 Name: Mark F.

Question 1. (4 points) Consider the following Python code.

```
for i in range(n):  -n x
    j = 1
    while j < n:  - log n
        for k in range(n):  - n x
            print( i, j, k)
        j = j * 2
```

4

$O(n^2 \log_2 n)$

+2 $O(n^3)$

What is the big-oh notation $O(\,)$ for this code segment in terms of n?

Question 2. (4 points) Consider the following Python code.

```
for i in range(n):

    for j in range(n):
        print(j)

    k = n
    while k > 0:
        print(k)
        k = k // 2
```

4

$O(n^2)$

(12)

+2 $O(n^2 \log_2 n)$

What is the big-oh notation $O(\,)$ for this code segment in terms of n?

Question 3. (4 points) Consider the following Python code.

```
def main(n):
    for i in range(n):  -n x
        doSomething(n)
    doMore(n)

def doSomething(n):
    for k in range(n):  -n x
        print(k)

def doMore(n):
    for j in range(n * n * n):  - n^3 x
        print(j)

main(n)
```

4

$O(n^3)$

+2 $O(n^4)$

What is the big-oh notation $O(\,)$ for this code segment in terms of n?

Question 4. (8 points) Suppose a $O(n^4)$ algorithm takes 10 second when n = 100. How long would you expect the algorithm to run when n = 1,000?

$T(n) = cn^4 \quad T(100) = c\,100^4 = 10sec$

$c = \frac{10sec}{100^4}$

$c = \frac{10sec}{10^8} = 10^{-7}sec$

$T(1000) = c\,1000^4$

$= 10^{-7}sec\,10^{12}$

$= 10^5 sec$

6

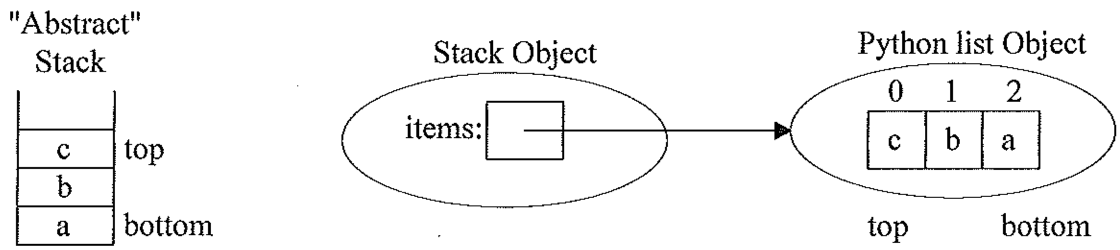
Question 5. (10 points) Why should you design a program instead of "jumping in" and start by writing code?

10

By designing the program first you are able to avoid mistakes and reworking of code, so overall time is saved.

30

Question 6. Consider the following Stack implementation utilizing a Python list:



a) (6 points) Complete the big-oh notation for the Stack methods assuming the above implementation: ("n" is the # items)

6

	push(item)	pop( )	peek( )	size( )	isEmpty( )	__~~init~~__ str
Big-oh	O(n)	O(n)	O(1)	O(1)	O(1)	O(n)

b) (9 points) Complete the code for the pop method including the precondition check.

```
class Stack:

   def __init__(self):
      self._items = []

   def pop(self):
       """Removes and returns the top item of the stack
          Precondition:  the stack is not empty.
          Postcondition: the top item is removed from the stack and returned"""
```

9

if len(self._items) == 0:
    raise ValueError("Cannot pop empty stack!")
return self._items.pop(0)
+1 +2

c) (5 points) Suggest an alternate Stack implementation to speed up some of its operations.

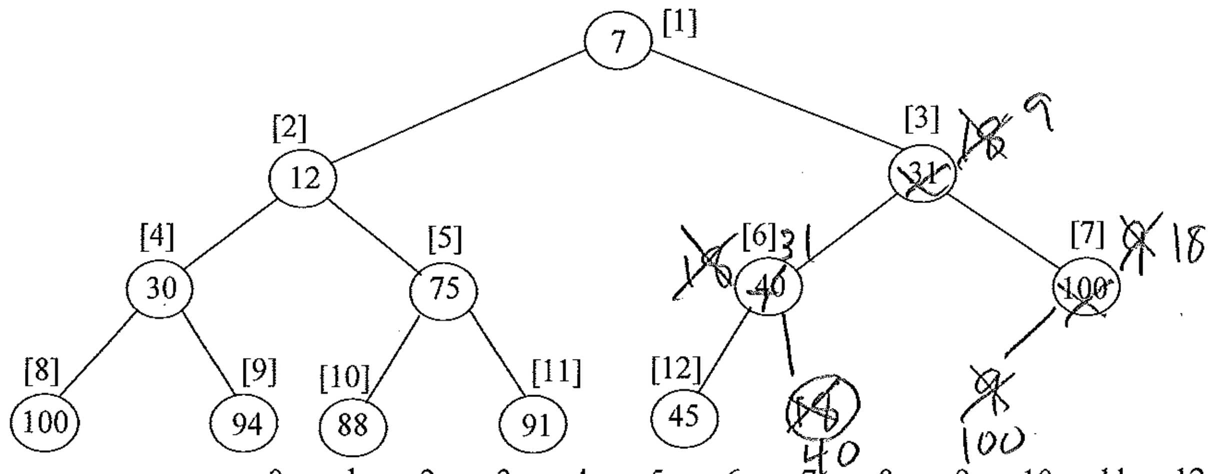
5

Flip the order of the stack so the top is at the largest index, so all operations (except __str__) are O(1).

20

Question 7. Consider the binary heap approach to implement a priority queue. A Python list is used to store a *complete binary tree* (a full tree with any additional leaves as far left as possible) with the items being arranges by *heap-order property*, i.e., each node is ≤ either of its children. An example of a *min* heap "viewed" as a complete binary tree would be:

7



Python List actually used to store heap items

0	1	2	3	4	5	6	7	8	9	10	11	12
not used	7	12	31	30	75	40	100	100	94	88	91	45

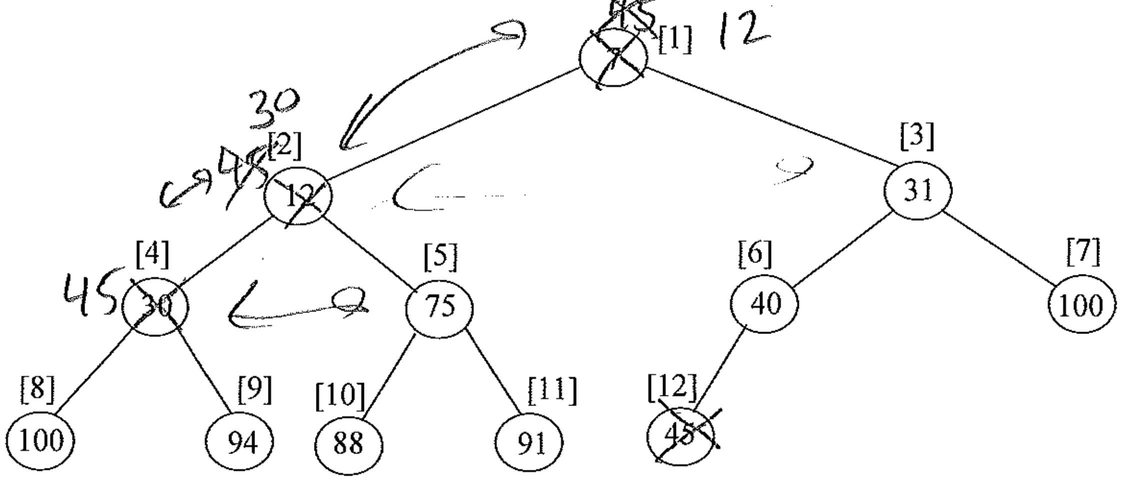
a) (3 points) For the above heap, the list indexes are indicated in [ ]'s. For a node at index *i*, what is the index of:

3
- its left child if it exists: i*2
- its right child if it exists: i*2+1
- its parent if it exists: i//2

b) (7 points) What would the above heap look like after inserting 18 and then 9 (show the changes on above tree)

Now consider the `delMin` operation that removes and returns the minimum item.

7



Python List actually used to store heap items

0	1	2	3	4	5	6	7	8	9	10	11	12
not used	7	12	31	30	75	40	100	100	94	88	91	45

2 c) (2 point) What item would `delMin` remove and return from the above heap? 7

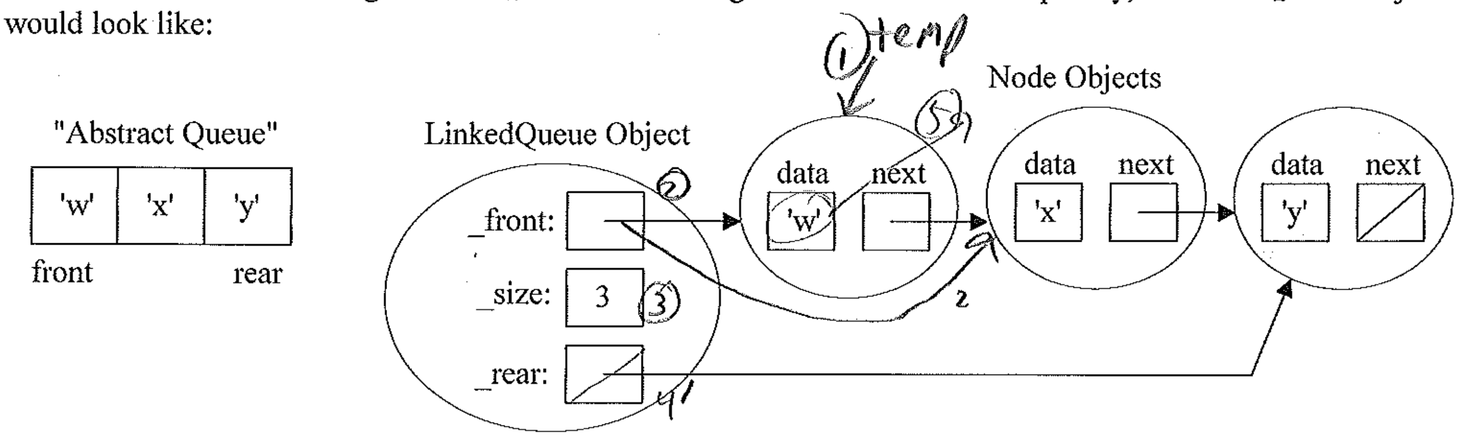
d) (7 points) What would the above heap look like after `delMin`? (show the changes on above tree)

e) (6 points) What is the big-oh notation for the `delMin` operation? **(EXPLAIN YOUR ANSWER)**

6 $O(\log_2 n)$ Moving last item to index 0 takes O(1), percolating down from index 1 cause the index to at least double. You can only double the index 1 $\log_2(n)$ times before it reaches n, so $\log_2 n$

25

Question 8. The `Node` class (in `node.py`) is used to dynamically create storage for a new item added to the stack. Consider the following `LinkedQueue` class using this Node class. Conceptually, a `LinkedQueue` object would look like:



a) (13 points) Complete the `dequeue` method including the precondition check.

```
class LinkedQueue(object):
    """ Linked-list based queue implementation."""

    def __init__(self):
        self._front = None
        self._size = 0
        self._rear = None

    def dequeue(self):
        """ Removes and returns the front item in the queue.
            Precondition:  the queue is not empty. """
        if self._size == 0:
            raise ValueError("Cannot dequeue from empty queue.")
        temp = self._front
        self._front = self._front.getNext()
        self._size -= 1
        if self._size == 0:
            self._rear = None
        return temp.getData()
```

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext
```

13 +3

+ normal case
- temp ptr
- change _front
- change _size

return getData()

special case dequeue last item set _rear.

7

b) (7 points) Assuming the queue ADT described above. Complete the big-oh $O(\,)$ for each queue operation. Let n be the number of items in the queue.

`__init__`	`enqueue(item)` +2	`dequeue( )`	`size()`	`__str__()` +2
O(1)	O(1)	O(1)	O(1)	O(n)

c) (5 points) Would using doubly-linked nodes (i.e., `Node2way`) speed up some of queue operations? Justify your answer. No, slow down operations if "previous" links must be maintained.

5

25