CS 331 Fall 2019
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.
- Write and bubble in your student number, without the leading 'A', 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 does the following list comprehension evaluate to?

   ```
   [x+str(y) for x in 'abc' for y in range(5,20,5)]
   ```

   (a) `['a5', 'a10', 'a15', 'b5', 'b10', 'b15', 'c5', 'c10', 'c15']`

   (b) `['abc5', 'abc10', 'abc15']`

   (c) `['a5', 'b5', 'c5', 'a10', 'b10', 'c10', 'a15', 'b15', 'c15']`

   (d) `['abc', '5', 'abc', '10', 'abc', '15']`

2. Consider the following code:

   ```
   l = 'it was the best of times it was the worst of times'.split()
   d = {x[0]: [] for x in l}
   for y in l:
       d[y[0]].append(y)
   ```

   What does `d['w']` evaluate to, after executing the above?"

   (a) `[]`

   (b) `['worst']`

   (c) `['was', 'was', 'worst']`

   (d) `['the', 'the', 'of']`

3. Which of the following method calls is equivalent to the statement "`d[x] = y`"?

   (a) `d.__getitem__(x) = y`

   (b) `d.__setitem__(x, y)`

   (c) `d.__delitem__(x) ; d.__setitem__(y)`

   (d) `d.replace(x, y)`

4. Which of the following types in Python is immutable?

   (a) `range`

   (b) `list`

   (c) `set`

   (d) `dict`

5. In which situation would it make sense to use a generator expression instead of a list comprehension to define a sequence?
   (a) when the sequence must be iterated over multiple times
   (b) when it is necessary to access elements in the sequence in an arbitrary order (by index)
   (c) when the sequence must be iterated over backwards
   (d) when we plan to iterate over the sequence just once

6. Which of the following operations is unsupported on some non-empty set `s`?
   (a) `list(s)`
   (b) `iter(s)`
   (c) `'a' in s`
   (d) `s[0]`

7. 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 500,000 elements?
   (a) 14
   (b) 19
   (c) 24
   (d) 30

8. What property of an input list most notably improves the runtime complexity of insertion sort when run on it?
   (a) the input list is already close to being sorted
   (b) the input list is reverse sorted
   (c) the input list is array-backed
   (d) the input list contains duplicate elements

9. Which of the following relations is true?
   (a) $2^n - 1000 = O(\log n)$
   (b) $3n^2 + \log n + 1000 = O(n^2)$
   (c) $n + 5 = O(1)$
   (d) $8n^2 + 10n = O(n)$

10. What is the runtime complexity for deleting an arbitrary element (by index) in an array-backed list of N elements?

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

11. What is the runtime complexity for locating and returning the largest element (by value) in an array-backed list of N elements, assuming the values in the list are in no particular order?

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

12. Which of the following scenarios makes the most efficient use (in terms of runtime complexity) of the array-backed list data structure?

    (a) alternating appending elements to the list and sorting the list
    (b) inserting elements at the beginning of the list
    (c) removing elements from the middle of the list
    (d) appending elements and removing elements from the end of the list

Estimating Big-O (9 points):

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

13.
```
def fA(lst):
    for _ in range(len(lst)):
        x = lst[0]
        del lst[0]
        lst.append(x)
```

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

14.
```
def fB(lst):
    x = lst[0]
    cs = [1]
    for j in range(1, len(lst)):
        if x == lst[j]:
            cs[-1] += 1
        else:
            x = lst[j]
            cs.append(1)
```

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

15.
```
def fC(lst):
    r = 0
    n = 100
    if len(lst) < n:
        n = len(lst)
    for x in range(n):
        r += x
```

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

Python data structures (8 points):

WP1 Implement `zip`, which accepts zero or more sequences in the star parameter `seqs`, and returns a list of tuples where the elements of each tuple are drawn from each input sequence. The number of tuples in the returned list is determined by the length of the shortest input sequence.

Some sample calls/results:

- `zip([1, 2, 3], [10, 20, 30])` ⇒ `[(1,10), (2,20), (3,30)]`
- `zip('hello', [1,2,3,4], 'abracadabra')`
  ⇒ `[('h',1,'a'), ('e',2,'b'), ('l',3,'r'), ('l',4,'a')]`.
- `zip(range(100,0,-1), range(0,25,5), range(3), 'abcdefg')`
  ⇒ `[(100,0,0,'a'), (99,5,1,'b'), (98,10,2,'c')]`.

Mystery sort (8 points):

Consider the following mystery sort function:

```
def mystery_sort(lst):
    for i in range(len(lst)-1):
        k = i
        for j in range(i, len(lst)):
            if lst[j] < lst[k]:
                k = j
        lst[i], lst[k] = lst[k], lst[i]
        print(lst)
```

WP2 (a) Show the list contents, in order, displayed by all calls to `print(lst)` when `mystery_sort` is called with the input list `[3, 2, 4, 6, 0, 7, 5, 1]`. (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) If you were given the choice to use this sort function or insertion sort, which one would you choose for optimal runtime performance across all types of input lists? Explain. (3 points)

Array-backed list (8 points):

WP3 Implement the method `move` for the array-backed list. When called with the argument indices `idx_src` and `idx_dst`, the element at `idx_src` will be relocated to `idx_dst`, while the intervening elements will be shifted up or down (towards `idx_src`) as necessary. You may assume both `idx_src` and `idx_dst` are valid indices ≥ 0.

Your implementation may only use the following methods on the built-in list:

- `len(self.data)`
- Accessing a valid, positive index (e.g., `self.data[i]`)

Your implementation should not use any other `ArrayList` methods.

Some sample calls/results:

- Calling `move(1, 3)` on the list `['lions','tigers','bears','oh','my']` should result in the list `['lions','bears','oh','tigers','my']`
- Calling `move(8, 4)` on the list `[0,1,2,3,4,5,6,7,8,9]` should result in the list `[0,1,2,3,8,4,5,6,7,9]`