Data Structures - Test 1

Question 1. (5 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
O(n^2 log_2 n)
```

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

\( (+3 n \log_2 n) \)

Question 2. (5 points) Consider the following Python code.

```python
i = 2**n  # this is 2^n
while i > 1:
    for j in range(n):
        print(j)
    i = i // 2
O(n^2)
```

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

\( (+3 n \log_2 n) \)

Question 3. (5 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):
        print(k)
main(n)
\( O(n^2) \)
```

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

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

\[
T(n) = c \cdot n^4 \quad T(1000) = c \cdot 1000^4 = 10 \text{ sec} \quad c = \frac{10 \text{ sec}}{1000^4} = 10^{-11} \text{ sec}
\]

\[
T(10,000) = c \cdot 10000^4 = 10^{-11} \text{ sec} \cdot 10000^4 = 10^{-11} \text{ sec} \cdot 10^5 = 10^4 \text{ sec}
\]

\[
= 10000 \text{ sec} = 27.78 \text{ hrs}
\]

Question 5. (10 points) Why should you design a program instead of “jumping in” and start writing code?

By designing a program you can split the work into smaller, manageable pieces. It is easier to implement/code and debug/test the smaller pieces.
Question 6. Consider the following Stack implementation utilizing a Python list:

<table>
<thead>
<tr>
<th>Abstract Stack</th>
<th>Stack Object</th>
<th>Python list Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>items:</td>
<td>0 1 2</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>c b a</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>top bottom</td>
</tr>
<tr>
<td>a</td>
<td></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)

|         | push(item) | pop()   | peek()  | size()  | isEmpty() | _init_
|---------|------------|---------|---------|---------|-----------|--------
| Big-oh  | O(n)       | O(n)    | O(1)    | O(1)    | O(1)      | O(1)   |

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

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"
    if len(self._items) == 0:
        raise ValueError("cannot pop from empty stack")
    return self._items.pop(0)

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

Reverse the top and bottom, i.e., let index 0 hold the bottom item of the stack.
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” on a complete binary tree would be:

Python List actually used to store heap items

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: \( \lfloor \frac{i + 1}{2} \rfloor \)
   - its right child if it exists: \( \lfloor \frac{i + 2}{2} \rfloor \)
   - its parent if it exists: \( \lfloor \frac{i}{2} \rfloor \)

b) (6 points) What would the above heap look like after inserting 35 and then 12 (show the changes on above tree)

c) (2 points) What is the big-oh notation for inserting a new item in the heap? \( O(\log n) \)

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

Python List actually used to store heap items

d) (1 point) What item would \texttt{delMin} remove and return from the above heap? \( 10 \)

e) (6 points) What would the above heap look like after \texttt{delMin}? (show the changes on above tree)

f) (2 points) What is the big-oh notation for \texttt{delMin}? \( O(\log n) \)
Question 8. The textbook’s ordered list ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:

```
OrderedList Object

_data_ _next_ _data_ _next_ _data_ _next_
'b'     'e'     'm'     'v'
```

a) (15 points) The `pop(position)` method removes and returns the item at the specified position. The precondition is that position is a nonnegative integer corresponding to an actual list item (e.g., for the above list `0 ≤ position ≤ 3`). Complete the `pop(position)` method code including the precondition check.

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

    def pop(self, position):
        if not isinstance(position, int):
            raise TypeError("Position must be an integer")

        if position < 0 or position > self._size:
            raise IndexError("Position outside value range")

        previous = None
        current = self._head
        for count in range(position):
            previous = current
            current = current.get_next()

        if previous == None:
            self._head = self._head.get_next()
        else:
            previous.set_next(current.get_next())

        self._size -= 1
        return current, get_data()
```

b) (10 points) Assuming the ordered list ADT described above. 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>Big-Oh</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop(position)</td>
<td>(O(n))</td>
</tr>
<tr>
<td>pop()</td>
<td>(O(n))</td>
</tr>
<tr>
<td>length()</td>
<td>(O(1))</td>
</tr>
<tr>
<td>index(item)</td>
<td>(O(n))</td>
</tr>
<tr>
<td>add(item)</td>
<td>(O(n))</td>
</tr>
</tbody>
</table>