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

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

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
for i in range(n):
    for j in range(n):
        print(j)
    for k in range(n):
        print(k)
```

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)

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

def doMore(n):
    for j in range(n):
        print(j)

main(n)
```

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

Question 4. (8 points) Suppose an \(O(n^3)\) algorithm takes 10 second when \(n = 100\). How long would the algorithm run when \(n = 1,000\)?

Question 5. (10 points) Why should any method/function having a "precondition" raise an exception if the precondition is violated?
Question 6. A Deque (pronounced “Deck”) is a linear data structure which behaves like a double-ended queue, i.e., it allows adding or removing items from either the front or the rear of the Deque. One possible implementation of a Deque would be to use a built-in Python list to store the Deque items such that

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

\[ a \]  \[ b \]  \[ c \]  \[ d \] 
\[ 0 \]  \[ 1 \]  \[ 2 \]  \[ 3 \] 

_a_ items: Python List Object

**Deque Object**

**Python List Object**

\[ 'a' \]  \[ 'b' \]  \[ 'c' \]  \[ 'd' \]
front  rear

_a) (6 points) Complete the big-oh \( O(\ )\), for each Deque operation, assuming the above implementation. Let \( n \) be the number of items in the Deque.

<table>
<thead>
<tr>
<th>operation</th>
<th>( O() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEmpty</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>addRear</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>removeRear</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>addFront</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>removeFront</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>size</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>

**b) (9 points) Complete the method for the removeFront operation, including the precondition check to raise an exception if it is violated.**

```python
def removeFront(self):
    """Removes and returns the Front item of the Deque
    Precondition: the Deque is not empty.
    Postcondition: Front item is removed from the Deque and returned"
```

**c) (5 points) An alternate Deque implementation would swap the location of the front and rear items as in:**

\[ d \]  \[ c \]  \[ b \]  \[ a \] 
\[ 0 \]  \[ 1 \]  \[ 2 \]  \[ 3 \] 

Deque Object

Python List Object

**Deque Object**

**Python List Object**

Why is this alternate implementation probably not very helpful with respect to the Deque’s performance?
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:

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 18 and then 9 (show the changes on above tree)?

c) (6 points) What is the big-oh notation for the insert operation? (EXPLAIN YOUR ANSWER)

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

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)
Question 8. The Node2Way class (which inherits the node.py class) can be used to dynamically create storage for each new item added to a Deque using a doubly-linked implementation as in:

```
DoublyLinkedDeque Object

_size: 4
_front: __________ previous data next __________ previous data next __________ previous data next __________ previous data next
_rear: __________ 'a' __________ 'b' __________ 'c' __________ 'd'

Node2Way Objects

previous data   next previous data   next previous data   next previous data   next
'a'  'b'  'c'  'd'
```

a) (6 points) Determine the big-oh, $O(\cdot)$, for each Deque operation assuming the above doubly-linked implementation. Let $n$ be the number of items in the Deque.

<table>
<thead>
<tr>
<th>Operation</th>
<th>$O(\cdot)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFront</td>
<td></td>
</tr>
<tr>
<td>removeFront</td>
<td></td>
</tr>
<tr>
<td>addRear</td>
<td></td>
</tr>
<tr>
<td>removeRear</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td></td>
</tr>
<tr>
<td><strong>str</strong></td>
<td></td>
</tr>
</tbody>
</table>

b) (14 points) Complete the addRear method.

```python
class DoublyLinkedDeque(object):
    """ Doubly-Linked list based Deque implementation."
    def __init__(self):
        self._size = 0
        self._front = None
        self._rear = None
    def addRear(self, newItem):
        """ Adds the newItem to the rear of the Deque.
        Precondition: none """
```

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

class Node2Way(Node):
    def __init__(self, initdata):
        Node.__init__(self, initdata)
        self.previous = None
    def getPrevious(self):
        return self.previous
    def setPrevious(self, newprevious):
        self.previous = newprevious
```

c) (5 points) Would using singly-linked nodes (i.e., Node objects instead of Node2Way) slow down any of the Deque operations? Justify your answer