods.

b) Stack methods big-oh's?
   (Assume "n" items in stack
   
   • constructor __init__:
   • push(item):
   • pop():
   • peek():
   • size():
   • isEmpty():
   • __str__():

```python
class LinkedStack(object):
    """ Link-based stack implementation. """
    def __init__(self):
        self._top = None
        self._size = 0
    def push(self, newItem):
        """ Inserts newItem at top of stack. """
        temp = Node(newItem)
        temp.setNext(self._top)
        self._top = temp
        self._size += 1
    def pop(self):
        """ Removes and returns the item at top of the stack. Precondition: the stack is not empty. """
        return self._top.getData()
    def peek(self):
        """ Returns the item at top of the stack. Precondition: the stack is not empty. """
        return self._top.getData()
    def size(self):
        """ Returns the number of items in the stack. """
        return self._size
    def isEmpty(self):
        return self._size == 0
    def __str__(self):
        """ Items strung from top to bottom. """
```

Steps for implementing "Linked" method "push"

1. Draw "normal-case" picture
   stuff in it already

2. Update picture for method

3. Number steps

4. Write "normal-case" code

5. Consider special cases:
   - empty stack
   - draw picture
   - run normal-case code
   - adjust as needed

\[\text{top} \quad \text{size}\]