This quiz has 3 questions, for a total of 30 points.

1. **10 points** Recall the Disjoint Sets (aka. union-find) data structure which maintains a partitioning of elements, where each partition is represented as a tree. The root of the tree is the representative of the partition and each element has a pointer to its parent in the tree. The make_set function places an element in a partition by itself. The find function returns the representative of a given element. The union function merges two partitions into a single partition. Implement the basic version of these functions in Python (no need to use path compression or union-by-range).

   ```python
   def make_set(elt):
       elt.parent = elt

   def find(elt):
       if elt is elt.parent:
           return elt
       else:
           return find(elt.parent)

   def union(elt1, elt2):
       elt2.parent = elt1
   ```

   **Solution:**

   ```python
   def make_set(elt):
       # 3 points
       elt.parent = elt

   def find(elt):
       # 4 points
       if elt is elt.parent:
           return elt
       else:
           return find(elt.parent)

   def union(elt1, elt2):
       # 3 points
       elt2.parent = elt1
   ```

2. **10 points** Draw a B-tree of minimum degree $t = 2$ that represents the set of integers $\{1,2,3,4,5\}$. 

   ```text
   # Draw B-tree representation
   ```
There are several correct answers. Every node has at most 3 keys. Every node other than the root must have 1 or more keys. The B-tree needs to obey the ordering constraints for the keys (like a Binary Search Tree). Here are some examples:

\[ \begin{array}{c}
\text{1,2,3} \\
\text{2,4} \\
\text{3} \\
\text{1,2,5} \\
\end{array} \]

3. [10 points] Suppose you implement a queue (first-in-first-out) using two stacks (last-in-first-out) as follows.

```python
class QueueStack:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, elt):
        self.stack1.append(elt)

    def dequeue(self):
        assert (len(self.stack1) + len(self.stack2)) != 0
        if len(self.stack2) == 0:
            while len(self.stack1) != 0:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop()
```

What is the amortized time complexity of a sequence of \( n \) operations on this queue? Each operation can be an enqueue or dequeue operation, though for a dequeue, the queue must not be empty.

**Solution:** For each element that is enqueued, there is only ever a constant amount (4 operations) of work done to it: 1) push to stack 1, 2) pop from stack 1, 3) push on stack 2, 4) pop from stack 2. So the total cost of the \( n \) operations is at most \( 4n \), and thus, the amortized time complexity of one operation is \( O(1) \).