Name: _________________________________ Date: ___________________

Pledge: ____

Closed book: no textbook, no electronic devices, one sheet of paper with notes. Read each question carefully before answering! Write your answers on the test paper and turn in your notes.

Question 1 (5 points) Assess: [execution]
Consider the following code:

```
score = {}
score['Brian'] = 76
score['Ammar'] = 95
score['Brian'] = 96
print 'Nairb' in score or 'Ramma' in score
print score['Brian']
```

What is printed on the screen after these statements have executed?
False
96

Rubric: (3 points for each correct value, up to a maximum of 5 points)

Question 2 (15 points) Assess: [execution]

(a) What is the binary representation of twenty three (i.e., 23_{10})? Write it using exactly 8 bits.

00010111
Rubric: (2 points for 8 bits, 3 points for correct value)

(b) Using two's complement with exactly 8 bits, what is the binary representation of negative 19 (i.e., -19_{10})?

11101101
Rubric: (5 points for correct answer, or else 2 points if correct method of computation was used)

(c) Using your answers from (a) and (b), what is 23_{10} - 19_{10} in **binary**? Perform the operation using **addition in binary with 8 bits**. Show your work. No credit will be given otherwise.

```
 00010111
+11101101
---------------
100000100, but 8 bits means the leading one is omitted, so 00000100
```

Rubric: (5 points for correct answer, 3 points if answer contained leading one, or else 2 points if correct method of computation was used)

Question 3 (15 points) Assess: [design]
Implement the following function, using recursion on L. That means you can access L only through the expressions L[0], L == [], and L[1:].

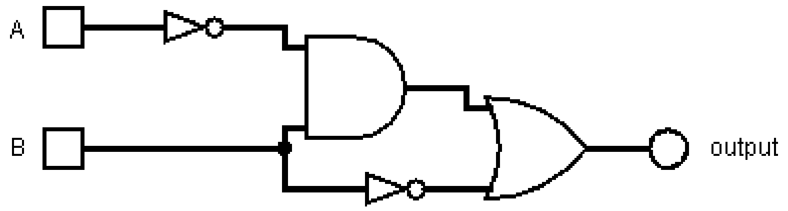
```
def take_until(f, L):
    '''Assume L is a list and f is a function that returns True or False.

    Returns the elements of L while f is False, up to but not including the
    first element that makes f True.

    Example 1:
    take_until(lambda x: x >= 5, [-1, 0, 1, 4, 5, 3, 2, 1]) => [-1, 0, 1, 4]

    Example 2:
    take_until(lambda x: x % 2 == 1, [-2, 0, 5, 6, 7]) => [-2, 0]
    '''

    if L == [] or f(L[0]):
        return []
    return [L[0]] + take_until(f, L[1:])
```

Rubric: (
3 points for syntactically reasonable attempt,
3 points for base case condition L == [],
3 points for base case condition f(L[0]),
3 points for [L[0]] in return statement,
3 points for proper recursive call [1 point for take_until, 1 point for f, 1 points for L[1:]]
)

Question 4 (10 points) Assess: [testing]
Implement two PyUnit tests for the `take_until` function you wrote in question 3. `test1` should cover example 1 from the docstring, and `test2` should cover example 2. Assume the `take_until` function appears in module `testmod2`. Fill in the blanks in the PyUnit script:

```
import unittest
import testmod2

class Test(unittest.TestCase):

    def test1(self):
        self.assertEqual(
            testmod2.take_until(lambda x: x % 2 == 1, [-2, 0, 5, 6, 7]),
            [-1, 0, 1, 4])

    def test2(self):
        self.assertEqual(
            testmod2.take_until(lambda x: x % 2 == 1, [-2, 0, 5, 6, 7],
            [-2, 0])

if __name__ == "__main__":
    unittest.main()
```

Question 5 (10 points)
Consider the following circuit:



(a) Write out the expression for the circuit using the boolean notation we discussed in class (not Python syntax).

$\overline{A}B + \overline{B}$

(b) What is the output when A = 1 and B = 0?

1 (or True)

Rubric: (5 points for each part)

Question 6 (20 points) Assess: [coding]
Complete the implementation of this function for subset sum with the "*use it or lose it*" strategy.

```
subset(2500, [27, 24, -1, 6, -2]) -> False
subset(25, [27, 24, -1, 6, -2]) -> True      because of [27, -2]
subset(50, [27, 24, -1, 6, -2]) -> True      because of [27, 24, -1]
```

```
def subset(target, L):
    '''Determines whether or not it is possible to create target sum using
    the values in the list. Values in list can be positive, negative, or
    zero. Returns True if possible, False otherwise.'''
    if target == 0:
        return True
    if L == []:
        return False

    use_it = subset(target - lst[0], lst[1:])
    lose_it = subset(target, lst[1:])
    return use_it or lose_it
```

Rubric: (2 points for correct syntax, 6 points for lose it, 6 points for use it, 6 for returning either use it or lose it)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 7 (10 points)
' Implement this function using recursion.
' Lucas numbers follow a pattern similar to Fibonacci numbers, except that the
' first two numbers are 2 and 1. The sequence looks as follows:
' 2, 1, 3, 4, 7, 11, 18, ...
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
def lucas(n):
    '''Returns the nth Lucas number. The 0th number is 2, the 1st number is 1,
    and any number beyond that is the sum of the previous two numbers.
    Examples:
    lucas(0) -> 2
    lucas(2) -> 3
    lucas(4) -> 7'''
    if n == 0:
        return 2
    if n == 1:
        return 1
    return lucas(n - 1) + lucas(n - 2)
```

Rubric: (2 points for each correct base case, 3 points for each correct recursive call)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 8 (20 points)
' Implement this function using recursion and memoization.
' Lucas numbers follow a pattern similar to Fibonacci numbers, except that the
' first two numbers are 2 and 1. The sequence looks as follows:
' 2, 1, 3, 4, 7, 11, 18, ...
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
def lucas_memo(n):
    '''Returns the nth Lucas number. The 0th number is 2, the 1st number is 1,
    and any number beyond that is the sum of the previous two numbers.
    Uses memoization to improve performance.'''
    def lucas_helper(n, memo):
        if n in memo:
            return memo[n]

        if n == 0:
            result = 2
        elif n == 1:
            result = 1
        else:
            result = lucas_helper(n - 1, memo) + lucas_helper(n - 2, memo)

        memo[n] = result
        return result

    return lucas_helper(n, {})
```

Rubric: (
5 points for checking if the key is in the memo and returning it,
10 points for performing computation and storing the result in a variable (according to the rules in Q7),
5 points for storing and returning the result)