```python
# Problem 1, 2, 3
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)
        print(f"Added to Queue: {item}")

    def dequeue(self):
        if not self.is_empty():
            item = self.queue.pop(0)
            print(f"Removed from Queue: {item}")
            return item
        else:
            print("The Queue is empty, nothing to remove.")

    def display(self):
        print("Current Queue State:", self.queue)

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

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

# Problem 4, 5, 6
class CircularQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = -1
        self.rear = -1

    def is_empty(self):
        return self.front == -1

    def is_full(self):
        return (self.rear + 1) % self.capacity == self.front

    def enqueue(self, element):
        if self.is_full():
            print("Cannot add to Circular Queue. It's full!")
            return
        if self.is_empty():
            self.front = 0
        self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = element
        print(f"Inserted into Circular Queue: {element}")

    def dequeue(self):
        if self.is_empty():
            print("Cannot remove from Circular Queue. It's empty!")
            return None
        element = self.queue[self.front]
        if self.front == self.rear:
            self.front = -1
            self.rear = -1
        else:
            self.front = (self.front + 1) % self.capacity
        print(f"Removed from Circular Queue: {element}")
        return element

    def display(self):
        if self.is_empty():
            print("Circular Queue is currently empty.")
        else:
            print("Elements in Circular Queue:", end=" ")
            if self.rear >= self.front:
                print(self.queue[self.front:self.rear + 1])
            else:
                print(self.queue[self.front:self.capacity] + self.queue[0:self.rear + 1])

# Problem 7, 8, 9
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item, priority):
        self.queue.append((priority, item))
        self.queue.sort(key=lambda x: x[0])
        print(f"Added to Priority Queue: {item} (Priority: {priority})")

    def dequeue(self):
        if not self.is_empty():
            priority, item = self.queue.pop(0)
            print(f"Removed from Priority Queue: {item} (Priority: {priority})")
            return item
        else:
            print("Priority Queue is empty. Nothing to remove.")

    def display(self):
        if self.is_empty():
            print("Priority Queue is currently empty.")
        else:
            print("Priority Queue Items:", [(priority, item) for priority, item in self.queue])

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



# 1. Basic Queue Operations
print("\n1. Basic Queue Operations:")
queue1 = Queue()
queue1.enqueue(10)
queue1.enqueue(20)
queue1.enqueue(30)
queue1.display()
queue1.dequeue()
queue1.display()

# 2. Queue Is Empty
print("\n2. Queue Is Empty:")
print("Is Queue Empty?", queue1.is_empty())
queue1.dequeue()
queue1.dequeue()
print("Is Queue Empty?", queue1.is_empty())

# 3. Queue Size
print("\n3. Queue Size:")
print("Queue Size:", queue1.size())
queue1.enqueue(40)
print("Queue Size:", queue1.size())

# 4. Circular Queue Basics
print("\n4. Circular Queue Basics:")
queue2 = CircularQueue(3)
queue2.enqueue(10)
queue2.enqueue(20)
queue2.enqueue(30)
queue2.display()
```

```
queue2.dequeue()
queue2.display()

# 5. Circular Queue Is Full
print("\n5. Circular Queue Is Full:")
print("Is Circular Queue Full?", queue2.is_full())
queue2.enqueue(40)
print("Is Circular Queue Full?", queue2.is_full())

# 6. Circular Queue Wraparound
print("\n6. Circular Queue Wraparound:")
queue2.dequeue()
queue2.enqueue(50)
queue2.display()

# 7. Priority Queue Basics
print("\n7. Priority Queue Basics:")
queue3 = PriorityQueue()
queue3.enqueue("Task1", 2)
queue3.enqueue("Task2", 1)
queue3.enqueue("Task3", 3)
queue3.display()
queue3.dequeue()
queue3.display()

# 8. Priority Queue Sorting
print("\n8. Priority Queue Sorting:")
queue3.enqueue("TaskA", 5)
queue3.enqueue("TaskB", 2)
queue3.enqueue("TaskC", 3)
queue3.display()

# 9. Priority Queue Edge Cases
print("\n9. Priority Queue Edge Cases:")
queue3.enqueue("TaskD", 3)
queue3.enqueue("TaskE", 1)
queue3.display()
queue3.dequeue()
queue3.display()




# Problem 10
```

```
print("\nChoose the Queue Type:")
print("1. Standard Queue")
print("2. Circular Queue")
print("3. Priority Queue")

choice = int(input("Enter your selection (1/2/3): "))
if choice == 1:
  queue = Queue()
  print("\nStandard Queue selected.")
elif choice == 2:
  capacity = int(input("Enter the size of the Circular Queue: "))
  queue = CircularQueue(capacity)
  print("\nCircular Queue selected.")
elif choice == 3:
  queue = PriorityQueue()
  print("\nPriority Queue selected.")
else:
  print("Invalid choice. Exiting program.")

while True:
  print("\nAvailable Operations:")
  print("1. Add (Enqueue)")
  print("2. Remove (Dequeue)")
  print("3. View Queue (Display)")
  print("4. Exit Program")
  option = int(input("Enter your choice: "))
  if option == 1:
    if isinstance(queue, PriorityQueue):
      item = input("Enter the item: ")
      priority = int(input("Assign a priority (integer): "))
      queue.enqueue(item, priority)
    else:
      item = input("Enter the item: ")
      queue.enqueue(item)
  elif option == 2:
    queue.dequeue()
  elif option == 3:
    queue.display()
  elif option == 4:
    print("Goodbye!")
    break
  else:
    print("Invalid option, please try again.")
```