Fall 2012 Name: Mark F.

Data Structures - Test 1

Question 1. (5 points) Consider the following Python code.

```
for i in range(n * n):        -n²
    for j in range(0, n, 2):  -n/2
        print (i, j)
```

5

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

$O(n^3)$

$+O(n^2)$ ©3
$O(n^2 \log_2 n)$ ©3

Question 2. (5 points) Consider the following Python code.

```
i = 1
while i <= n:        - log2 n

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

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

    i = i * 2
```

$O(n^2 \log_2 n)$

$O(\log_2 n^3)$ ©4

5

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

Question 3. (5 points) Consider the following Python code.

```
for i in range(n):   - n

    j = n
    while j > 1:      - log n
        print(j)
        j = j // 2

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

Note: not nested!

$O(n^2)$

5

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

$+3\ O(n \log_2 n)$
$+3\ O(n^2 \log_2 n)$

Question 4. (10 points) Suppose a $O(n^2)$ algorithm takes 10 seconds when n = 1,000. How long would you expect the algorithm to run when n = 10,000?

$O(n^2) \Rightarrow T(n) \doteq cn^2$ $\quad T(1{,}000) = c\,1000^2 = 10\text{sec}$ $\quad c = \frac{10\text{sec}}{1000^2}$

10

$c = \frac{10}{10^6} = 10^{-5}$ $\quad T(10{,}000) = c\,10{,}000^2 = \frac{10^8}{10^5} = 10^3 = 1000\text{ sec.}$

Question 5. (10 points) Why should any method/function having a "precondition" raise an exception if the precondition is violated?

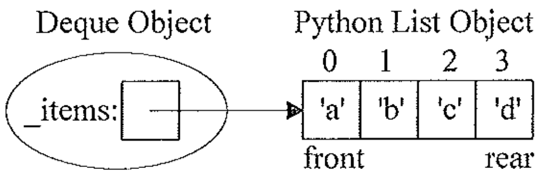
10

This helps in debugging the program since the error is immediately detected. Otherwise, the error might be detected later in the execution so the real error will be hard to track down.

35

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.

6

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

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

9

+4

+4

+3 -0

+1

alternate Deque implementation

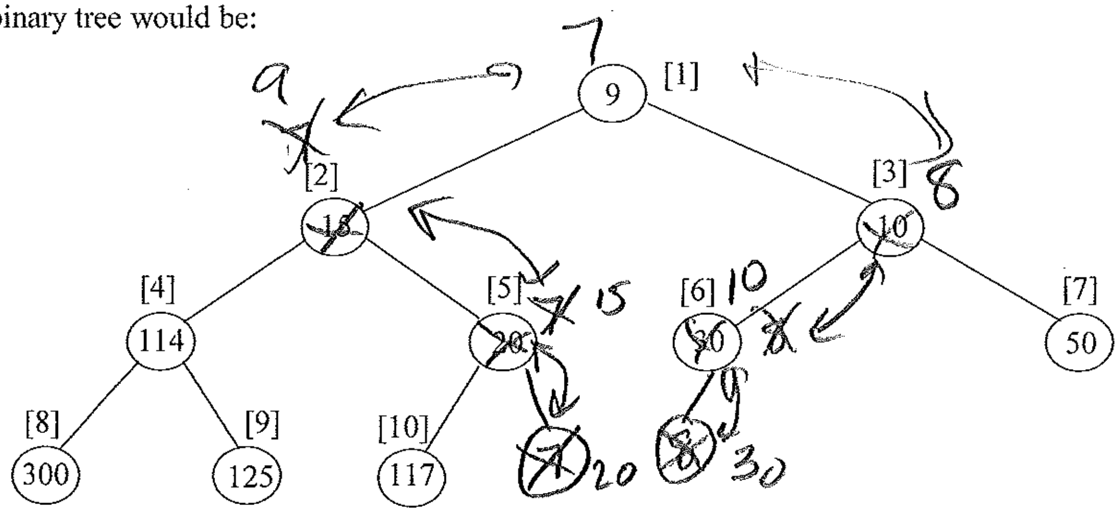
c) (5 points) Suggest an i~~mprovement to the above Python List implementation of the Deque~~ to speed up some of its operations.

Use a doubly-linked list of Node with _front and _rear pointers, All operations are O(1), except __str__

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" an a complete binary tree would be:



Python List actually used to store heap items

0	1	2	3	4	5	6	7	8	9	10
not used	9	15	10	114	20	30	50	300	125	117

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:

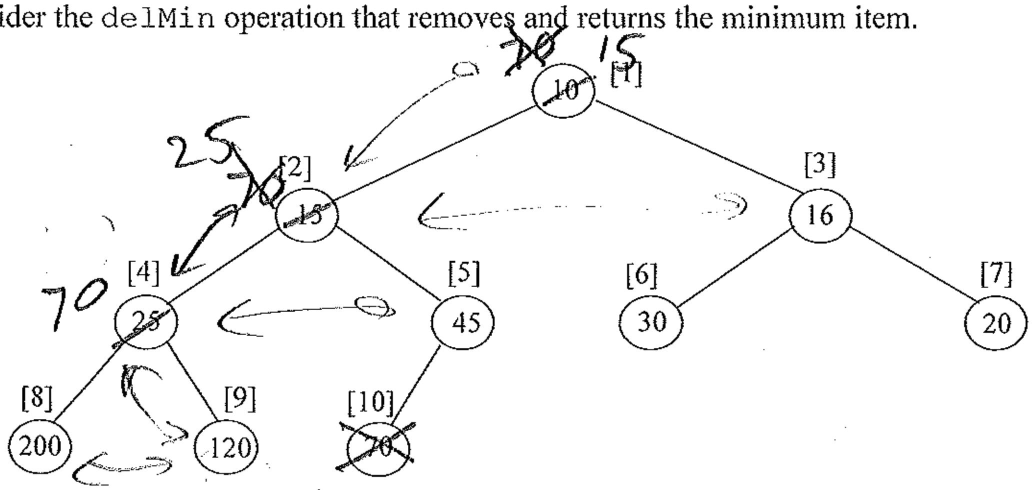
- its left child if it exists: $i * 2$
- its right child if it exists: $i * 2 + 1$
- its parent if it exists: $i // 2$

b) (6 points) What would the above heap look like after inserting 7 and then 8 (show the changes on above tree)

c) (2 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.



Python List actually used to store heap items

0	1	2	3	4	5	6	7	8	9	10
not used	10	15	16	25	45	30	20	200	120	70

d) (1 point) What item would `delMin` remove and return from the above heap? 10

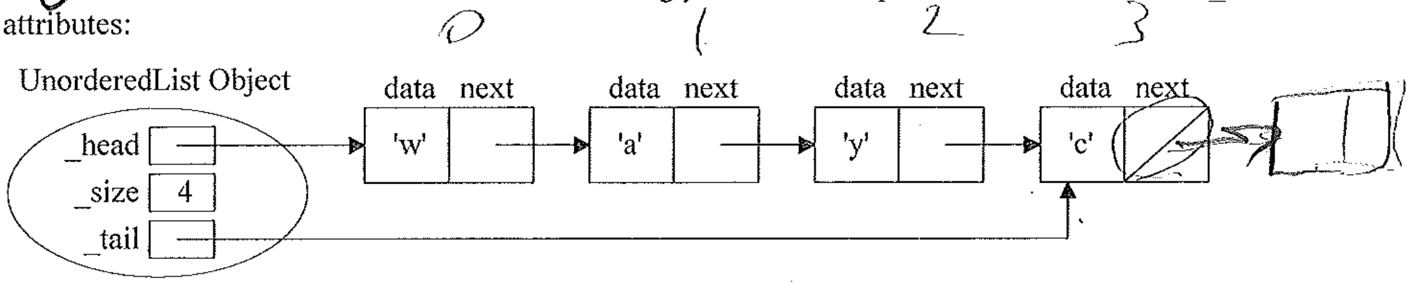
e) (6 points) What would the above heap look like after `delMin`? (show the changes on above tree)

f) (2 points) What is the big-oh notation for `delMin`? $O(\log_2 n)$

10

Fall 2012 Name: ______________________

Question 8 The textbook's unordered list ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:



a) (15 points) The `insert(position, item)` method adds the `item` to the list at the specified `position`. Unlike the textbook's implementation, ASSUME that the list may contain duplicate items!!! The precondition is that `position` is a nonnegative integer. If `position` is 0, then add it to the head of the list. If `position` is `_size` or bigger, then add it to the tail of the list. Complete the `insert(position, item)` method code including the precondition check.

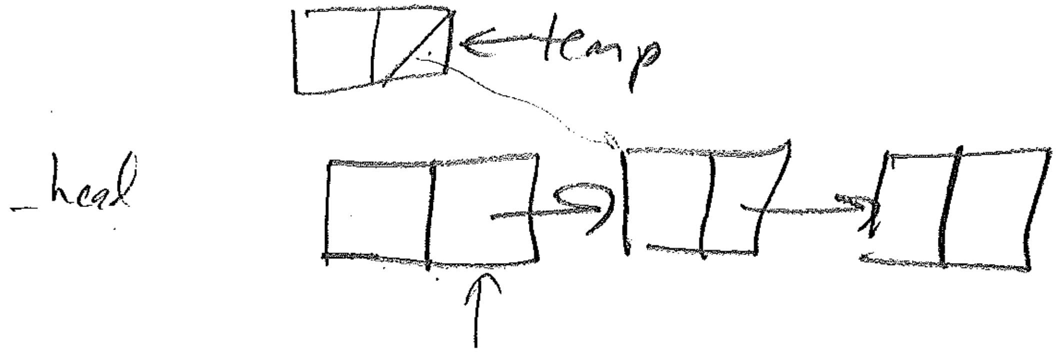
```
class UnorderedList:

   def __init__(self):
      self._head = None
      self._size = 0
      self._tail = None

   def insert(self, position, item):
        if not isinstance(position, int):   +1
            raise TypeError("Position of list must be an integer")
        if position < 0:  +2
            raise ValueError("Position must be nonnegative")
        temp = Node(item) +2
        if self._size == 0:  +2
            self._head = temp
            self._tail = temp
        elif position >= self._size:
            self._tail.setNext(temp)   +2
            self._tail = temp
        else:
            current = self._head
            (See attached)
```

b) (10 points) Assuming the unordered list ADT described above **that allows duplicate items**. Complete the big-oh $O(\,)$ for each operation. Let n be the number of items in the list.

`insert(position,item)`	`pop()` removes and returns tail item	`length()` returns number of items in list	`append(item)` adds item to the tail of list	`add(item)` adds item to the head of list
$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$




```
else:
    current = self._head                          +1
    for count in range(position-1):
        current = current.getNext()               +2
    temp.setNext(current.getNext())               +1
    current.setNext(temp)                         +1

self._size+=1  +1
```

15