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

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

\[ O(n^2 \log n) \]
```

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)
    for k in range(n):
        print(k)
    i = i + 2

\[ O(n^2 \log n) \]
```

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 k in range(n):
        print(k)

\[ O(n^2) \]
```

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

Question 4. (8 points) Suppose a \( O(n^5) \) algorithm takes 10 seconds when \( n = 100 \). How long would you expect the algorithm to run when \( n = 1,000 \)?

\[
T(n) = cn^5 \\
T(100) = c(100)^5 = 10^5\text{sec} \\
T(1000) = c(1000)^5 = 10^5\times c \\
= 10^{-9}\text{sec} \\
= 10^6\text{sec}
\]

Question 5. (5 points) Why should you design a program instead of “jumping in” and start by writing code?

You will make fewer mistakes working from a design.
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 rear item is always stored at index 0,
- the front item is always at index len(self._items)-1 or -1

<table>
<thead>
<tr>
<th>Deque Object</th>
<th>Python List Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>_items:</td>
<td>'a' 'b' 'c' 'd'</td>
</tr>
<tr>
<td>rear</td>
<td>front</td>
</tr>
</tbody>
</table>

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>$O$-notation</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>$O(1)$</td>
</tr>
<tr>
<td>size</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

b) (9 points) Complete the method for the `removeRear` operation including the precondition check.

```python
def removeRear(self):
    """Removes and returns the rear item of the Deque
    Precondition: the Deque is not empty.
    Postcondition: Rear item is removed from the Deque and returned"
    
    if len(self._items) == 0:
        raise ValueError("Cannot remove from empty Deque")
    return self._items.pop(0)
```

c) (5 points) Suggest an alternate Deque implementation to speed up some of its operations.

"""A doubly-linked list implementation."""
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:

Pyton List actually used to store heap items

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>18</td>
<td>13</td>
<td>45</td>
<td>23</td>
<td>30</td>
<td>50</td>
<td>200</td>
<td>51</td>
<td>77</td>
<td>61</td>
<td></td>
</tr>
</tbody>
</table>

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? \( O(\log_2 n) \)

Now consider the \texttt{delMin} operation that removes and returns the minimum item.

Pyton List actually used to store heap items

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>18</td>
<td>13</td>
<td>45</td>
<td>23</td>
<td>30</td>
<td>50</td>
<td>200</td>
<td>51</td>
<td>77</td>
<td>61</td>
<td></td>
</tr>
</tbody>
</table>

d) (2 points) What item would \texttt{delMin} remove and return from the above heap? 7

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

f) (3 points) What is the big-oh notation for \texttt{delMin}? \( O(\log_2 n) \)
Question 8. The textbook’s **Ordered list** ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:

![Diagram of OrderedList Object]

a) (15 points) The `add(item)` method adds the `item` to the list. Recall that the textbook’s implementation, cannot contain duplicate items!! Thus, the precondition is that `item` is a not already in the list. Complete the `add(item)` method code including the precondition check.

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

    def add(self, item):
        temp = Node(item)
        prev = None
        current = self._head
        while current != None:
            if current.getData() == item:
                raise ValueError("Cannot add duplicates")
            if current.getData() > item:
                break
            prev = current
            current = current.getNext()
        if prev == None:
            temp.setNext(self._head)
            self._head = temp
        else:
            temp.setNext(current)
            self._size += 1
```

b) (10 points) Assuming the ordered list ADT described above does not allows duplicate items. Complete the big-oh `O()` for each operation. Let `n` be the number of items in the list.

<table>
<thead>
<tr>
<th>Operation</th>
<th><code>O()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(item)</code></td>
<td><code>O(n)</code></td>
</tr>
<tr>
<td><code>pop()</code></td>
<td><code>O(n)</code></td>
</tr>
<tr>
<td><code>length()</code></td>
<td><code>O(1)</code></td>
</tr>
<tr>
<td><code>remove(item)</code></td>
<td><code>O(n)</code></td>
</tr>
<tr>
<td><code>index(item)</code></td>
<td><code>O(n)</code></td>
</tr>
</tbody>
</table>