J. Case Study 10: Amortized Analysis - Queue Implementation

```
class AmortizedQueue:
    def _init_(self):
        self.queue = []
        self.front = 0  # Tracks the front of the queue
        self.total_operations = 0
        self.total_cost = 0

    def enqueue(self, value):
        self.queue.append(value)
        self.total_operations += 1
        self.total_cost += 1  # O(1) cost

    def dequeue(self):
        if self.front == len(self.queue):
            print("Queue is empty!")
            return None
        value = self.queue[self.front]
        self.front += 1
        self.total_operations += 1
        self.total_cost += 1  # O(1) cost
        # Compact if too much unused space at the front
        if self.front > len(self.queue) // 2:
            self.compact()
```

```
        return value

    def compact(self):
        self.queue = self.queue[self.front:]
        self.front = 0
        self.total_operations += 1
        self.total_cost += len(self.queue)  # O(n) cost

    def get_amortized_cost(self):
        return self.total_cost / self.total_operations if self.total_operations else 0

# Example usage:
queue = AmortizedQueue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print(queue.dequeue())  # 1
print(queue.dequeue())  # 2
queue.enqueue(4)
queue.enqueue(5)
print(queue.dequeue())  # 3
print(queue.get_amortized_cost())  # Should be close to O(1)
```