Question:1 Implement Stack using Python.

```python
class Stack:
    stack = []
    def insert(self, num):
        self.stack.append(num)

    def remove(self):
        try:
            self.stack.pop()
        except:
            raise "Something went wrong"
    def top(self):
        return self.stack[-1:]
    def print_stack(self):
        for num in self.stack:
            print(num, end = " ")
        print()
```

Question:2 Implement Queue using Python.

```python
class Queue:
    queue = []

    def enqueue(self, element):
        self.queue.append(element)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.pop(0)
        else:
            raise IndexError("dequeue from empty queue")

    def top(self):
        if not self.is_empty():
            return self.queue[0]
        else:
            raise IndexError("top from empty queue")

    def is_empty(self):
        return len(self.queue) == 0

    def size(self):
        return len(self.queue)
```

Question 3: Binary Search

```python
def binary_search(nums, element):
    nums.sort()
    start = 0
    end = len(nums) - 1
    while not(start >= end):
        mid = start + (end - start) // 2
        if(nums[mid] == element):
            return True
        elif (nums[mid] > element):
            end = mid - 1
            continue
        elif (nums[mid] < element):
            start = mid + 1
            continue

    return False

nu = [1,2,3,4,5,6,7,8]
print(binary_search(nu, 10))
print()
print()
print()
print()
print()
#Code to test Stack and Queue
print("--- Testing Stack ---")
my_stack = Stack()
print(f"Initial stack: ", end="")
my_stack.print_stack()
print(f"Is stack empty? {my_stack.stack == []}")

my_stack.insert(10)
my_stack.insert(20)
my_stack.insert(30)
print(f"Stack after insertions: ", end="")
my_stack.print_stack()
print(f"Top element: {my_stack.top()}")

my_stack.remove()
print(f"Stack after removal: ", end="")
my_stack.print_stack()
print(f"Top element after removal: {my_stack.top()}")

try:
    my_stack.remove()
    my_stack.remove()
    my_stack.remove()
    my_stack.remove()  # Try to remove from an empty stack
except IndexError as e:
    print(f"Error during stack removal: {e}")
except Exception as e:
    print(f"Unexpected error during stack removal: {e}")

print(f"Final stack: ", end="")
my_stack.print_stack()
print(f"Is stack empty? {my_stack.stack == []}")
print("-" * 20)

# Test Queue functionality
print("--- Testing Queue ---")
my_queue = Queue()
print(f"Initial queue: {my_queue.queue}")
print(f"Is queue empty? {my_queue.is_empty()}")
print(f"Queue size: {my_queue.size()}")

my_queue.enqueue(1)
my_queue.enqueue(2)
my_queue.enqueue(3)
print(f"Queue after enqueues: {my_queue.queue}")
print(f"Is queue empty? {my_queue.is_empty()}")
print(f"Queue size: {my_queue.size()}")
print(f"Top element (front): {my_queue.top()}")

dequeued_item = my_queue.dequeue()
print(f"Dequeued item: {dequeued_item}")
print(f"Queue after dequeue: {my_queue.queue}")
print(f"Queue size: {my_queue.size()}")
print(f"Top element (front) after dequeue: {my_queue.top()}")

try:
    my_queue.dequeue()
    my_queue.dequeue()
    my_queue.dequeue()
    my_queue.dequeue()  # Try to dequeue from an empty queue
except IndexError as e:
    print(f"Error during queue dequeue: {e}")

try:
    print(f"Top element of empty queue: {my_queue.top()}")
except IndexError as e:
    print(f"Error getting top of empty queue: {e}")

print(f"Final queue: {my_queue.queue}")
print(f"Is queue empty? {my_queue.is_empty()}")
print(f"Queue size: {my_queue.size()}")
print("-" * 20)
```
