Fall 2010 Name: Mark F.

Data Structures - Test 1

Question 1. (10 points) Determine the theta notation $\theta(\,)$ for the following Python code.

(+5 for $\theta(n)$
+7 for $\theta(n\log n)$)

```
for i in xrange(n):
    for j in xrange(i):
        sum = i + j
    # end for j
# end for i
```

loops n times

loops: $0+\underbrace{1+\underbrace{2+3+\cdots+(n-2)}_{n}+(n-1)}_{n} = \underbrace{n+n+\cdots+n}_{\frac{(n-1)}{2}\text{ times}}$

$$= n\frac{(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \text{ is } \theta(n^2)$$

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

(+2 for 100 sec.)

$\theta(n^2)$ means Exec. time $T(n) \approx cn^2$

$$T(1000) = c\,1000^2 = 10\text{ sec} \qquad c = \frac{10}{1000^2} = \frac{10}{(10^3)^2} = 10^{-5}\text{ sec}$$

$$T(10{,}000) = c\,10000^2 = (10^{-5})(10^4)^2 = 10^{-5}10^8 = 10^3\text{ sec}$$

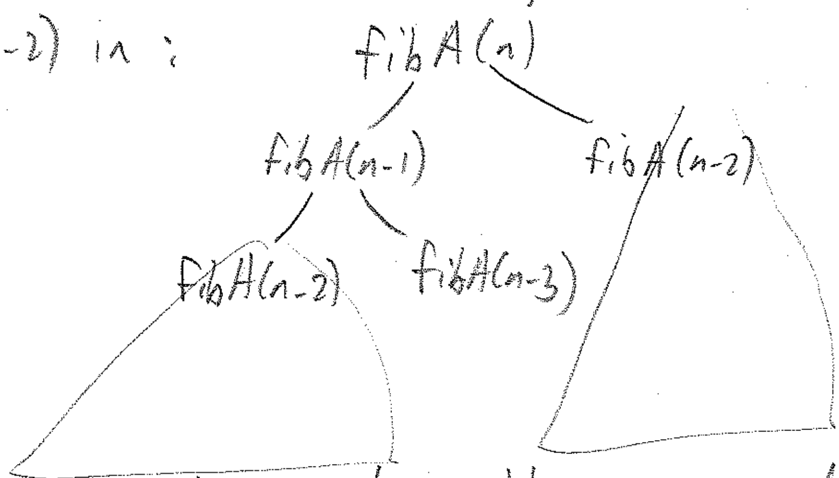
$= 1{,}000$ sec

Question 3. (15 points) For the two implementations of fibonacci given below, explain why fibA is so much slower than fibB.

```
def fibA(n):
    if n <= 1:
        return n
    else:
        return fibA(n-1) + fibA(n-2)
```

```
def fibB(n):
    fibs = [0, 1]
    for i in range(2,n+1):
        fibs.append(fibs[i-1]+fibs[i-2])
    return fibs[n]
```

The fibA implementation solves the same smaller problems over and over during the recursion. For example, fibA(n-2) in:



FibB however solves each problem once and stores the answer in the list fibs so these answers can be looked up when needed again.

15

Question 4. (5 points) What is the difference between unit testing and integration testing?

In Unit testing you test each module (e.g. class, function, etc.) separately. In integration testing you put these units together and test them together.

5

Question 5. (15 points)

a) In the following recursive binary search code, what would be a precondition on the binarySearch function?

(5) The list myList is in ascending order.

```
def binarySearch(myList, target):
    """Returns the position of the target in myList or -1 if not found"""

    def binarySearchHelper(myList, target, first, last):
       print "first is", first, "last is", last
       if first > last:
           return -1            # -1 indicates target not found in myList
        else:
           midpoint = (last+first)/2
           if myList[midpoint] == target:
               return midpoint
           elif target < myList[midpoint]:
               return binarySearchHelper(myList, target, first, midpoint-1)
           else:
               return binarySearchHelper(myList, target, midpoint+1, last)

    return binarySearchHelper(myList, target, 0, len(myList)-1)
```

b) Show the output of the following program which calls `binarySearch`. (INCLUDE the output of the debugging `print` statement in the `binarySearchHelper` function)

(10)

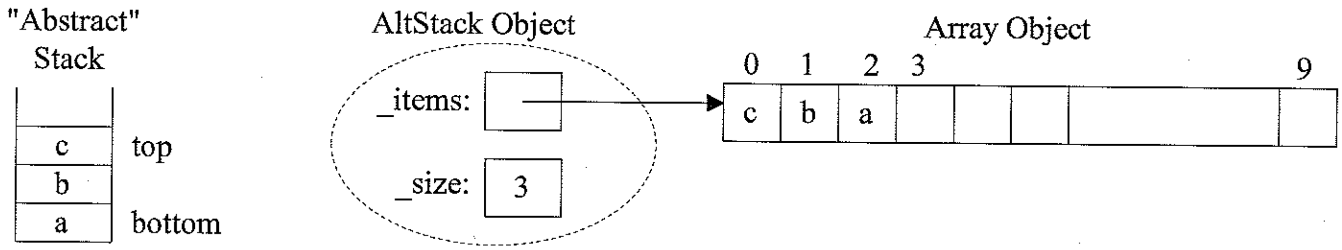
```
aList = [10, 20, 30, 40, 50, 60, 70, 80]
print "The list is: ",aList
target = 50
location = binarySearch(aList, target)
if location == -1:
    print target, "NOT founde"
else:
    print target, "FOUND at index", location
```

Output of the above program which calls `binarySearch`:

The list is: [10, 20, 30, 40, 50, 60, 70, 80]
first is 0 last is 7
first is 4 last is 7
first is 4 last is 4
50 FOUND at index 4

20

Question 6. (25 points) Consider the following `AltStack` class that uses an Array to store the items in the stack. The "top" item on the stack is always stored at index 0. (NOTE: this is different from the `ArrayStack` class of section 14.4)



a) Complete the theta notation $\theta()$ for each stack methods of the above `AltStack` implementation: (Let us define "n" as the # items in the stack)

	__init__ (constructor)	push(item)	pop( )	peek( )	len( )	isEmpty( )
(8) Theta notation	$\theta(1)$ +1	$\theta(n)$ +2	$\theta(n)$ +2	$\theta(1)$ +1	$\theta(1)$ +1	$\theta(1)$ +1

b) Assume that the array size DOES NOT grow during the `push` method, but has a fixed physical capacity from the `__init__` constructor. What would be the precondition on the `push` method.

(5) The stack is not full.

c) Write the code for the `push` method of the `AltStack` class.

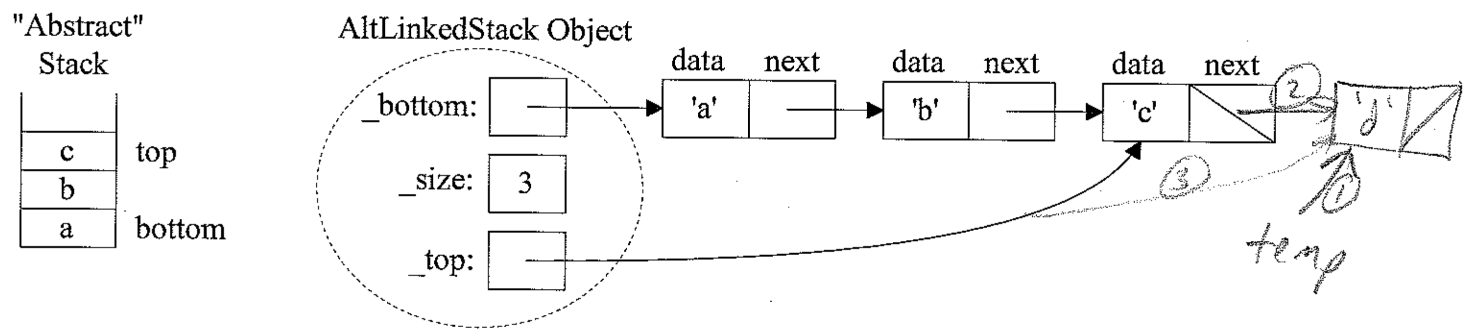
```
def push(self, newItem):
    """Inserts newItem at the top of stack."""
    for index in xrange(len(self._items), 0, -1):          +7
(12)    self._items[index] = self._items[index-1]

    self._items[0] = newItem      +3
    self._size += 1      +2
```

```
""" File: node.py   Node class for one-way linked structures.  """
class Node(object):

    def __init__(self, data, next = None):
        """Instantiates a Node with default next of None"""
        self.data = data
        self.next = next
```

Question 7. (20 points) Consider the following `AltLinkedStack` class which uses the `Node` class (from the text and listed above) to dynamically create storage for a new item added to the stack. Conceptually, an `AltLinkedStack` object would look like the below picture. (NOTE: this is different from the `LinkedStack` class in section 14.4)



a) Complete the theta notation $\theta()$ for each stack methods of the above `AltLinkedStack` implementation: (Let us define "n" as the # items in the stack)

8

	__init__ (constructor)	push(item)	pop( )	peek( )	len( )	isEmpty( )
Theta notation	$\theta(1)$	$\theta(1)$ 2	$\theta(n)$ 2	$\theta(1)$	$\theta(1)$	$\theta(1)$

b) Write the code for the push method of the `AltLinkedStack` class.

```
def push(self, newItem):
    """Inserts newItem at the top of stack."""
    temp = Node(newItem)          +3
    if self._size == 0:
        self._bottom = temp
    else:                          +4
        self._top.next = temp
    self._top = temp              +3
    self._size += 1               +2
```

12