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

```python
for i in range(n):
    j = 1
    while j < n:
        for k in range(n):
            print( i, j, k)
        j = j * 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)
    k = n
    while k > 0:
        print(k)
        k = k // 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 j in range(n * n * 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 a $O( n^4 )$ algorithm takes 10 second when $n = 100$. How long would you expect the algorithm to run when $n = 1,000$?

Question 5. (10 points) Why should you design a program instead of “jumping in” and start by writing code?
Question 6. Consider the following Stack implementation utilizing a Python list:

a) (6 points) Complete the big-oh notation for the Stack methods assuming the above implementation: ("n" is the # items)

<table>
<thead>
<tr>
<th>Method</th>
<th>Big-oh</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(item)</td>
<td></td>
</tr>
<tr>
<td>pop()</td>
<td></td>
</tr>
<tr>
<td>peek()</td>
<td></td>
</tr>
<tr>
<td>size()</td>
<td></td>
</tr>
<tr>
<td>isEmpty()</td>
<td></td>
</tr>
<tr>
<td><strong>str</strong></td>
<td></td>
</tr>
</tbody>
</table>

b) (9 points) Complete the code for the `pop` method including the precondition check.

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

    def pop(self):
        """Removes and returns the top item of the stack
        Precondition: the stack is not empty.
        Postcondition: the top item is removed from the stack and returned"
```

c) (5 points) Suggest an alternate Stack 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:

- 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)?

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

c) (2 points) What item would delMin remove and return from the above heap?

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

e) (6 points) What is the big-oh notation for the delMin operation? (EXPLAIN YOUR ANSWER)
Question 8. The `Node` class (in `node.py`) is used to dynamically create storage for a new item added to the stack. Consider the following `LinkedQueue` class using this `Node` class. Conceptually, a `LinkedQueue` object would look like:

```
"Abstract Queue"

'w'  'x'  'y'

front  rear

LinkedQueue Object

_data_  _next_

数据  数据  数据  下一个  下一个  下一个

Node Objects

'w'  'x'  'y'

front  rear
```

a) (13 points) Complete the `dequeue` method including the precondition check.

```python
class LinkedQueue(object):
    """ Linked-list based queue implementation."""
    def __init__(self):
        self._front = None
        self._size = 0
        self._rear = None
    def dequeue(self):
        """ Removes and returns the front item in the queue.
        Precondition: the queue is not empty. """
        ...  
```

b) (7 points) Assuming the queue ADT described above. Complete the big-oh $O(\cdot)$ for each queue operation. Let $n$ be the number of items in the queue.

```
<table>
<thead>
<tr>
<th><strong>init</strong></th>
<th>enqueue(item)</th>
<th>dequeue()</th>
<th>size()</th>
<th><strong>str</strong>()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

c) (5 points) Would using doubly-linked nodes (i.e., `Node2way`) speed up some of the queue operations? Justify your answer.