Question 1. (4 points) Consider the following Python code.

```python
def main(n):
    for i in range(n):
        for j in range(n*n):
            print (i, j)
```

What is the big-oh notation $O(\ )$ for this code segment in terms of $n$?

Question 2. (4 points) Consider the following Python code.

```python
i = 1
while i < n:
    for j in range(n):
        print(j)
    i = i * 2
```

What is the big-oh notation $O(\ )$ for this code segment in terms of $n$?

Question 3. (4 points) Consider the following Python code.

```python
def main(n):
    for i in range(n):
        doSomething(n)
        doMore(n)

def doSomething(n):
    for k in range(n):
        print(k)

def doMore(n):
    for k in range(n):
        print(k)
```

What is the big-oh notation $O(\ )$ for this code segment in terms of $n$?

Question 4. (8 points) Suppose a $O(n^3)$ algorithm takes 10 seconds when $n = 1,000$. How long would you expect the algorithm to run when $n = 10,000$?

Question 5. (5 points) Why should any method/function having a "precondition" raise an exception if the precondition is violated?
Question 6. Consider the following FIFO (First-In-First-Out) Queue implementation utilizing a Python list:
Recall that a queue is a linear data structure that allows adding new items at the rear and removing items from the front. One possible implementation of a queue would be to use a built-in Python list to store the items such that

- the **rear** item is **always stored at index 0**.
- the **front** item is always at index \( \text{len(self._items)} - 1 \) or -1

a) (8 points) Complete the big-oh \( O() \), for each Queue operation, assuming the above implementation. Let \( n \) be the number of items in the Queue.

<table>
<thead>
<tr>
<th>Operation</th>
<th>( O() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEmpty</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>enqueue</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>dequeue</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>size</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

b) (10 points) Complete the method for the dequeue operation including the precondition check.

```python
def dequeue(self):
    """Removes and returns the front item of the queue
    Precondition: the queue is not empty.
    Postcondition: front item is removed from the queue and returned""
```

c) (7 points) Suggest an alternate Queue implementation to speed up some of its operations.
Question 7. Consider the binary heap approach to implement a priority queue. A Python list is used to store a complete binary tree (a full tree with any additional leaves as far left as possible) with the items being arranged by heap-order property, i.e., each node is ≤ either of its children. An example of a min heap “viewed” as a complete binary tree would be:

```
9
[1]
[2] 15
[3] 10
[4] 114
[5] 20
[6] 30
[7] 50
[8] 300
[9] 125
[10] 117
```

Python List actually used to store heap items

```
[not used, 9, 15, 10, 114, 20, 30, 50, 300, 125, 117]
```

a) (3 points) For the above heap, the list indexes are indicated in [ ]'s. For a node at index i, what is the index of:
• its left child if it exists:
• its right child if it exists:
• its parent if it exists:

b) (7 points) What would the above heap look like after inserting 12 and then 25 (show the changes on above tree)

c) (3 points) What is the big-oh notation for inserting a new item in the heap?

Now consider the delMin operation that removes and returns the minimum item.

```
9
[1]
[2] 15
[3] 10
[4] 114
[5] 20
[6] 30
[7] 50
[8] 300
[9] 125
[10] 117
```

Python List actually used to store heap items

```
[not used, 9, 15, 10, 114, 20, 30, 50, 300, 125, 117]
```

d) (2 point) What item would delMin remove and return from the above heap?

e) (7 points) What would the above heap look like after delMin? (show the changes on above tree)

f) (3 points) What is the big-oh notation for delMin?
Question 8. The textbook’s unordered list ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:

UnorderedList Object

a) (15 points) The `insert(position, item)` method adds the item to the list at the specified position. Unlike the textbook’s implementation, ASSUME that the list may contain duplicate items!!! The precondition is that `position` is a nonnegative integer. If `position` is 0, then add it to the head of the list. If `position` is `_size` or bigger, then add it to the tail of the list. Complete the `insert(position, item)` method code including the precondition check.

```python
class UnorderedList:
    def __init__(self):
        self._head = None
        self._size = 0
        self._tail = None

    def insert(self, position, item):
        # Method code implementation
```

b) (10 points) Assuming the unordered list ADT described above that allows duplicate items. Complete the big-o O( ) for each operation. Let n be the number of items in the list.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>insert(position, item)</code></td>
<td>O(n)</td>
</tr>
<tr>
<td><code>pop()</code></td>
<td>O(1)</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>O(1)</td>
</tr>
<tr>
<td><code>append(item)</code></td>
<td>O(n)</td>
</tr>
<tr>
<td><code>add(item)</code></td>
<td>O(n)</td>
</tr>
</tbody>
</table>