Qno.1)

from collections import deque

def genBin(n):

# Create a queue to store the binary number sequences

queue = deque()

# Add the first binary number to the queue

queue.append('110')

# Loop until the queue is empty

while queue:

# Get the first binary number in the queue

s = queue.popleft()

# Convert the binary number to an integer and check if it's within the range 1 to n

num = int(s, 2)

if num <= n:

print(s, end=' ')

# Add the next two binary number sequences to the queue

queue.append(s + '0')

queue.append(s[:-1] + '1')

print()

# Call the function with an input of 7

genBin(7)

Qno.2)

from collections import deque

class Stack:

def \_\_init\_\_(self):

self.queue = deque()

def push(self, val):

# Get the current size of the queue

n = len(self.queue)

# Add the new value to the end of the queue

self.queue.append(val)

# Move the first n elements of the queue to the end

for \_ in range(n):

self.queue.append(self.queue.popleft())

def pop(self):

# Remove and return the last element of the queue

return self.queue.popleft()

def is\_empty(self):

# Check if the queue is empty

return not bool(self.queue)

# Create a new stack object

s = Stack()

# Push values onto the stack

s.push(1)

s.push(2)

s.push(3)

# Pop values off the stack

print(s.pop()) # Output: 3

print(s.pop()) # Output: 2

# Check if the stack is empty

print(s.is\_empty()) # Output: False

# Push another value onto the stack

s.push(4)

# Pop the remaining values off the stack

print(s.pop()) # Output: 4

print(s.pop()) # Output: 1

# Check if the stack is empty

print(s.is\_empty()) # Output: True

Qno.3)

class PriorityQueue:

def \_\_init\_\_(self):

self.queue = []

def push(self, val, priority):

# Add the value and priority as a tuple to the queue

self.queue.append((val, priority))

# Sort the queue in ascending order of priority

self.queue.sort(key=lambda x: x[1])

def pop(self):

if not self.is\_empty():

# Remove and return the first element of the queue

return self.queue.pop(0)[0]

def is\_empty(self):

# Check if the queue is empty

return not bool(self.queue)

# Create a new priority queue object

pq = PriorityQueue()

# Push values onto the queue with their priorities

pq.push('c', ord('c'))

pq.push('a', ord('a'))

pq.push('b', ord('b'))

# Pop values off the queue

print(pq.pop()) # Output: 'a'

print(pq.pop()) # Output: 'b'

# Push another value onto the queue

pq.push('d', ord('b'))

# Pop the remaining values off the queue

print(pq.pop()) # Output: 'd'

print(pq.pop()) # Output: 'c'

# Check if the queue is empty

print(pq.is\_empty()) # Output: True

Qno.4)

def getNumElem(size, front, rear):

if front == -1:

# If the front index is -1, the queue is empty

return 0

elif rear >= front:

# If the rear index is greater than or equal to the front index, the queue is not wrapped around

return rear - front + 1

else:

# If the rear index is less than the front index, the queue is wrapped around

return size - front + rear + 1

# Define the maximum size and indices of a circular queue

size = 10

front = 2

rear = 7

# Calculate the number of elements in the circular queue

num\_elem = getNumElem(size, front, rear)

# Print the result

print(num\_elem) # Output: 6