Spring 2013 Name: Mark F.

Data Structures - Test 1

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

```
for i in range(n):  — n
    j = 1
    while j < n:  — log2n
        for k in range(n): ← n
            print (i, j, k)
        j = j * 2
```

5

$O(n^2 \log_2 n)$

$(+3\ n\log_2 n)$

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

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

```
i = 2**n    # this is 2ⁿ
while i > 1:  ←
    for j in range(n): ← n
        print(j)

    i = i // 2
```

i's values ; $2^n, 2^{n-1}, 2^{n-2}, \ldots, 2^2, 2^1$ ← loops n times

5

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

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

Question 3. (5 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
        print(k)

main(n)
```

5

$O(n^2)$

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

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

$T(n) = c\,n^4$ $T(1000) = c\,1000^4 = 10\,sec$ $c = \frac{10\,sec}{1000^4} = 10^{-11}\,sec$

10

$T(10{,}000) = c\,10000^4 = 10^{-11}\,sec\ 10000^4 = 10^{-11}\,sec \times 10^{16} = 10^5\,sec$

$= 100{,}000\,sec$

$= 27.78\,hrs$

(+2 100 sec)

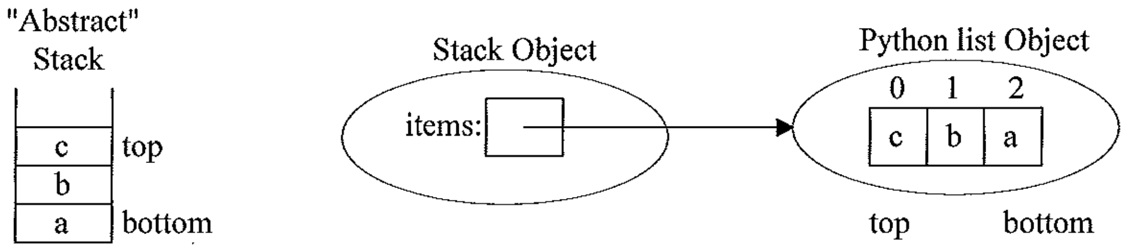
Question 5. (10 points) Why should you design a program instead of "jumping in" and start writing code?

By designing a program you can split the work into smaller, managable pieces. It is easier to implement/code, and debug/test the smaller pieces.

10

35

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



a) Complete the big-oh notation for the ~~"alternate"~~ Stack methods: ("n" is the # items)

6

	push(item)	pop( )	peek( )	size( )	isEmpty( )	__init__
Big-oh	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)

b) (9 points) Complete the method for the pop operation 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 +2

```
         if len(self.items) == 0:
              raise ValueError("cannot pop from empty stack")

         return self.items.pop(0)
```

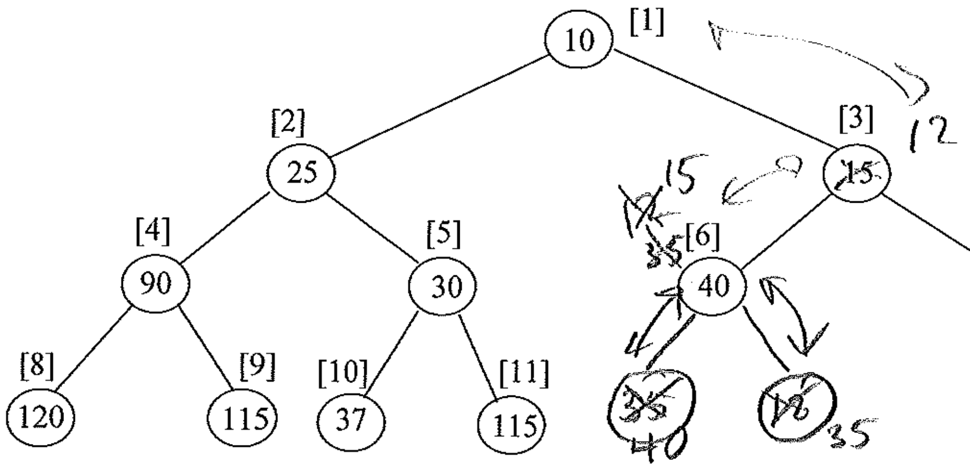
c) (5 points) Suggest an alternate Stack implementation to speed up some of its operations.

5

Reverse the top and bottom, i.e., let index 0 hold the bottom item of the stack

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~~ as 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	11
not used	10	25	15	90	30	40	50	120	115	37	115

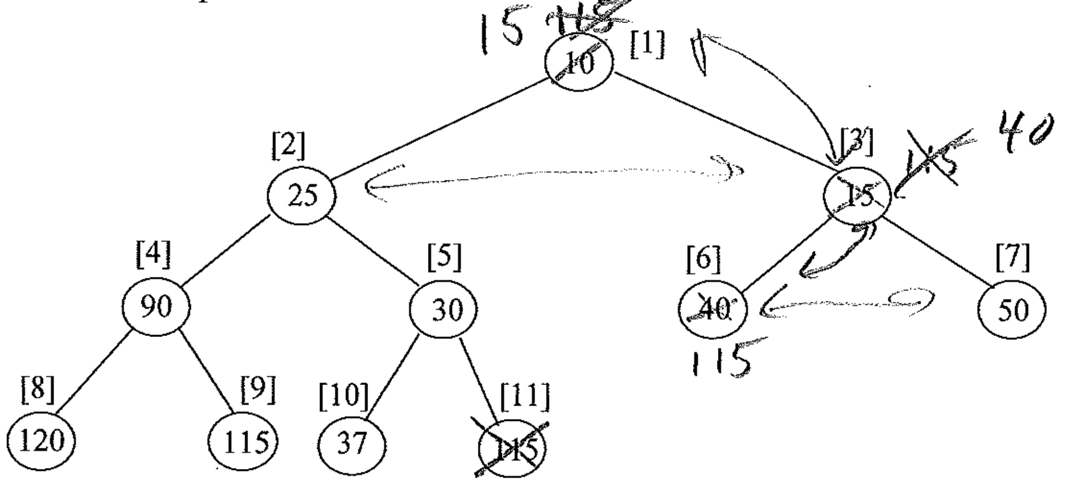
a) (3 points) For the above heap, the list indexes are indicated in [ ]'s. For a node at index *i*, what is the index of:

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

6 b) (6 points) What would the above heap look like after inserting 35 and then 12 (show the changes on above tree)

2 c) (2 points) What is the big-oh notation for inserting a new item in the heap? O(log n)

11 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	11
not used	10	25	15	90	30	40	50	120	115	37	115

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

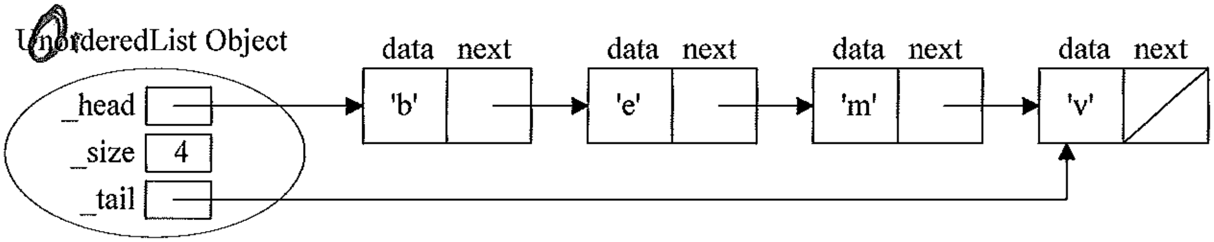
6 e) (6 points) What would the above heap look like after `delMin`? (show the changes on above tree)

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

9

20

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



a) (15 points) The `pop(position)` method removes and returns the item at the specified `position`. The precondition is that `position` is a nonnegative integer corresponding to an actual list item (e.g., for the above list 0 ≤ `position` ≤ 3). Complete the `pop(position)` method code including the precondition check.

```
class OrderedList:

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

   def pop(self, position):
      if not isinstance(position, int):
+1       raise TypeError("Pop position must be an integer")
      if position < 0 or position >= self._size:
+1       raise IndexError("Pop position outside value range")
+1    previous = None
      current = self._head
+4    for count in range(position):
          previous = current
          current = current.getNext()
+2    if previous == None:
          self._head = self._head.getNext()
      else:
+1       previous.setNext(current.getNext())
      if current == self._tail:
+2       self._tail = previous
+1    self._size -= 1
+2    return current.getData()
```

(15)

b) (10 points) Assuming the ordered list ADT described above. Complete the big-oh $O()$ for each operation. Let n be the number of items in the list.

+2 each

(10)

`pop(position)` removes and returns the item at the specified position	`pop()` removes and returns tail item	`length()` returns number of items in the list	`index(item)` returns the position of item in the list	`add(item)` adds item to its sorted spot in the list
$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$

100

15