A Deque (pronounced “Deck”) is a linear data structure which behaves like a double-ended queue, i.e., it allows adding or removing items from either the front or the rear of the Deque.

1. One possible implementation of a Deque would be to use a Python list to store the Deque items such that
   • the rear item is always stored at index 0,
   • the front item is always stored at the highest index (or -1)

   Deque Object
   items: [ ]
   (rear) (front)

   list Object
   [ 'a', 'b', 'c', 'd' ]
   0 1 2 3

   class Deque(object):
       def __init__(self):

   a) Complete the __init__ method and determine the big-oh, \( O() \), for each Deque operation, assuming the above implementation. Let \( n \) be the number of items in the Deque.

<table>
<thead>
<tr>
<th>isEmpty</th>
<th>addFront</th>
<th>removeFront</th>
<th>addRear</th>
<th>removeRear</th>
<th>size</th>
</tr>
</thead>
</table>

   b) Write the methods for the addRear and removeRear operation.

   def addRear(self, newItem):  
   def removeRear(self):

2. An alternative implementation of a Deque would be a linked implementation as in:

   LinkedDeque Object
   _rear: [ ]
   _front: [ ]
   _size: 4

   class LinkedDeque(object):
       def __init__(self):

   a) Complete the __init__ method and determine the big-oh, \( O() \), for each Deque operation assuming the above linked implementation. Let \( n \) be the number of items in the Deque.

<table>
<thead>
<tr>
<th>isEmpty</th>
<th>addFront</th>
<th>removeFront</th>
<th>addRear</th>
<th>removeRear</th>
<th>size</th>
</tr>
</thead>
</table>

   b) Suggest an improvement to the above linked implementation of the Deque to speed up some of its operations.
from node import Node

class Node2Way(Node):
    def __init__(self, initdata):
        Node.__init__(self, initdata)
        self.previous = None

    def getPrevious(self):
        return self.previous

    def setPrevious(self, newprevious):
        self.previous = newprevious

3. An alternative implementation of a Deque would be a doubly-linked implementation as in:

DoublyLinkedDeque Object

a) Determine the big-oh, $O()$, for each Deque operation assuming the above doubly-linked implementation. Let $n$ be the number of items in the Deque.

<table>
<thead>
<tr>
<th>Method</th>
<th>Big-Oh</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEmpty</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>addFront</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>removeFront</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addRear</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeRear</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>size</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

4. A priority queue has the same operations as a regular queue, except the items are NOT returned in the FIFO (first-in, first-out) order. Instead, each item has a priority that determines the order they are removed. A hospital emergency room operates like a priority queue -- the person with the most serious injury has highest priority even if they just arrived.

a) Suppose that we have a priority queue with integer priorities such that the smallest integer corresponds to the highest priority. For the following priority queue, which item would be dequeued next?

priority queue:

b) To implement a priority queue, we could use an unordered Python list. If we did, what would be the big-oh notation for each of the following methods: (justify your answer)

- enqueue:
- dequeue:

c) To implement a priority queue, we could use a Python list order by priorities in descending order. If we did, what would be the big-oh notation for each of the following methods: (justify your answer)

- enqueue:
- dequeue