Question 1. [6 marks]

Define each of the following in a short sentence or phrase for 1 mark each.

Abstract Data Type:

```
A type of object existing as an idea.
```

Inheritance:

```
Property of Object Oriented programming languages that allow
building hierarchies of types, or extending types.
```

Test Driven Development:

```
The method of programming where the tests are written prior to the main
program.
```

Binary Tree:

```
A tree whose nodes have a maximum of two children.
```

Leaf Node:

```
A tree node with no children.
```

Internal Node:

```
A tree node with at least one child.
```

Question 2. [12 MARKS]

A fast food restaurant owner has asked you to design a system for them that lets them keep track of their orders in a computer. The following is the description they have given you:

When a customer places an order, they specify which item(s) from the menu they want and how many of each they would like. They also specify whether they want to upgrade any of the items to a combo, and whether they want to dine in or take out. The cashier should be able to input this information and create a new order. They should also be able to compute an order's subtotal, and compute taxes by specifying the tax rate as a percentage. There are two menus: a main menu and a side menu and so there are two kinds of menu items. Only main menu items are upgradable to combo. Each menu item has a name, and a price. Side menu items come in three different sizes: small, medium and large.

Your task is to perform an object oriented analysis for this project, i.e. give a list of classes that are needed along with their data attributes and instance methods. If applicable, specify whether a class is an extension of another class.

```
Order
-------
Attributes:

* dictionary of menu item objects with quantities as values (two lists of the
same size, one containing menu items and the other containing quantities is
also acceptable although it's not good design)
* boolean dine_in or take_out

Methods:

* compute_subtotal()
* compute_tax(rate)

MenuItem
------------
Attributes:

* name
* price

MainMenuItem (extends MenuItem)
------------------------------------------
Methods:

* upgrade_to_combo()

SideMenuItem (extends MenuItem)
------------------------------------------
Other Attributes:

* string size (s/m/l)
```

Question 3. [12 marks]

Given the function `list_to_lower` below, write three different nose tests that test the functionality of `list_to_lower` under three distinct circumstances. Each assert statement should provide a small description of the test case which it is testing. For example, if a function with an integer argument was to be tested, a sample test description could read "Testing Odd Numbers". You are NOT asked to complete the body of the function.

```
def list_to_lower(str_list):
    '''Return a new list of the elements in str_list after they have been
    turned to lowercase strings. str_list should be a list of strings,
    otherwise a BadTypeException is raised with the error message:
    'Non-string object found.'
    '''
```

```
Any three of the following:

def test_list_to_lower_empty_list():
    assert q2.list_to_lower([]) == [], 'Case: Empty List'

def test_list_to_lower_bad_types():
    try:
        q2.list_to_lower(['OH!', 3])
    except q2.BadTypeException, error:
        assert error.message == 'Non-string object found.'
    else:
        assert False, 'Case: List with non-string elements'

def test_list_to_lower_side_effects():
    my_list = ['HO HO HO', 'Wassup?']
    q2.list_to_lower(my_list)
    assert my_list == ['HO HO HO', 'Wassup?'], \
           'Case: Original list shoud not change'

def test_list_to_lower_general():
    my_list = ['Ho Ho Ho', 'Wassup?']
    assert q2.list_to_lower(my_list) == ['ho ho ho', 'wassup?'], \
           'Case: General list of strings that have capital letters'

def test_list_to_lower_all_lower():
    my_list = ['already lower case', 'all lower case here']
    assert q2.list_to_lower(my_list) == \
           ['already lower case', 'all lower case here'], \
           'Case: General list of strings with no capital letters'
```

Question 4. [15 marks]

Write merge sort by completing the functions below according to their docstring.

Part (a) [5 marks]

```
def merge_sort(list):
    '''Sort list by taking advantage of the merge helper function.'''





    if len(list_) <= 1:
        return list_
    else:
        middle = len(list_)/2
        left = merge_sort(list_[0:middle])
        right = merge_sort(list_[middle:])
        return merge(left, right)
```

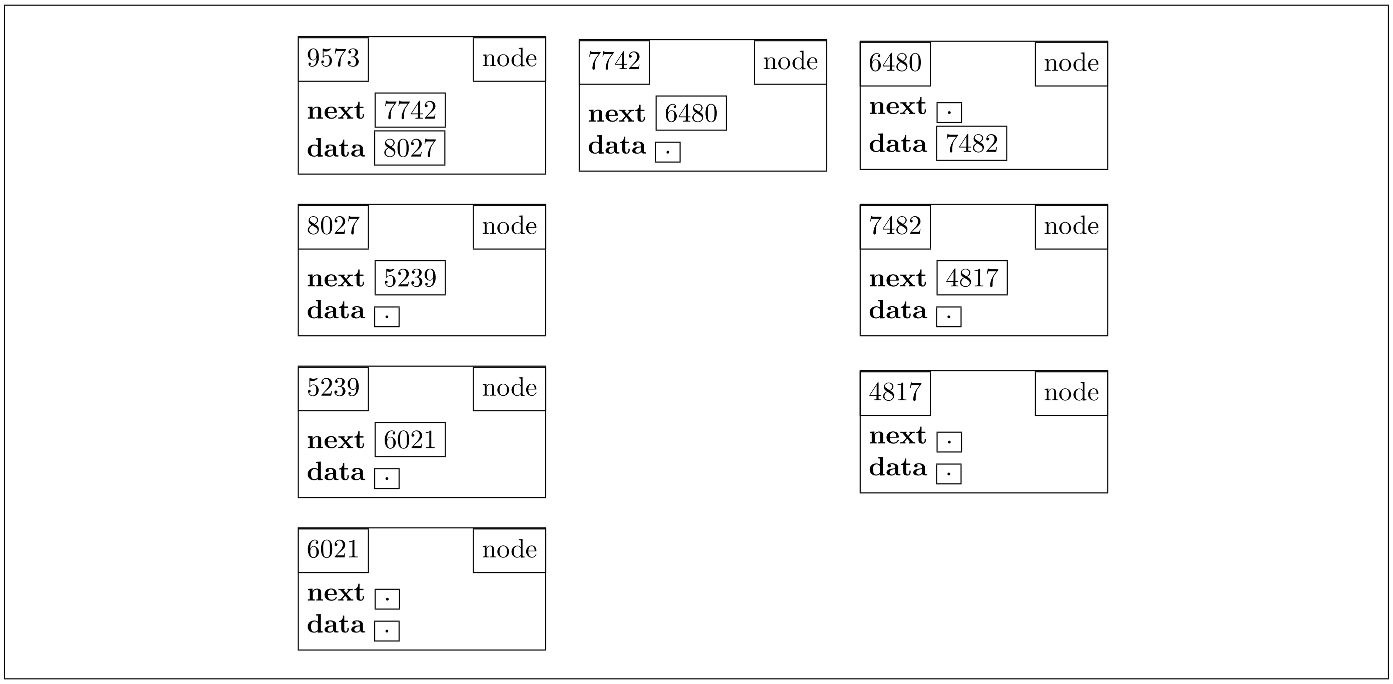
Part (b) [10 marks]

```
def merge(left, right):
    '''Merge sorted lists left and right by returning a new sorted
    list that holds all the elements of left and right.'''






    left_index = 0
    right_index = 0
    result = []
    while left_index < len(left) and right_index < len(right):
        if left[left_index] < right[right_index]:
            result.append(left[left_index])
            left_index += 1
        else:
            result.append(right[right_index])
            right_index +=1

    while left_index < len(left):
        result.append(left[left_index])
        left_index += 1

    while right_index < len(right):
        result.append(right[right_index])
        right_index += 1

    return result
```

Question 5. [16 MARKS]

Part (a) [1 MARK]

Describe as best as you can what you see in the picture below. Make sure you understand what each variable is pointing to. Draw arrows to help you keep track of this information if you like. A square with a '.' inside signifies that the variable points to None.



Part (b) [15 MARKS]

On the next page, write a recursive function that takes in a pointer to the first node object on the first row and modifies the whole picture so that the very last node in each column points back to the first node in that column. In the cases where there are no nodes dangling off of the node in the first row, you need not do anything. There can be any number of nodes in each row or column. The picture given is to be used just as an example. Your function should work by mutating its argument, no return value required.

You are allowed to use one helper function to get the last node in each column. Maximum mark available for solutions that use a `for` or `while` loop is 7 marks.

Continued from the previous page.

```
def loopy(head):
```

```
def get_last(head):
    if head and head.next:
        return get_last(head.next)
    else:
        return head

def loopy(head):

    if head:
        if head.data:
            last_node = get_last(head.data)
            last_node.next = head
        if head.next:
            loopy(head.next)
```

CONT'D...

Question 6. [12 MARKS]

Implement a Queue ADT using two Stacks by completing the body of the methods below. This means that the elements inside your queue should be stored in the stacks instead of normal Python lists or linked lists (as you did in E2). This question is continued on the next page. You can assume you have a `Stack` class (where the constructor creates an empty Stack) with `push`, `pop`, and `is_empty` methods, and you don't know how it's implemented.

Hint: As scrap work, draw yourself two stacks and try pushing and popping elements to see how you can generate a Queue-like behaviour.

Part (a) [2 MARKS]

```
from stack import Stack
class Queue(object):
    def __init__(self):
        '''Initialize an empty queue.'''

        self.stack_1 = Stack()
        self.stack_2 = Stack()
```

Part (b) [1 MARK]

```
    def enqueue(self, item):
        '''Add the given item to the end of the queue. '''

        self.stack_1.push(item)
```

CONT'D...

Continued from the previous page.

Part (c) [9 MARKS]

```
    def dequeue(self):
        '''Remove and return the next item at the beginning/head of the queue.'''
```

```
        # take everything out of stack 1 and put in stack 2 to reverse order.
        while not self.stack_1.is_empty():
            self.stack_2.push(self.stack_1.pop())

        # if there's anything in stack_2, get the top in order to return later
        if not self.stack_2.is_empty():
            result = self.stack_2.pop()
        else:
            result = None

        # empty the elements in stack_2 back in stack_1
        while not self.stack_2.is_empty():
            self.stack_1.push(self.stack_2.pop())

        return result
```

Question 7. [8 MARKS]

These questions have you draw trees. You don't need to draw the full picture of memory; instead, use circles with values inside them.

Part (a) [4 MARKS]

This question is about binary search trees. Draw the tree resulting from the following operations, where when we remove a value we replace it with the smallest value in the right subtree.

1. Insert 7
2. Insert 10
3. Insert 12
4. Insert 4
5. Remove 10
6. Insert 9
7. Insert 3
8. Insert 11
9. Remove 7

```
    9
  4    12
3     11
```

Part (b) [4 MARKS]

This question is about min-heaps. Draw the heap resulting from the following operations.

1. Insert 7
2. Insert 10
3. Insert 12
4. Insert 4
5. Remove the value at the root
6. Insert 9
7. Insert 3
8. Insert 11
9. Remove the value at the root

```
    7
 9     11
10   12
```

Question 8. [12 marks]

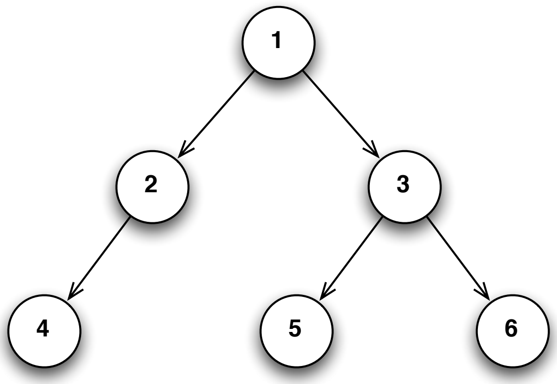
Consider the following implementation of a binary tree node and a function that makes use of it.

```
class Node(object):
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

```
def func(node):
    if node:
        node.left = func(node.left)
        node.right = func(node.right)

        cur_node = node
        while cur_node.right:
            cur_node = cur_node.right

        cur_node.right = node.left
        node.left = None
    return node
```

Part (a) [3 marks]

Draw the tree that results when `func` is called on this tree:



Answer:

Part (b) [3 marks] Write a docstring for `func`:

Stringify the tree by attaching the left subtree to the rightmost child of the right subtree after they have both been recursively stringified.

Part (c) [6 marks]

Implement a function `level_order(r)` that uses a queue to print the values of the nodes at each level of the tree rooted at `Node r`, in order from left to right. For example, when called on the tree from Part (a), `level_order` should return the list `[1, 2, 3, 4, 5, 6]`. Note that we are not working with a BST so even though the result for the tree from Part (a) may look like an in order traversal, that is not what you have to do.

```
def level_order(node):

    print_queue = Queue()
    print_queue.enqueue(node)

    while not print_queue.is_empty():
        next_node = print_queue.dequeue()
        print str(next_node)

        if next_node.left:
            print_queue.enqueue(next_node.left)

        if next_node.right:
            print_queue.enqueue(next_node.right)
```

This page is for rough work and for answers that didn't fit in the space provided.

cont'd...

This page is for rough work and for answers that didn't fit in the space provided.

cont'd...

Short Python function/method descriptions:

```
__builtins__:
    abs(x) -> number
        Return the absolute value of x.
    lambda: expr -> function
        Returns a function that evaluates the Python expression expr.
    len(x) -> integer
        Return the length of the list, tuple, dict, or string x.
    max(L) -> value
        Return the largest value in L.
    min(L) -> value
        Return the smallest value in L.
    open(name[, mode]) -> file object
        Open a file. Legal modes are "r" (read), "w" (write), and "a" (append).
    range([start], stop, [step]) -> list of integers
        Return a list containing the integers starting with start and ending with
        stop - 1 with step specifying the amount to increment (or decrement).
        If start is not specified, the list starts at 0. If step is not specified,
        the values are incremented by 1.
    isinstance(object, class-or-type) -> bool
        Return whether an object is an instance of a class or of type given.
dict:
    D[k] or D.get(k) -> value
        Return the value associated with the key k in D.
    k in D or D.has_key(k) -> boolean
        Return True if k is a key in D and False otherwise.
    D.keys() -> list of keys
        Return the keys of D.
    D.values() -> list of values
        Return the values associated with the keys of D.
file (also called a "reader"):
    F.close()
        Close the file.
    F.read([size]) -> read at most size bytes, returned as a string.
        If the size argument is negative or omitted, read until EOF (End
        of File) is reached.
    F.readline([size]) -> next line from the file, as a string. Retain newline.
        A non-negative size argument limits the maximum number of bytes to return (an incomplete
        line may be returned then). Return an empty string at EOF.
float:
    float(x) -> floating point number
        Convert a string or number to a floating point number, if possible.
int:
    int(x) -> integer
        Convert a string or number to an integer, if possible. A floating point
        argument will be truncated towards zero.
list:
    L.append(x)
        Append x to the end of the list L.
    L.index(value) -> integer
```

```
        Returns the lowest index of value in L.

    L.insert(index, x)
        Insert x at position index.
    L.remove(value)
        Removes the first occurrence of value from L.
    L.sort()
        Sorts the list in ascending order.
str:
    str(x) -> string
        Convert an object into its string representation, if possible.
    S.find(sub[,i]) -> integer
        Return the lowest index in S (starting at S[i], if i is given) where the
        string sub is found or -1 if sub does not occur in S.
    S.index(sub) -> integer
        Like find but raises an exception if sub does not occur in S.
    S.isdigit() -> boolean
        Return True if all characters in S are digits and False otherwise.
    S.replace(old, new) -> string
        Return a copy of string S with all occurrences of the string old replaced
        with the string new.
    S.rstrip([chars]) -> string
        Return a copy of the string S with trailing whitespace removed.
        If chars is given and not None, remove characters in chars instead.
    S.split([sep]) -> list of strings
        Return a list of the words in S, using string sep as the separator and
        any whitespace string if sep is not specified.
    S.strip() -> string
        Return a copy of S with leading and trailing whitespace removed.
```