CS 331 Spring 2018

Midterm Exam

Instructions:

- This exam is closed-book, closed-notes. Computers of any kind are not permitted.
- For numbered, multiple-choice questions, fill your answer in the corresponding row on the "bubble" sheet.
- For problems that require a written solution (labeled with the prefix "WP"), write your answer in the space provided on the written solution sheet. Please write legibly and clearly indicate your final answer.
- Turn in the exam question packet, bubble sheet, and written solution sheet separately.

Basic Concepts (24 points):

1. What are the contents of the list `lst` after the following code is executed?

```
lst = [2*x+y for x in range(3) for y in range(3,6)]
```

   (a) `[3, 4, 5, 6, 7, 8, 9]`
   (b) `[3, 4, 5, 5, 6, 7, 7, 8, 9]`
   (c) `[3, 5, 7, 4, 6, 8, 5, 7, 9]`
   (d) `[6, 7, 8, 8, 9, 10, 10, 11, 12]`

2. What are the contents of the dictionary `dct` after the following code is executed?

```
dct = {}
for x in range(1,21):
    dct[x//4] = x
```

   (a) `{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5}`
   (b) `{0: 1, 1: 4, 2: 8, 3: 12, 4: 16, 5: 20}`
   (c) `{0: 1, 1: 5, 2: 10, 3: 15, 4: 20, 5: 25}`
   (d) `{0: 3, 1: 7, 2: 11, 3: 15, 4: 19, 5: 20}`

3. What are the contents of the list `lst` after the following code is executed?

```
def gen(m, n):
    while m < n:
        yield m
        m *= 2
    yield n

lst = []
g1 = gen(2, 5)
g2 = gen(4, 20)
while True:
    try:
        lst.append(next(g1))
        lst.append(next(g2))
    except StopIteration:
        break
```

   (a) `[2, 4, 4, 8, 5, 16]`
   (b) `[2, 4, 8, 16, 32, 40]`
   (c) `[4, 8, 16, 20, 2, 4, 5]`
   (d) `[2, 4, 4, 8, 5, 16, None, 20]`

4. What is the worst-case runtime complexity of locating and returning the *largest* element in an unsorted array-backed list of N elements?

   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

5. What is the worst-case runtime complexity of inserting an element into an array-backed list of N elements?

   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

6. What is the worst-case runtime complexity of using binary search to determine whether a given value exists in a sorted array-backed list of N elements?

   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

7. Which of the following scenarios will consistently cause binary search (given search value x and list `lst`) to exhibit the poorest runtime performance?

   (a) `lst` contains duplicates of x
   (b) x is the middle element of `lst`
   (c) x is either the smallest or largest value in `lst`
   (d) x occurs in the lower half of `lst`

8. What is the maximum number of elements a properly implemented binary search will need to compare a value against in order to determine its position in a sorted list of 1,000 elements?

   (a) 5
   (b) 10
   (c) 15
   (d) 20

9. Which of the following relations is true?

   (a) $3n^2 = O(n)$
   (b) $64n + 1024 = O(n^2)$
   (c) $2n^3 + 4n - 10 = O(n^2)$
   (d) $10^n - n^2 = O(n^2)$

10. Which of the following datatypes in Python is *not* immutable?

   (a) tuple
   (b) string
   (c) range
   (d) list

11. Which of the following operations on some built-in Python list `lst` has $O(1)$ runtime complexity (assume that `x` refers to a value and `i` to a valid index)?

   (a) `lst[i] = x`
   (b) `lst.insert(i, x)`
   (c) `x in lst`
   (d) `del lst[i]`

12. Which of the following is *not true* of all correctly implemented **iterable** objects?

   (a) they can be used as the target of a `for` loop
   (b) they can be passed to `iter` to obtain an iterator
   (c) they can be indexed, and their lengths can be retrieved with the `len` function
   (d) the object obtained by calling `iter` on them can be passed to `next` to get an item or raise a `StopIteration` exception

Estimating Big-O (9 points):

For each of the following functions, determine the corresponding worst-case runtime complexity in terms of N. Assume that any `lst` arguments are Python lists.

13.
```
def fA(lst):
    N = len(lst)
    accum = 0
    for i in range(1, N, N//10):
        accum += lst[i]
        print('doing')
    return accum
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

14.
```
def fB(N):
    accum = 0
    for i in range(N):
        for j in range(N, i, -1):
            accum += i+j
    return accum
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

15.
```
def fC(lst):
    N = len(lst)
    accum = 0
    if N < 1000:
        for i in range(N * 10):
            accum += i
        return accum
    else:
        return 0
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

Lists and Dicts (6 points):

WP1 Implement `merge_dicts`, which accepts zero or more input dictionaries, and returns a new dictionary containing keys found in all the input dictionaries. When a key is found in more than one of the input dictionaries, the corresponding values are combined into a list.

E.g., `merge_dicts({'a': 'apple', 'b': 'banana'}, {'a': 'ant', 'c': 'cat'})` returns the merged dictionary `{'a': ['apple', 'ant'], 'b': 'banana', 'c': 'cat'}`.

E.g., `merge_dicts({1: 2, 2: 4, 3: 6}, {2: 8, 3: 12}, {3: 15, 4: 20})` returns the merged dictionary `{1: 2, 2: [4, 8], 3: [6, 12, 15], 4: 20}`.

You may assume that none of the input dictionaries contain lists as values.

Mystery Sort (8 points):

Consider the following mystery sort function:

```
def mystery_sort(lst):
    print(lst) # display list
    for i in range(0, len(lst)-1):
        to_swap = i
        for j in range(i+1, len(lst)):
            if lst[j] < lst[to_swap]:
                to_swap = j
        lst[to_swap], lst[i] = lst[i], lst[to_swap]
        print(lst) # display list
```

WP2 (a) Show the list contents, in order, displayed by all calls to `print` when `mystery_sort` is called with the input list `[5, 3, 6, 2, 8, 1, 4, 7]`. (3 points)

WP2 (b) What is the Big-O runtime complexity of `mystery_sort`, when called with an input list of length N? (2 points)

WP2 (c) What characteristic(s) of an input list of arbitrary size will cause `mystery_sort` to perform *worse* than insertion sort? Briefly explain why. (3 points)

Array-backed List (8 points):

Your `ArrayList` lab implementation failed to account for list *slice* arguments — e.g., "`lst[1:5]`". Working with slices is easy, as Python automatically creates `slice` objects and passes them to the `__getitem__`, `__setitem__`, and `__delitem__` methods. `slice` objects have `start` and `stop` attributes that give us the values specified in the array bracket notation.

Below is an implementation of `__getitem__` that accepts both regular integer indexes and slices:

```
class ArrayList:
    def __init__(self):
        self.data = []

    def __getitem__(self, idx):
        if not isinstance(idx, slice):
            return self.data[idx] # deal with integer indices
        else:
            return [self[i] for i in range(idx.start, idx.stop)]
```

Now, with an `ArrayList` named `l` whose `data` attribute refers to the list `[1, 2, 3, 4, 5]`, the expression `l[1:4]` evaluates to `[2, 3, 4]`.

WP3 Complete the implementations of `__setitem__` and `__delitem__` so as to accept slices. You may assume all slices contain only valid, positive index values, and do not include a step (i.e., you need not support slices of the form `l[start:stop:step]`).

As in the lab, you should treat the underlying Python list (in `data`) as an array. You may assume that the `ArrayList` class has working `append`, `insert`, and `__getitem__` methods (not shown).

The following code and accompanying output demonstrates how slices may be used with `__setitem__` and `__delitem__`:

```
l = ArrayList()
for x in range(10):
    l.append(x)
print('1:', l)
l[1:9] = ('a', 'b', 'c')
print('2:', l)
del l[2:4]
print('3:', l)

# Output
#
# 1: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# 2: [0, 'a', 'b', 'c', 9]
# 3: [0, 'a', 9]
```