CS 61A Fall 2024

Structure and Interpretation of Computer Programs

MIDTERM 2

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

(a) What is your full name?

(b) What is your student ID number?

(c) What is your @berkeley.edu email address?

(d) Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

1. (5.0 points) What Would Python Display?

Assume the following code has been executed. The `Tree` class appears on the midterm 2 study guide (page 2, left side).

```
scare = Tree(0, [Tree(4), Tree(5, [Tree(10)]), Tree(2)])
crow = Tree(9, [scare, Tree(7, [Tree(6)])])

def climb(t, f):
    if t.is_leaf():
        return [t.label]
    return [t.label] + climb(max(t.branches, key=f), f)

def run(t):
    if t.is_leaf():
        return t.label
    else:
        return max(map(run, t.branches))

def hide(t):
    return t.label

jump = {(2 * x - 1): x for x in range(50)}
```

Write the value of each expression below or *Error* if an error occurs.

(a) (1.0 pt) `crow.branches[0].label`

(b) (1.0 pt) `[hide(b) for b in scare.branches]`

(c) (1.0 pt) `climb(crow, hide)`

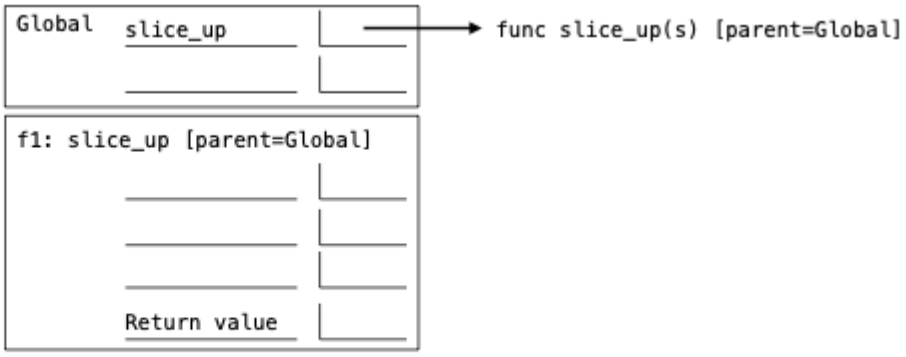
(d) (1.0 pt) `climb(crow, run)`

(e) (1.0 pt) `jump[jump[13]]`

2. (5.0 points) Pizza by the Slice

Answer the questions about the code below. You are encouraged to draw an environment diagram, but the diagram itself will not be scored.

```
 1: def slice_up(s):
 2:     copy = s
 3:     t = [copy]
 4:     while len(s) > 1:
 5:         t.append(s[:len(s)-1])
 6:         s.pop()
 7:         s[0] = len(s)
 8:     return t
 9:
10: t = [3, 4, 5]
11: print(slice_up(t))
12: print(t)
```



(a) (3.0 pt) What would be printed by the expression `print(slice_up(t))` on line 11?

(b) (1.0 pt) What would be printed by the expression `print(t)` on line 12?

- ◯ `[1]`
- ◯ `[2]`
- ◯ `[3]`
- ◯ `[1, 4]`
- ◯ `[2, 4]`
- ◯ `[3, 4]`
- ◯ `[1, 4, 5]`
- ◯ `[2, 4, 5]`
- ◯ `[3, 4, 5]`

(c) (1.0 pt) What is the order of growth of the time it takes to evaluate `list(list(q))` for a list of numbers `q` in terms of the length of `q`. Assume that creating a list with one element takes constant time, and making a copy of a list takes linear time in the length of the list.

- ◯ exponential
- ◯ quadratic
- ◯ linear
- ◯ logarithmic
- ◯ constant

3. (6.0 points) CS 61A Software Store

A `Store` instance has a list of `branches`, and each branch is also a `Store` instance, forming a tree of stores. Each `Store` instance also has a list of programs (strings) called its `inventory`. For a `Store`:

- The `copies(s)` method returns the number of times `s` appears in the inventories of all nodes in its tree of stores.
- The `add_to(k)` method returns a function that takes a string and appends it to the `inventory` of its `k`th branch. Assume that `k` is non-negative and less than the number of branches.

```
class Store:
    """A Store selling programs has branches forming a tree of Stores.
    >>> north, south = Store(['Cats', 'Ants']), Store(['Cats', 'Cats', 'Hog'])
    >>> east, west = Store(['Cats', 'Hog', 'Hog'], [north, south]), Store(['Cats', 'Cats', 'Ants'])
    >>> main = Store(['Ants', 'Ants', 'Hog'], [east, west])
    >>> east.copies('Cats')  # 1 in north, 2 in south, 1 in east
    4
    >>> main.copies('Ants')  # 2 in main, 1 in north, 1 in west
    4
    >>> main.add_to(1)('Ants') # Add 'Ants' to the inventory of west, the branch of main at index 1
    >>> [main.copies('Ants'), west.copies('Ants')]  # increased copies in both main and west
    [5, 2]
    """
    def __init__(self, programs, branches=[]):
        assert all([isinstance(b, Store) for b in branches])
        self.branches = branches
        self.inventory = programs

    def copies(self, s):
        """Return the number of times s (string) appears in all inventories of this tree."""
        return sum([_______ for p in self.inventory if p == s] + [_______ for b in self.branches])
                      (a)                                                    (b)

    def add_to(self, k):
        """Return a function that appends a string to the inventory of the branch at index k."""
        return _______
                 (c)
```

(a) (1.0 pt) Fill in blank (a).

- ◯ `1`
- ◯ `p`
- ◯ `len(p)`
- ◯ `len(self.inventory)`

(b) (2.0 pt) Fill in blank (b).

(c) (3.0 pt) Fill in blank (c).

4. (18.0 points) Almost a Perfect Question

Definition. A *proper divisor* of an integer `n` is an integer `d` less than `n` that evenly divides `n`, and so `n` divided by `d` has no remainder. A *semiperfect* number is a positive integer that is the sum of some (or all) of its proper divisors. For example, $24 = 2 + 4 + 6 + 12$ and so it is semiperfect. The sum of divisors cannot contain repeated numbers.

(a) (6.0 points)

Implement `semiperfect`, which takes a positive integer `n`. It returns `True` if `n` is semiperfect and `False` otherwise.

```
def semiperfect(n):
    """Return whether positive integer n is a sum of some (or all) of its proper divisors.
    >>> [k for k in range(1, 40) if semiperfect(k)]
    [6, 12, 18, 20, 24, 28, 30, 36]
    """
    def f(s, d):
        if _______:
             (a)
            return True
        if d >= n:
            return False
        if _______ and f(_______, d + 1):
             (b)           (c)
            return True
        return _______
                  (d)
    return f(n, 1)
```

i. (1.0 pt) Fill in blank (a).

- ◯ `n == 0`
- ◯ `s == 0`
- ◯ `s == d`
- ◯ `n == d`

ii. (1.0 pt) Fill in blank (b).

- ◯ `n % d == 0`
- ◯ `s % d == 0`
- ◯ `n % d > 0`
- ◯ `s % d > 0`

iii. (2.0 pt) Fill in blank (c).

iv. (2.0 pt) Fill in blank (d).

(b) (6.0 points)

Implement `subsums`, a generator function that takes a list of **unique** (no repeats) positive integers `s` and a positive integer `n`. It yields all sublists of `s` that sum to `n`. A sublist of `s` is a list containing some of the elements of `s` in order. The `subsums` function should not yield the same list twice. The lists may appear in any order.

```
def subsums(s, n):
    """Yield all sublists of s that sum to n.
    >>> list(subsums([1, 2, 3, 6, 9], 18))
    [[1, 2, 6, 9], [3, 6, 9]]
    >>> list(subsums([1, 2, 3, 4, 6], 16))
    [[1, 2, 3, 4, 6]]
    >>> list(subsums([1, 2, 3, 4, 6, 8, 12], 12))
    [[1, 2, 3, 6], [1, 3, 8], [2, 4, 6], [4, 8], [12]]
    """
    if s:
        if _______:
             (e)
            yield [n]
        for t in _______:
                    (f)
            yield _______
                     (g)
        yield from _______
                     (h)
```

i. (1.0 pt) Fill in blank (e).

◯ `n == s`

◯ `n == s[0]`

◯ `n in s`

◯ `[n] == s`

ii. (3.0 pt) Fill in blank (f).

iii. (1.0 pt) Fill in blank (g).

◯ `s + t`

◯ `[s] + t`

◯ `s[1:] + t`

◯ `s[:1] + t`

iv. (1.0 pt) Fill in blank (h).

- ◯ `subsums(s, n)`
- ◯ `subsums(s[1:], n)`
- ◯ `subsums(s, n - 1)`
- ◯ `subsums(s[1:], n - 1)`

(c) (6.0 points)

Implement `semisums`, which takes a positive integer `n`. It returns a list of all lists of unique (no repeats) proper divisors of `n` that sum to `n`. You may call `semiperfect` and `subsums`.

```
def semisums(n):
    """Return a list of all lists (with no repeats) of
    proper divisors of n that sum to n.

    >>> semisums(22)  # 22 is not semiperfect, so there are no sums.
    []
    >>> semisums(30)  # 30 is semiperfect. It has three different sums.
    [[1, 3, 5, 6, 15], [2, 3, 10, 15], [5, 10, 15]]
    """
    return list( _______ ([k for k in range(1, n) if _______ ], _______ ))
                  (i)                                    (j)        (k)
```

i. (1.0 pt) Fill in blank (i).

- ◯ `map`
- ◯ `filter`
- ◯ `sorted`
- ◯ `subsums`
- ◯ `semiperfect`

ii. (1.0 pt) Fill in blank (j).

- ◯ `semiperfect(n)`
- ◯ `semiperfect(k)`
- ◯ `semiperfect(n) or semiperfect(k)`
- ◯ `n % k == 0`

iii. (1.0 pt) Fill in blank (k).

iv. (3.0 pt) Fill in blank (l) of `primitive_semiperfect`, a function that takes a positive integer `n`. It returns `True` if `n` is semiperfect and no smaller semiperfect number is a divisor of `n`. Your answer should have two expressions separated by a comma.

```
def primitive_semiperfect(n):
    """Return whether n is semiperfect and has no semiperfect proper divisors.

    >>> [k for k in range(1, 300) if primitive_semiperfect(k)]
    [6, 20, 28, 88, 104, 272]
    """
    return semiperfect(n) and not any(map(semiperfect, filter(_______)))
                                                                 (l)
```

5. (11.0 points) How Long is this Exam?

(a) (5.0 points)

Implement `longer`, which takes two linked lists of numbers `s` and `t`. It returns the longer of the two. If they have the same length, it returns `s`.

A linked list is either a `Link` instance or `Link.empty`. The `Link` class is on page 2 of the midterm 2 study guide.

```
def longer(s, t):
    """Return the longer linked list, s or t. (Same length? return s.)

    >>> longer(Link(2, Link(3)), Link.empty)
    Link(2, Link(3))
    >>> longer(Link(2, Link(3)), Link(4, Link(5)))
    Link(2, Link(3))
    >>> longer(Link(2, Link(3)), Link(4, Link(5, Link(6, Link(7)))))
    Link(4, Link(5, Link(6, Link(7))))
    >>> longer(Link.empty, Link.empty) is Link.empty
    True
    """
    a, b = s, t

    while b is not Link.empty:

        if _______:
             (a)

            return _______
                      (b)


        _______
          (c)

    return _______
             (d)
```

i. (2.0 pt) Fill in blank (a).

ii. (1.0 pt) Fill in blank (b).

- ◯ `a`
- ◯ `b`
- ◯ `s`
- ◯ `t`
- ◯ `longer(a, b)`
- ◯ `longer(s, t)`

iii. (1.0 pt) Fill in blank (c).

- ◯ `a = a.rest`
- ◯ `a = s.rest`
- ◯ `b = b.rest`
- ◯ `b = t.rest`
- ◯ `a, b = a.rest, b.rest`
- ◯ `a, b = s.rest, t.rest`

iv. (1.0 pt) Fill in blank (d).

- ◯ `a`
- ◯ `b`
- ◯ `s`
- ◯ `t`
- ◯ `longer(a, b)`
- ◯ `longer(s, t)`

(b) (6.0 points)

Implement `longest`, which takes a linked list of postive integers `s` and a positive integer `n`. It returns the longest sublist of `s` with elements that sum to a number less than or equal to `n`. **Do not mutate `s`.** In case of a tie, return any of the longest sublists whose sum is `n` or less. Assume `longer` is implemented correctly.

A sublist of a linked list `s` is a linked list with some (or none or all) of the elements of `s` in order.

Assume the sum of the elements of `Link.empty` is 0. `Link.empty` is a sublist of any linked list.

```
def longest(s, n):
    """Return the longest sublist of s that sums to n or less.

    >>> longest(Link(5, Link(1, Link(3, Link(4, Link(2, Link(7)))))), 7)
    Link(1, Link(3, Link(2)))
    >>> longest(Link(5, Link(1, Link(3, Link(4, Link(2, Link(7)))))), 70)
    Link(5, Link(1, Link(3, Link(4, Link(2, Link(7))))))
    >>> longest(Link(3, Link(4, Link(5))), 2) is Link.empty
    True
    """
    if s is Link.empty:
        return s

    t = _______
          (e)

    if _______:
         (f)

        return longer( _______ , t)
                            (g)
    else:
        return t
```

i. (1.0 pt) Fill in blank (e).

◯ `longest(s, n)`

◯ `longest(s, n - s.first)`

◯ `longest(s.rest, n)`

◯ `longest(s.rest, n - s.first)`

ii. (2.0 pt) Fill in blank (f).

iii. (3.0 pt) Fill in blank (g).

6. (0.0 points) Nice Path!

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

(a) (0.0 pt) Fill in the blank of `max_path`, which takes a Tree of integers `t` and a function `g`. It returns a list `s` containing the labels along the path from the root of `t` to a leaf for which `g(s)` is as large or larger than for any other path. In case of a tie, return any path with a maximum `g(s)` value.

```
def climb(t, f):
    if t.is_leaf():
        return [t.label]
    return [t.label] + climb(max(t.branches, key=f), f)

def max_path(t, g):
    """Return the path s from the root of t to a leaf for which g(s) is largest.

    >>> scare = Tree(0, [Tree(4), Tree(5, [Tree(10)]), Tree(2)])
    >>> crow = Tree(4, [Tree(5), Tree(9, [scare, Tree(7, [Tree(6)])]), Tree(8)])
    >>> max_path(crow, lambda p: -p[-1])            # The path to the smallest leaf
    [4, 9, 0, 2]
    >>> max_path(crow, len)                         # The longest path
    [4, 9, 0, 5, 10]
    >>> max_path(crow, lambda p: -abs(p[0]-p[-1]))  # To the leaf closest in value to the root
    [4, 9, 0, 4]
    """

    x = [t.label]  # You can use x instead of [t.label] to shorten your answer!

    return climb(t, lambda b: _______ )
```

No more questions.