CS 61A Structure and Interpretation of Computer Programs

Spring 2017

Final Solutions (revised)

INSTRUCTIONS

- You have 3 hours to complete the exam.
- The exam is open book, open notes, closed computer, closed calculator.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
CalCentral email (`@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
Room in which you are taking exam	
Seat number in the exam room	
I pledge my honor that during this examination I have neither given nor received assistance. **(please sign)**	

Reference Material.

```
# Linked Lists

class Link:
    """A linked list cell.
    >>> L = Link(0, Link(1))
    >>> L.first
    0
    >>> L.rest
    Link(1)
    >>> L.first = 2
    >>> L
    Link(2, Link(1))
    >>> L.rest = Link.empty
    >>> L
    Link(2)
    """
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest is Link.empty:
             return "Link({})".format(self.first)
        else:
             return "Link({}, {})".format(self.first, self.rest)

# Trees

class Tree:
    """A tree node."""

    def __init__(self, label, branches=[]):
        for c in branches:
            assert isinstance(c, Tree)
        self.label = label
        self.branches = branches

    def is_leaf(self):
        return not self.branches

class BinTree:
    empty = ()

    def __init__(self, label, left=empty, right=empty):
        self.label = label
        self.left = left
        self.right = right
```

```
# Streams

class Stream:
   """An infinite, lazily computed linked list."""
   def __init__(self, first, rest):
       """A stream whose .first element is FIRST and whose .rest is either
       REST, if it is a Stream, or else the Stream resulting from calling
       REST().  In the latter case, the function is called only once when
       first needed, and its result is memoized."""

def combine_streams(func, sa, sb):
    """Return the stream of values (FUNC(sa1, sb1), FUNC(sa2, sb2), ...)
    where SA is (sa1, sa2, ...) and SB is (sb1, sb2, ...)."""

def filter_stream(pred, s):
    """Return the substream of S consisting of values, si, for
    which PRED(si)."""

def make_integer_stream(start):
    """Return the stream containing START, START+1, START+2, ...."""

def stream_to_list(s, n):
    """Return a list containing the first N elements of Stream S."""
```

1. (8 points) What Would Python Do?

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error (exception) occurs, write "`Error`". If an expression yields (or prints) a function, write "`<Function>`". The first two rows have been provided as examples.

Important: The statements in the table are cumulative—assume that all preceding statements in the table have been executed before each entry.

Assume that `python3` has executed the statements on the left initially:

```
def g(L):
    def k():
        yield iter(L)
        yield iter(L)
    return k
def inc(x):
    x += 1
    return x
def f1():
    def h(x):
        y = x
    return h
def f2():
    def h(x):
        S[2] = x
    return h
S = [1, 2, 3]
p = g(S)()
```

Expression	Interactive Output
`pow(2, 3)`	`8`

`print(4, 5) + 1`	`4 5`
	`Error`

`p[0]`	`Error`

`t1 = next(p)`
`t2 = next(p)`
`next(t1)`
	`1`

`list(t1)`	`[2, 3]`
`next(t1)`	`Error`
`next(t2)`	`1`

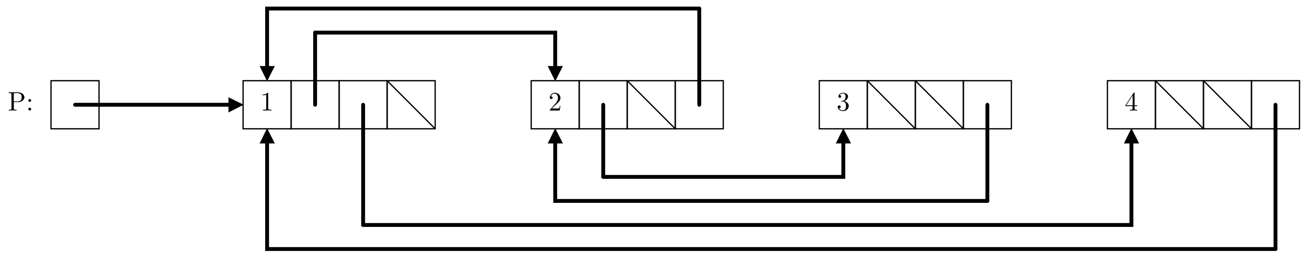
`y = 4`
`z = inc(y)`
`print(z, y)`
	`5 4`

`y = 4`
`z = f1()(6)`
`print(z, y)`
	`None 4`

`f2()(6)`
`a = next(t2)`
`b = next(t2)`
`print(a, b)`
	`2 6`

2. (8 points) What's the Point(er)?

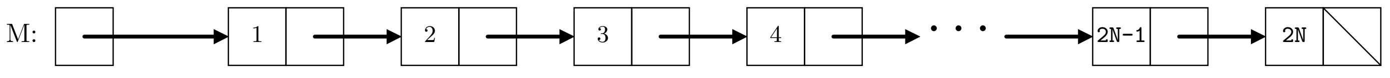
(a) (4 pt) Fill in code below that, when executed, yields the situation shown in the diagram. The composite boxes in the diagram are Python list objects. Single boxes with labels to their left denote variables, not list objects. Boxes with diagonal lines through them contain `None`. You might not need the second blank line.

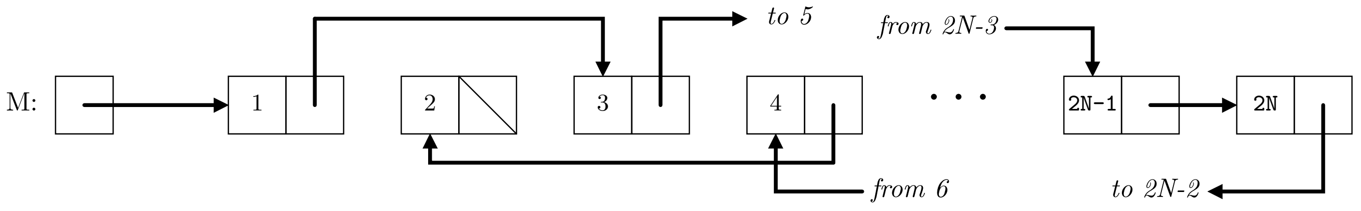


```
P = [1, [2, [3, None, None, None], None, None], [4, None, None, None], None]

P[1][3] = P

P[2][3] = P

P[1][1][3] = P[1]
```

(b) **(4 pt)** In the top diagram below, the two-slot objects are `Links` (see page 2). There are $2N > 0$ of these `Link` objects altogether. Slots with diagonal lines through them contain `Link.empty`. Fill in the blanks to modify the boxes on the top into the state shown in the bottom diagram. That is, modify the list `M` destructively so that node k points to node $k+2$ if k is odd, and to node $k-2$ if k is even. The only exceptions are that node $2N-1$ continues to point to node $2N$, and the resulting list ends at node 2.

Do *not* create any new `Links` and do not modify any `.first` fields.



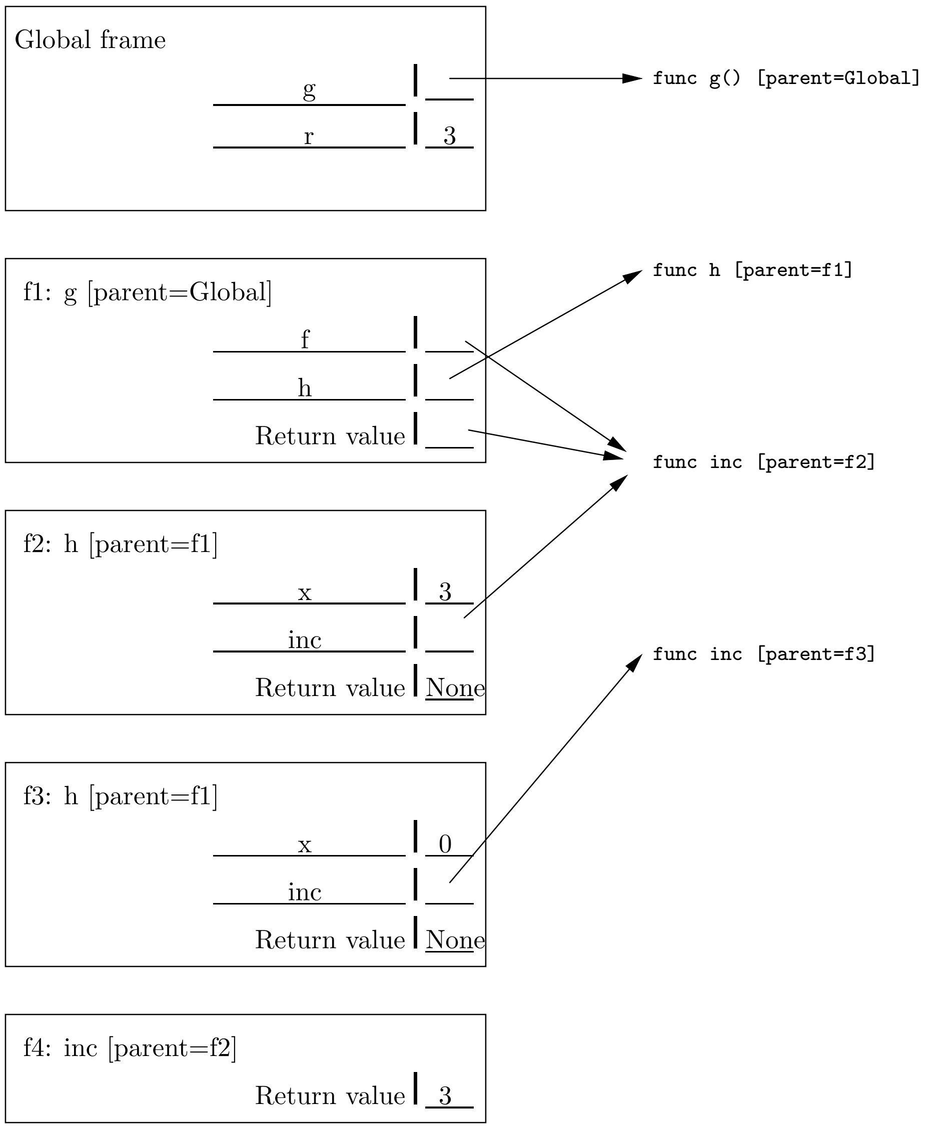


```
def relink(L, prev):

    t = L.rest

    if t.rest is not Link.empty:

        L.rest = relink(t.rest, t)

    t.rest = prev

    return L

M = relink(M, Link.empty)
```

3. (8 points) Environmental Issues

(a) (5 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished. Fill in the function values (`func ...[parent=...]`) with all function values created during execution.

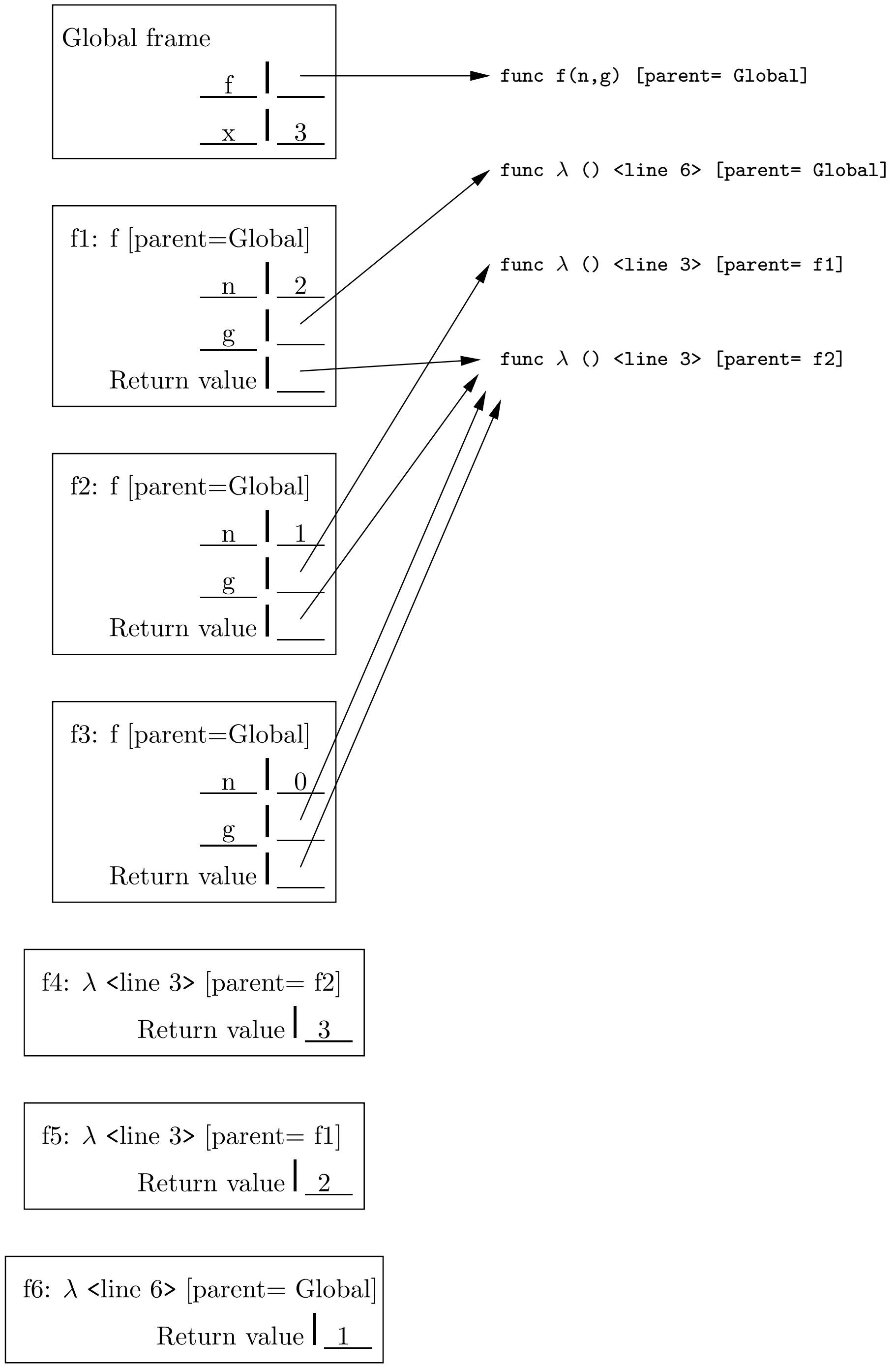
```
def g():
    f = None
    def h(x):
        nonlocal f
        def inc():
            nonlocal x
            x += 1
            return x
        if x > 0:
            f = inc
            h(x-2)
    h(2)
    return f
r = g()()
```



(b) (3 pt)

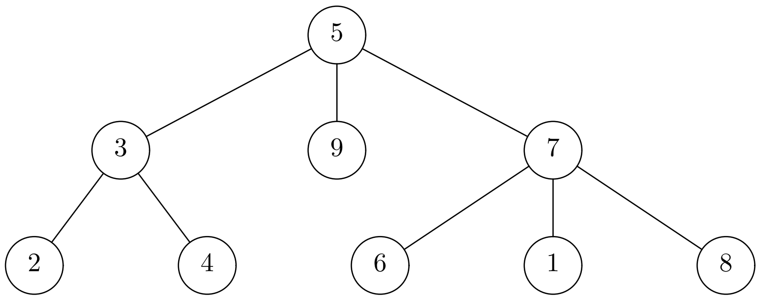
The diagram on the right below shows an environment diagram resulting from execution of a particular program. Fill in the blanks in the program on the left so as to create a situation consistent with the diagram on the right (do not change any of the environment diagram.)

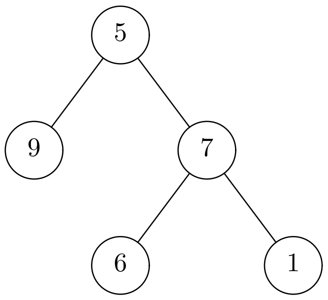
```
1 def f(n, g):

2   if n > 0:

3     return f(n-1, lambda: g()+1)

4   else:

5     return g

6 x = f(2, lambda: 1)()
```


Global frame
f
x
3
func f(n,g) [parent= Global]
func λ () <line 6> [parent= Global]
func λ () <line 3> [parent= f1]
func λ () <line 3> [parent= f2]
f1: f [parent=Global]
n
2
g
Return value
f2: f [parent=Global]
n
1
g
Return value
f3: f [parent=Global]
n
0
g
Return value
f4: λ <line 3> [parent= f2]
Return value
3
f5: λ <line 3> [parent= f1]
Return value
2
f6: λ <line 6> [parent= Global]
Return value
1

4. (8 points) Spring Pruning

Fill in the function `prune` below so that given a Tree (see page 2) with integer labels, it (destructively) deletes all nodes whose label is strictly less than that of their parent if their parent is at even depth, or whose label is strictly greater that that of their parent if their parent is at odd depth. Deleting a node deletes the entire subtree below it. The root of the entire tree is at depth 0. For example, given the tree on the left, your function should produce the tree on the right without creating any new Tree nodes. The function `prune` does not return a value.





```
def prune(t):
    def prune_level(t, d):
        if d % 2 == 0:
             t.branches = [ b for b in t.branches if b.label >= t.label]
        else:
             t.branches = [ b for b in t.branches if b.label <= t.label]
        for b in t.branches:
             prune_level(b, d+1)
    prune_level(t, 0)
```

5. (1 points) Sum of Human Knowledge

In what work do we find the sentence, "His series of judgements in F sharp minor, given *andante* in six-eight time, are among the most remarkable effects ever achieved in a Court of Chancery."?

Answer: *Iolanthe* by G. S. Gilbert and A. Sullivan

6. (6 points) Stream

What is printed by the following statements? The `Stream` class is defined on page 3.

```
from operator import add, mul

nums = make_integer_stream(1)
astream = \
  Stream(1,
    Stream(2,
      Stream(3,
        Stream(4,
          lambda:
            combine_streams(mul, bstream,
              combine_streams(add, bstream.rest, nums))))))
bstream = filter_stream(lambda x: x % 2 == 0, astream)
print(stream_to_list(bstream, 5))
```

Answer: `[2, 4, 10, 48, 510]`

7. (8 points) Raises

Every employee in a company is given a raise every year by a percentage specific to that employee. Assume that there is a table 'current' of employee names and salaries, and another table 'raises' listing employee names and the fraction by which their salaries increase each year:

current

name	salary
Ben Bitdiddle	60000
Alyssa P Hacker	40000
Cy D Fect	35000

raises

name	raise
Ben Bitdiddle	0.03
Alyssa P Hacker	0.02
Cy D Fect	0.01

For example, Ben Bitdiddle has a current salary of $60000, which is raised by 0.03 (3%) per year. From these tables, create two new tables 'next' and 'future' giving employee names and salaries next year and in K years, respectively. For example, given the tables above and taking $K = 2$, we should have

next

name	salary
Ben Bitdiddle	61800
Alyssa P Hacker	40800
Cy D Fect	35350

future

name	salary
Ben Bitdiddle	63654
Alyssa P Hacker	41616
Cy D Fect	35703.5

For the 'future' table, write 'K' wherever you need to put the number of years in the future (that is, your solution should work for any value of K.)

```
CREATE TABLE next AS
   SELECT current.name, salary * (1 + raise) FROM current, raises
       WHERE current.name = raises.name

CREATE TABLE future AS
   WITH temp (name, salary, year) AS (
       SELECT name, salary, 0 FROM current UNION
       SELECT temp.name, salary * (1+raise), year+1 FROM temp, raises
            WHERE temp.name = raises.name AND year < K
   )
SELECT name, salary FROM temp WHERE year = k;
```

8. (6 points) Pair Up

Fill in the Scheme `pairs` function so that `(pairs L)`, where `L` is a list, produces a list of lists, where each of these lists contains a pair of elements from `L`. The function must be tail-recursive. You need not define (or use) the `reverse` function.

```
scm> (pairs '(1 2 3 4))
((1 2) (3 4))
scm> (pairs '(1 2 3 4 5))  ; Odd element at end put in singleton list.
((1 2) (3 4) (5))
scm> (pairs '())
()
```

```
(define (reverse P)
  """Returns the reverse of list P. This function is tail-recursive"""
  ;;; Implementation not shown
)

(define (pairs L)
  (define (accum-pairs lst result)
    (cond ((null? lst) result)
          ((or (null? (cdr lst)) (null? (cdr (cdr lst))))
           ; The second clause of the or isn't really needed, but avoids
           ; one cons when the list has even length.
           (cons lst result))
          (else (accum-pairs (cdr (cdr lst))
                             (cons (list (car lst) (car (cdr lst)))
                                   result)))))
  (reverse (accum-pairs L '()))
)
```

9. (10 points) Don't Go Down!

For the purposes of this problem, define the *greedy nondescending subsequence* of a sequence $S = (s_0, s_1, \ldots, s_{n-1})$ to be the sequence $(s_{i_1}, s_{i_2}, \ldots, s_{i_m})$ where $0 = i_1 < i_2 < \cdots < i_m < n$, i_{k+1} is the smallest index such than $s_{i_k} \leq s_{i_{k+1}}$, and i_m is as large as possible (given the first two constraints).

(a) (5 pt) Fill in the function upsubseq such that upsubseq(L) returns an iterator that yields a greedy nondescending subsequence of L, where L is a Python list or tuple. Do not use a generator. Your iterator must use constant space; do not create a new list containing the subsequence and return a standard iterator for it. You need not use all the spaces provided.

```
class upseq_iter:
    def __init__(self, L):
        self._L = L
        self._k = 0

    def __iter__(self): return self

    def __next__(self):
        if self._k >= len(self._L):
            raise StopIteration
        r = self._L[self._k]
        self._k += 1
        while self._k < len(self._L) and self._L[self._k] < r:
            self._k += 1
        return r

def upsubseq(L):
    """
    >>> list(upsubseq([4, 2, 3, 6, 6, 5, 7, 1, 3]))
    [4, 6, 6, 7]
    >>> list(upsubseq([]))
    []
    """
    return upseq_iter(L)
```

(b) (5 pt) Fill in the function `genupseq` so that `genupseq(L)` returns a generator that yields a greedy nondescending subsequence of `L`, where `L` is a Python list or tuple. Your generator must use constant space; do not create a new list containing the subsequence and return a standard iterator for it. You need not use all the spaces provided.

```
def genupseq(L):
    """
    >>> list(genupseq([4, 2, 3, 6, 6, 5, 7, 1, 3]))
    [4, 6, 6, 7]
    >>> list(genupseq([]))
    []
    """

    k = 0

    while k < len(L):
        r = L[k]
        yield r
        k += 1
        while k < len(L) and L[k] < r:
            k += 1
```

10. (8 points) Sum Range

Fill in the function sumrange so that, given a binary search tree T with integer labels and two integer bounds low and high, it returns the sum of all values in T between low and high, inclusive. If the label of the root of T is within this range, your program must take time $\Theta(M)$, where M is the number of labels in T that are within the bounds low and high, regardless of the number of values in T. Assume that T is of type BinTree, as defined on page 2. You do not need to use all lines.

```
from defns import BinTree

def sumrange(T, low, high):
    """The sum of all values in BST T that are between LOW and HIGH,
    inclusive.
    >>> S = BinTree(50, BinTree(25, BinTree(10, BinTree(5), BinTree(20)),
    ...                             BinTree(30, BinTree(28), BinTree(40))),
    ...                 BinTree(75, BinTree(60, BinTree(55), BinTree(65)),
    ...                             BinTree(90, BinTree(80), BinTree(100))))
    >>> sumrange(S, 20, 56)
    248
    """
    if T is BinTree.empty:
        return 0
    elif T.label >= low and T.label <= high:
        return T.label + sumrange(T.left, low, high) \
                 + sumrange(T.right, low, high)
    elif T.label < low:
        return sumrange(T.right, low, high)
    else:
        return sumrange(T.left, low, high)
```

11. (10 points) Big Theta

(a) (2 pt) Circle all of the following that are true.

i. $\Theta(3N) = \Theta(N/3)$

ii. $\Theta(2^N) = \Theta(2^N + N^2)$

iii. $\Theta(\lg(2^N)) = \Theta(\lg(2^N \cdot N^2))$

iv. $\Theta(2^N) = \Theta(2^N \cdot N^2)$

v. $\Theta(f(N) + 1/N) = \Theta(f(N))$ if $f(N) > K$ for some fixed $K > 0$.

(b) (2 pt) Using N as the length of the Python list `L`, give a worst-case bound for the execution time of the following code, assuming that execution time is proportional to the number of times the commented statement is executed.

```
s = 0
for k in range(len(L)):
    j = 1
    while j < k:
        s += L[j]    # Count this
        j *= 3
```

Answer: $\Theta(N \cdot \lg N)$

(c) (2 pt) If `T` is a `Tree` containing N nodes in all, what is a worst-case bound on the time required to determine whether a value `x` is a label of the tree, measuring the number of comparisons to `x` that are required? Do not assume that `T` is a search tree.

Answer: $\Theta(N)$

(d) (2 pt) If `T` is a binary search tree containing N nodes in all, what is a worst-case bound on the time required to determine whether a value `x` is a label of the tree, measuring the number of comparisons to `x` that are required?

Answer: $\Theta(N)$

(e) (2 pt) We've seen how to add a `__getitem__` method to the `Link` class so that one can index a linked list just as one can a Python list. That is, `L[k]`, for `L` a `Link` (see page 2), will traverse `L` to item k and return it. Given such an implementation, what is the worst-case running time of the following program, as a function of N the length of linked list `L`, where we are counting the number of times a `.rest` field is accessed? Assume `N` is set to the length of the list.

```
c = 0
for k in range(N):
    for j in range(k-1):
        if L[k] == L[j]:
            c += 1
```

Answer: $\Theta(N^3)$

Extra space if needed.

Extra space if needed.