CS304 Practice midterm

Midterm date: Tuesday February 22nd from 3:00 PM to 4:30 PM

QUESTIONS:

1: match the time complexity to the following python list operations

operation	Time complexity
L.append(elem)	O(n)
L.find_min()	O(1)
L.find(elem)	O(1)*
L[i]	O(n)
L.delete(elem)	O(1)*
L.pop()	O(n)

*amortized running time

2: evaluate the following postfix expressions

456*+

78+32+/

3: convert the following postfix expressions to infix

AB+C+D+

AB+C*

4: convert the following infix expressions to postfix

A+B*C+D

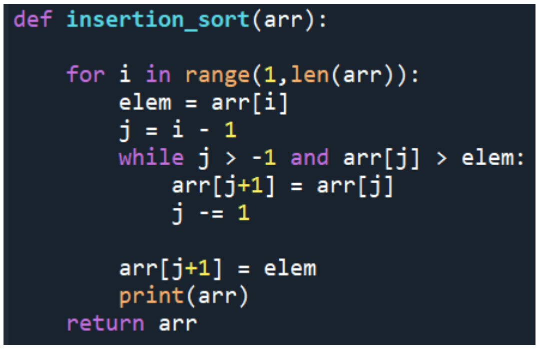
(A+B)*(C+D)

5: write an algorithm in python for merging two sorted arrays (examples below)

A1 = [1,2,3,4], A2 = [0,3,4,6]

A1 = [], A2 = [1]

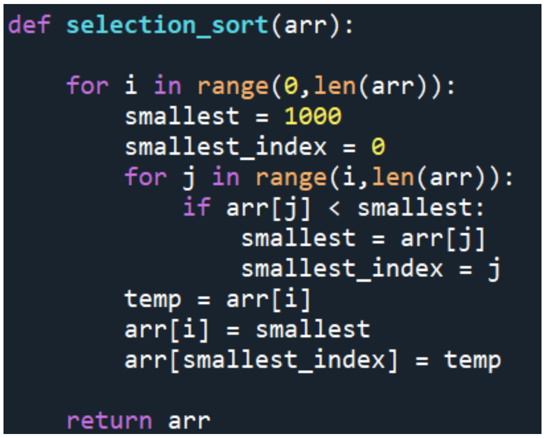
6: show the state of the array after each pass of the outer loop of insertion sort (algorithm below): arr = [5,4,3,2]



```python
def insertion_sort(arr):

    for i in range(1,len(arr)):
        elem = arr[i]
        j = i - 1
        while j > -1 and arr[j] > elem:
            arr[j+1] = arr[j]
            j -= 1

        arr[j+1] = elem
        print(arr)
    return arr
```

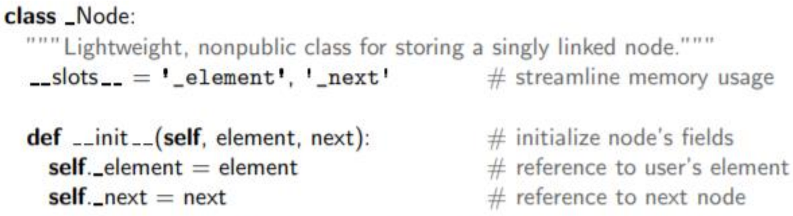
7: show the state of the array after each pass of the outer loop of selection sort (algorithm below).

arr = [5,4,3,2]



```
def selection_sort(arr):

    for i in range(0,len(arr)):
        smallest = 1000
        smallest_index = 0
        for j in range(i,len(arr)):
            if arr[j] < smallest:
                smallest = arr[j]
                smallest_index = j
        temp = arr[i]
        arr[i] = smallest
        arr[smallest_index] = temp

    return arr
```

8: given the following python definition of a Node, implement two functions 1) remove_first and 2) add_last for a singly linked list. Your function should initialize a new node and then perform the operations required to remove it from front of list (1) or add it to end (2).

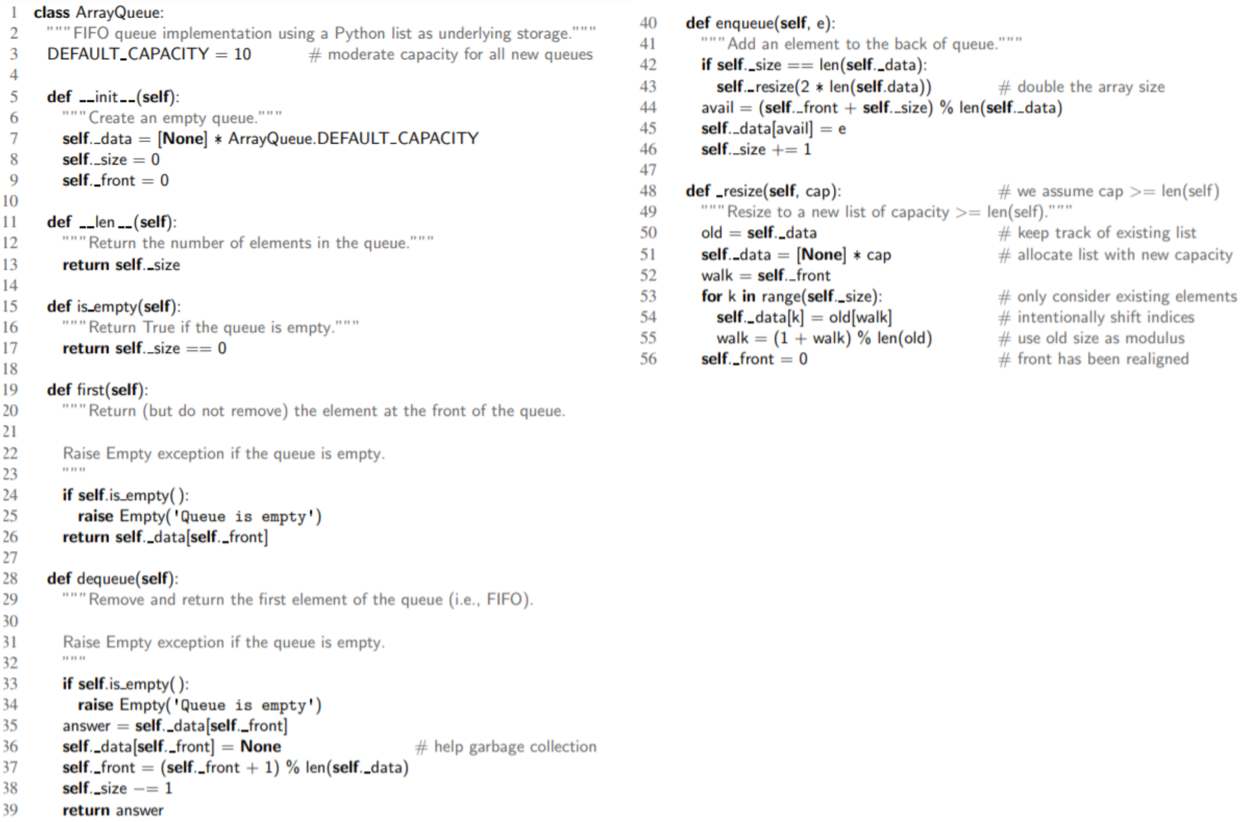


```
class _Node:
  """Lightweight, nonpublic class for storing a singly linked node."""
  __slots__ = '_element', '_next'          # streamline memory usage

  def __init__(self, element, next):       # initialize node's fields
    self._element = element                # reference to user's element
    self._next = next                      # reference to next node
```

Code Fragment 7.4: A lightweight _Node class for a singly linked list.

9: code for a circular array queue implementation is below: show the state of the list `_data` after each of the following operations. Use the Ø symbol to indicate empty locations in the list (example: after operation a, we have _data = [Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø]

```
a. q = ArrayQueue()
b. q.enqueue(a)
c. q.enqueue(1)
d. q.dequeue()
e. q.enqueue('asdf')
```



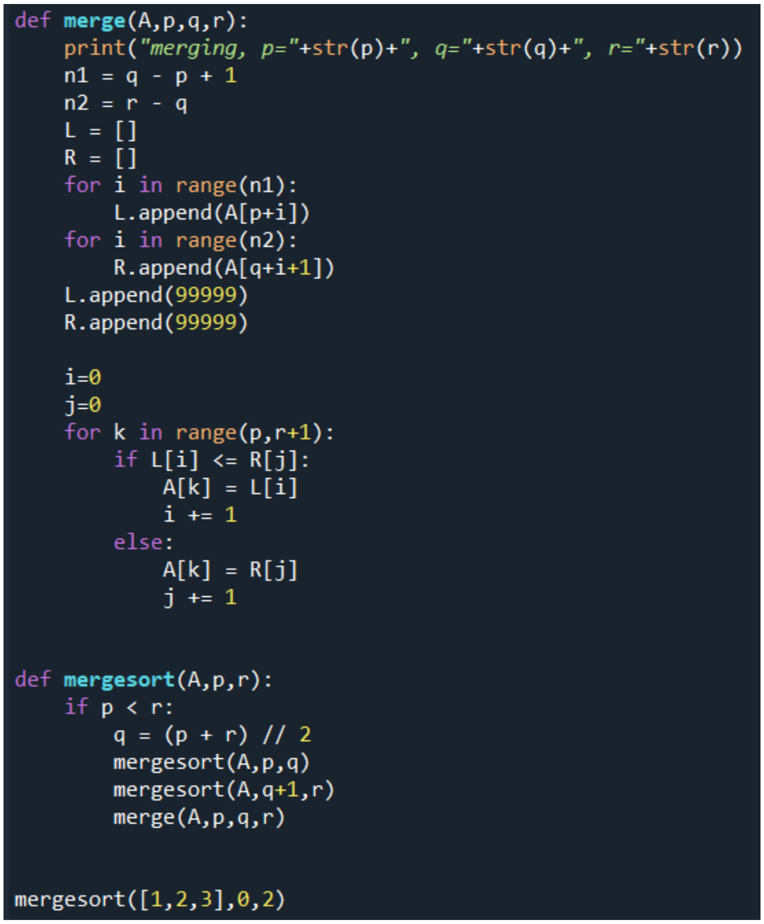
```
class ArrayQueue:
  """FIFO queue implementation using a Python list as underlying storage."""
  DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

  def __init__(self):
    """Create an empty queue."""
    self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
    self._size = 0
    self._front = 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 (i.e., FIFO).

    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              # help garbage collection
    self._front = (self._front + 1) % len(self._data)
    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
    avail = (self._front + self._size) % len(self._data)
    self._data[avail] = e
    self._size += 1

  def _resize(self, cap):                       # we 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):                 # only consider existing elements
      self._data[k] = old[walk]                 # intentionally shift indices
      walk = (1 + walk) % len(old)              # use old size as modulus
    self._front = 0                             # front has been realigned
```

10: the Fibonacci sequence is defined as 0, 1, 1, 2, 3, 5, … etc. each number in the sequence is the sum of the two preceding numbers (except for 0 and 1 which are the base case).

a) Write a recursive python function for computing the Fibonacci sum.

b) write a non-recursive python function for computing the Fibonacci sum.

11: what output does the following code print?



```
def merge(A,p,q,r):
    print("merging, p="+str(p)+", q="+str(q)+", r="+str(r))
    n1 = q - p + 1
    n2 = r - q
    L = []
    R = []
    for i in range(n1):
        L.append(A[p+i])
    for i in range(n2):
        R.append(A[q+i+1])
    L.append(99999)
    R.append(99999)

    i=0
    j=0
    for k in range(p,r+1):
        if L[i] <= R[j]:
            A[k] = L[i]
            i += 1
        else:
            A[k] = R[j]
            j += 1


def mergesort(A,p,r):
    if p < r:
        q = (p + r) // 2
        mergesort(A,p,q)
        mergesort(A,q+1,r)
        merge(A,p,q,r)


mergesort([1,2,3],0,2)
```

Hints:

Check here for answers to questions 2,3,4

For the sorting questions (6,7,11) you can copy-paste the code into spyder and run with some print statements to verify your results.

Note – the midterm will be *closed book.* No material besides pen, pencil, and eraser is allowed. Exam booklets will be supplied.