Faculty of
Information Technology
King Mongkut's Institute of Technology Ladkrabang

Name:.....

Student ID:.....

Assignment # 3: Queues

1. In your own words, describe the definition of a queue and its properties.

2. Given a class of Queue which is defined as follows:

```
class ArrayQueue:
  DEFAULT_CAPACITY = 10   # moderate capacity for all new queues

  def __init__(self):
    """ Create an empty queue with specified size. """
    self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
    self._size = 0
    self._front = 0
    self._rear = 0

  def __len__(self):
    """ Return the number of elements in the queue. """
    return self._size

  def is_empty(self):
    """ Return True if the queue is empty. """
    return self._size == 0

  def first(self):
  """ Return (but do not remove) the element at the front of the queue.
    Raise Empty exception if the queue is empty."""
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[self._front]

  def dequeue(self):
    """ Remove and return the first element of the queue.
    Raise Empty exception if the queue is empty."""
    if self.is_empty():
      raise Empty('Queue is empty')
    answer = self._data[self._front]
    self._data[self._front] = None          # Reclaiming unused space
    self._front = (self._front + 1) % len(self._data) # shift the
      location of the front index rightward
    self._size -= 1
    return answer
```

```
def enqueue(self, e):
  """ Add an element to the back of queue."""
  if self._size == len(self._data):
    self._resize(2*len(self._data))     # double the array size when
    all slots are occupied.

  if self._rear == 0 and self._data[0] == None: # For the first case
    of rear = 0 and no data in Q[0] - Empty Queue
    self._data[self._rear] = e
  else:
    self._rear = (self._rear + 1) % len(self._data) # shift the
    location of the rear index rightward
    self._data[self._rear] = e
  self._size += 1    # Keep track of number of elements in ArrayQueue


def _resize(self, cap):                 # Assume cap >= len(self)
  """ Resize to a new list of capacity >= len(self). """
  old = self._data                      # keep track of existing list
  self._data = [None] * cap             # allocate list with new
    capacity
  walk = self._front
  for k in range(self._size):           # consider existing elements
    self._data[k] = old[walk]           # Shift indices to start at 0
    walk = (1 + walk) % len(old)        # use old size as modulus
  self._front = 0                       # front has been realigned
  self._rear = self._size - 1           # rear has been realigned
```

A method to create an empty queue is:

Q = ArrayQueue()

What values are returned during the following series of queue operations, if executed upon an initially empty queue?

Operation	Return Value	Queue values
Q.enqueue(7)		
Q.enqueue(2)		
Q.dequeue()		
Q.enqueue(5)		
Q.enqueue(9)		
Q.first()		
Q.dequeue()		
Q.dequeue()		
Q.is_empty()		
Q.enqueue(9)		
len(Q)		
Q.enqueue(0)		
Q.dequeue()		
Q.dequeue()		
Q.dequeue()		
Q.dequeue()		
len(Q)		

3. Based on a Queue class as defined in Question 2, write the pseudocodes or python codes for the following tasks.
    - Create 3 queues: A, B and C.
    - Add new items including “CPU”, “RAM”, “GPU”, “Mainboard”, “PSU”, and “SSD”, respectively, into queue “A”.
    - Add new items including “16 GB”, “2 TB”, “850 Watt”, “3.2 GHz”, “ATX”, and “1350 MHz”, respectively, into queue “B”.

- Transfer the items from queue “A” and “B” to queue “C” such that the result of queue C is [‘CPU’, ‘3.2 GHz’, ‘Mainboard’, ATX’]’. Also, show the list elements of A and B after operations.

- Rearrange items into the following results:
  A = [‘RAM’, ‘16 GB’, ‘SSD’, ‘2 TB’],
  B = [‘GPU’, ‘1350 Mhz’, ‘PSU’, ‘850 Watt’], and
  C = [“CPU’, ‘3.2 GHz’, ‘Mainboard’, ATX”]

- Remove all items in Queue “C” by using a loop.

4. Suppose you have three nonempty queue X=[1, 2, 3], Y=[4, 5], and Z=[6, 7, 8, 9], describe a sequence of operations using Python or pseudocode that results in X=[9, 8, 7, 6] and Y=[1, 2, 3, 4, 5] with both sets of those elements in their original order.

5. Suppose an initially empty queue Q has executed a total of 32 enqueue operations, 10 first operations, and 15 dequeue operations, 5 of which raised Empty errors that were caught and ignored. What is the current size of Q? Also, show your calculation of how you obtain the size.

6. Based on a Queue class as defined in Question 2, implement **double-ended queues(deque)** class and the following functions using Python code or pseudocode.

```
class ArrayDeque:
    def __init__(self):
        """ Create an empty queue with specified size. """
        #Hint: Attributes _data, _size, _front, _rear of self parameter should be initialized similar to ArrayQueue class.
        #Add your code here
```

```
    def __len__(self):
        """ Return the number of elements in the queue. """
        #Add your code here
```

```
def is_empty(self):
        """ Return True if the queue is empty. """
        #Add your code here
```

```
def first(self):
        """ Return (but do not remove) the element at the front of the queue.
    Raise Empty exception if the queue is empty."""
        #Add your code here
```

```
def last(self):
        """ Return (but do not remove) the element at the back of the queue.
     Raise Empty exception if the queue is empty."""
        #Add your code here
```

```
def add_first(self, e):
        """ Add an element to the front of queue."""
        #Hint: 1) When a queue is full, a code should either increase queue size
or reject new elements.
                2) Front index should be used when adding new elements and
        updated by shifting the index leftward.
                3) Don't forget to update the _size attribute to keep track of
        number of elements in the queue
        #Add your code here
```

```
def remove_first(self):
        """ Remove and return the first element of the queue.
    Raise Empty exception if the queue is empty."""
        #Add your code here
```

```
def add_last(self, e):
    """ Add an element to the back of queue."""
    #Add your code here
```

```
def remove_last(self):
    """ Remove and return the last element of the queue.
  Raise Empty exception if the queue is empty."""
    #Hint: 1) Rear index should be used when removing elements and updated by shifting the index leftward.
          2) Don't forget to update the _size attribute to keep track of number of elements in the queue
    #Add your code here
```

```
def _resize(self, cap):
    """ Resize current queue size to a new list of capacity >= len(self). """
    #Add your code here
```

7. Based on a deque class on Question 7, what are the queue values (Deq._data) for the following operations?

Deq = ArrayDeque()

7.1) for i in range(5):
        Deq.add_last(i)

7.2) for i in range(11, 16):
        Deq.add_first(i)

7.3) Deq.add_first(20)

7.4) Deq.remove_first()

7.5) Deq.remove_last()

8. Maximum in sliding window problem

Given a sequence $a_0, \ldots, a_{n-1}$ of integers and a sliding window of size k ≤ n - 1, write a Python or pseudocode function that find the maximum among $\{a_i, \ldots, a_{i+k-1}\}$ for every 0 ≤ i ≤ n − k (in other words, for each subarray of size k).

Specifically, k starts from index 0 which means the first sliding window. In each iteration, the sliding window moves to the right by one position till n-k. Write a code to return an array providing the maximum number for each sliding window.

Sliding window problem is useful to compute a running or moving average, that is, calculating a series of averages from different subsets. This is useful for investigating a trend such as stock prices movement. Sliding window is also used to create a set of all adjacent pairs.

Illustration of sliding window of size 3 and array length of 5 (n=5)

8	4	3	5	9

|----------1st window ----------|
|----------2nd window ----------|
|----------3rd window ----------|

Expected output:

The first element of the array is max(data[0,...,k-1]), then the second element is max(data[1,...,k]) and so on.

The size of the output array will be n-k+1.

Example:

Input = [2, 7, 3, 1, 5, 2, 6 ,2] , k = 4

Description: The window size is 4 and the maximums for each window are as follows:

1st window -> max(2, 7, 3, 1) = 7

2nd window -> max(7, 3, 1, 5) = 7

3rd window -> max(3, 1, 5, 2) = 5

4th window -> max(1, 5, 2, 6) = 5

5th window -> max(5, 2, 6, 2) = 6

Output = [7, 7, 5, 5, 6]

8.1 Write a naive function that solves this problem by scanning each window separately which takes *O*(nk) time.

- The code should have two parameters, one is used as a list that collects all max numbers for all sliding windows, another, which is used in an inner loop, is a max number that compares with other numbers in a sliding window.
- The outer loop should run n-k+1 times representing each iteration of the sliding window.
    - It should add a max number for each iteration.
- The inner loop should run k times to iterate all numeric elements in a sliding window.
    - It should compare each number in a sliding window and select the maximum number.

```
def max_sliding_window(data, n, k):
    #Add your code here
    for #Add your code here
        #Add your code here
        for #Add your code here:
            if #Add your code here:
                #Add your code here
        #Add your code here
    return #Add your code here
```

8.2 Write another function that solves this problem in *O*(n) time.

Hint for 8.2: Store relevant items in a **deque** as defined in Question 7. Use a **deque** to store elements of the current window. At the same time, store only relevant elements: before adding a new element drop all smaller elements. Window is moved forward by dropping the first element and adding the next element to the back of the **deque**.

Solution steps

1. Create a **deque** to store elements.
2. 1st window: Add the first K elements into the back of **deque**. During inserting, remove all unnecessary indices of elements from the back of the queue that are smaller than the current array/data element. To remove these indices, we will remove all the elements from the back of the **deque** that is smaller than the current array element.
3. After the iteration for the first K elements, the maximum element's index is at the front of the **deque**. This element should be added into an answer list.
4. For after windows: Iterate through the remaining part of the array (from k to n) and remove the elements from the front if they are out of the current window.
5. Again, insert the element in the back of the **deque** and before inserting delete those unnecessary indices which are smaller than the current array element.
6. After each iteration, the result is the maximum elements of the current window. This element should be added into an answer list.

```
def max_sliding_window_deq(data, n, k):
  #Initialise 2 parameters: ArrayDeque D and empty answer list: ans
  #Add your code here

  #For first k elements (1st window)
  for i in range(k):
        #Remove the indices of elements from the back of deque that are smaller than the current elements
        while D.is_empty() != True and data[i] > data[D.last()]:
              #Add your code here
        #Add index of current element into the back of deque
        #Add your code here
  #Add maximum number of a current window into an answer list
  #Add your code here


  #For later windows from k to n
  for i in range(k, n):
        #Drop the front elements of deque that are out of window (k sized window)
        while D.is_empty() != True and D.first() <= i-k:
              #Add your code here
        #Remove the indices of elements from the back of deque that are smaller than the current elements
        while D.is_empty() != True and data[i] > data[D.last()]:
              #Add your code here
        #Add index of current element into the back of deque
        #Add your code here
        #Add maximum number of a current window into an answer list
        #Add your code here
  return ans
```