1) For the code below give the output of the statements listed:

```
#class to represent imaginary numbers

class Imaginary():
   __slots__ = ("real", "imaginary")

#class to represent a vector in 2d space
class VectorIn2DSpace():
   __slots__ = ("magnitude", "direction")

anImaginaryNumber = Imaginary()
aVector = VectorIn2DSpace()
x = 5
y = 0.5
z = 1
what = (isinstance(x, int) == isinstance(z, int))
```

What is the output of the following statements:

a) print(isinstance(x, Imaginary))

b) print(isinstance(anImaginaryNumber, Imaginary))

c) print(isinstance(y, int))

d) print(isinstance(y, float))

e) print(isinstance(z, str))

f) print(isinstance(z, float))

g) print(isinstance(aVector, bool))

h) print(isinstance(aVector, VectorIn2DSpace))

i) print(isinstance(what, bool))

2. Assume you are using the linked list code that was developed in lecture (i.e. a List class and a Node class). Show the output of the following code:

```
lst = mkMyList()
append(lst, "ahoy")
append(lst, "booty")
append(lst, "landlubber")
append(lst, "swashbuckler")
append(lst, "grog")
append(lst, "dubloon")
pop(lst, 3)
print("size:", lst.size)
print("get 1:", get(lst, 1))
print("get 4:", get(lst, 4))
print("index grog:", index(lst, "grog"))
```

3. Identify the Big-O time complexity of each list operation. Assume an "array based list" is Python's built in list and a "linked list" is the lecture problem's node based implementation.

a) Inserting an element into the front of a linked list.
b) Inserting an element into the front of an array list, which is currently full.
c) Accessing the last element in a linked list.
d) Accessing the last element in an array list.
e) Printing the elements in a linked list in reverse order.
f) Printing the elements in an array list in reverse order.

4. Write a list function, count, which takes a MyList and an element to search for. It should return the number of occurrences of the element in the list. Implement the function both iteratively and recursively.

5. Write a list function, set, which takes a list, an index and a new element. The function should change the node at the index to contain the new element. Implement the function both iteratively and recursively.