Spring 2015 Data Structures - Test 1 Name: Mark F.

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

```
i = n                                   - log2(n)
while i > 1:
    for j in range(n):                  - n
        for k in range(n * n):          - n^2
            print( i, j, k)
    i = i // 2
```

4

$O(n^3 \log_2 n)$

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):                      - n
    for j in range(n):
        print(j)
    for k in range(n):
        print(k)
```

4

} No nested 2n ⇒ $O(n^2)$ overall

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
        doSomething(n)

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

def doMore(n):
    for j in range(n):                  - n
        print(j)

main(n)
```

4

$O(n^3)$

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

Question 4. (9 points) Suppose a $O(n^3)$ algorithm takes 10 second when n = 1000. How long would you expect the algorithm to run when n = 10,000?

$T(n) = c\,n^3$

$T(1000) = c\,(1000)^3 = 10\text{sec}$

$c = \frac{10\text{sec}}{(1000)^3} = \frac{1\text{ sec}}{10^8}$

$T(10{,}000) = c\,(10{,}000)^3 = c\,10^{12}$

$= \left(\frac{1\text{ sec}}{10^8}\right) 10^{12}$

$= 10^4\text{ sec}$

$= 10{,}000\text{ sec}$

9

Question 5. (9 points) Why should any method/function having a "precondition" raise an exception if the precondition is violated?

The method probably does not work correctly if the precondition is violated, so it is better detect the error immediately. Thus, making it easier to find the error.

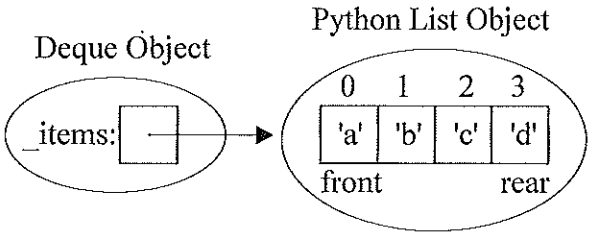
9

+2

30

Question 6. 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. One possible implementation of a Deque would be to use a built-in Python list to store the Deque items such that

- the **front** item is **always stored at index 0**,
- the rear item is always at index len(self._items)-1 or -1



a) (6 points) Complete the big-oh $O(\,)$, for each Deque operation, assuming the above implementation. Let n be the number of items in the Deque.

isEmpty	addRear	removeRear	addFront	removeFront	size
O(1)	O(1)	O(1)	O(n)	O(n)	O(1)

6

b) (9 points) Complete the method for the `removeRear` operation including the precondition check.

```
def removeRear(self):
    """Removes and returns the rear item of the Deque
       Precondition:  the Deque is not empty.
       Postcondition: Rear item is removed from the Deque and returned"""
```

+3

```
if len(self._items) == 0:
    raise ValueError("cannot remove from an empty Deque")
return self._items.pop()
```

9

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

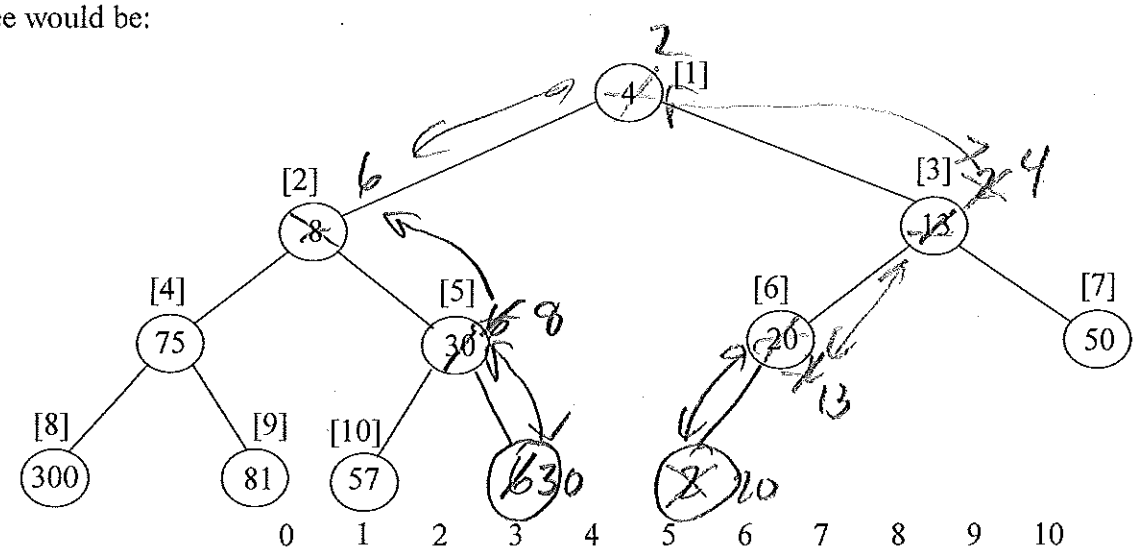
We could use a doubly-linked list of nodes with +3 _front and _rear pointers.

5

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
not used	4	8	13	75	30	20	50	300	81	57

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:

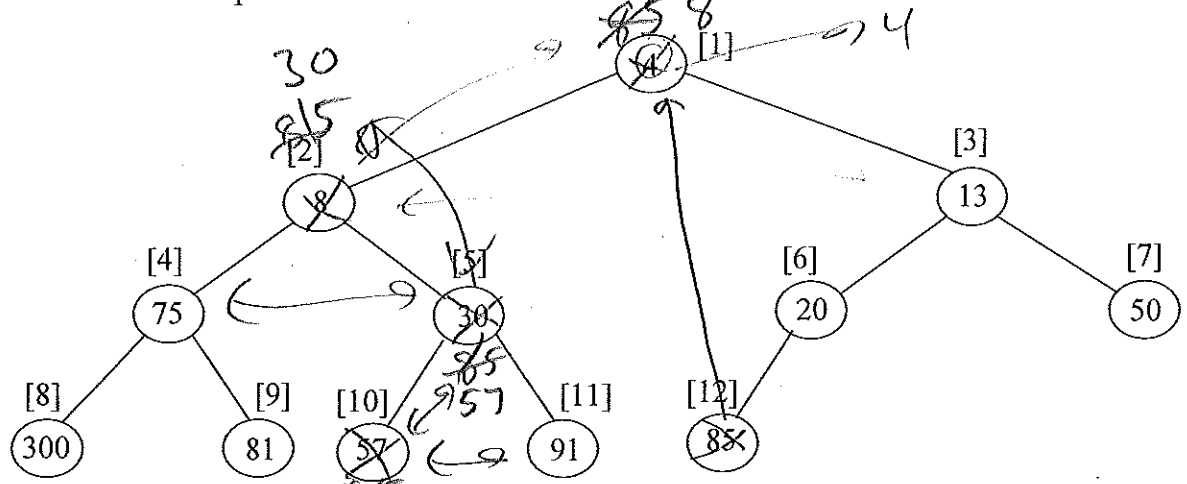
3
- its left child if it exists: $i*2$
- its right child if it exists: $i*2+1$
- its parent if it exists: $i//2$

b) (7 points) What would the above heap look like after inserting 6 and then 2 (show the changes on above tree)

3 c) (3 points) What is the big-oh notation for inserting a new item in the heap? $O(\log_2 n)$

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	4	8	13	75	30	20	50	300	81	57	91	85

2 d) (2 point) What item would `delMin` remove and return from the above heap? 4

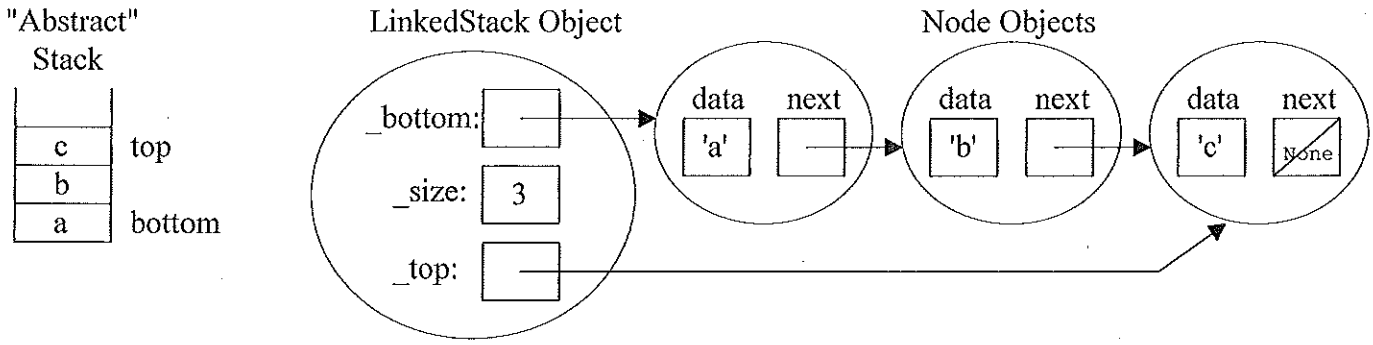
e) (7 points) What would the above heap look like after `delMin`? (show the changes on above tree)

3 f) (3 points) Why does a `delMin` operation typically take longer than an `insert` operation?

Insert only has to compare new item with parent while delMin compares twice per level (ie. two children to find min. and then min. child with item percolating down). Plus, inserted item generally travels fewer levels!

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 `LinkedStack` class using this Node class. Conceptually, a `LinkedStack` object would look like:




```
class LinkedStack(object):
    """ Link-based stack implementation."""

    def __init__(self):
        self._bottom = None
        self._size = 0
        self._top = None


    def __str__(self):
        """ Returns a string with items strung from bottom to
            top. Each item should be separated by a space. """
        resultStr = ""
        temp = self._bottom
        while temp != None:
            resultStr += str(temp.getData()) + " "
            temp = temp.getNext()
        return resultStr
```

```
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
```

11

a) (11 points) Complete the `__str__` method above.

7

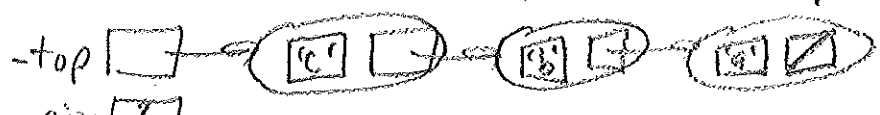
b) (7 points) Assuming the stack ADT described above. Complete the big-oh $O(\,)$ for each stack operation. Let n be the number of items in the stack.

push(item)	pop()	peek()	size()	__str__()
O(1)	O(n)	O(1)	O(1)	O(n)

7

c) (7 points) Suggest an alternate `LinkedStack` implementation to speed up some of its operations.

Reverse the "direction" of the next pointers.



25