CS 331 Summer 2017

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 (20 points):

1. What is the worst-case runtime complexity of locating and returning the maximum value in an unsorted array-backed list?

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

2. What is the worst-case runtime complexity of locating and returning the second-to-last element (by position) in an unsorted array-backed list?

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

3. What is the worst-case runtime complexity of deleting an element from the middle of an array-backed list?

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

4. To reverse the order of elements in a list, we can simply remove the elements one-by-one from the end of the list while appending them to a new list — at the end of the operation the new list will have the desired ordering. What is the worst-case runtime complexity of this algorithm?

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

5. In order to maintain an array-backed list in sorted order when adding an element, we can first use binary search to locate the position at which the new element should be inserted, after which the `insert` method would be used to splice the element in. What is the worst-case runtime complexity of this algorithm?

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

6. Which of the following scenarios will consistently cause insertion sort to exhibit the poorest runtime complexity?

   (a) the input list is sorted
   (b) the input list is reverse sorted *
   (c) the input list is randomized (shuffled)
   (d) all values in the input list are identical

7. What is the maximum number of elements a properly implemented binary search will need to compare a value against in order to determine if it exists in a sorted list of 10,000 elements?

   (a) 5
   (b) 8
   (c) 13 *
   (d) 20

8. Which of the following keywords, when present in a function, causes that function to return a *generator object* when called?

   (a) `iter`
   (b) `next`
   (c) `yield` *
   (d) `raise`

9. Which of the following Python list methods has a worst-case runtime complexity of O(1)?

   (a) `insert`
   (b) `__iter__` *
   (c) `__delitem__`
   (d) `__contains__`

10. Which of the following methods is *not* permitted on an *immutable* sequence type?

   (a) `+` (`__add__`)
   (b) `count`
   (c) `index`
   (d) `pop` *

Estimating Big-O (9 points):

For each of the following functions, determine the corresponding worst-case runtime complexity when called with an input list of size N. Assume the input list is a Python (array-backed) list.

11.
```
def fA(lst):
    acc = 0
    for x in lst:
        if x >= 100:
            acc += x
    return acc
```

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

12.
```
def fB(lst):
    acc = 0
    for i in range(len(lst)):
        for j in range(len(lst)):
            if lst[i] == lst[j] and i != j:
                acc += 1
                break
    return acc
```

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

13.
```
def fC(lst):
    n = len(lst)
    for i in range(100):
        a = random.randrange(n) # assume O(1)
        b = random.randrange(n) # assume O(1)
        lst[a], lst[b] = lst[b], lst[a]
    return lst
```

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

Python List/Dictionary Usage (14 points):

For each of the following questions, choose the option that corresponds to the value assigned to the variable `result` at the end of the given code listing.

14. `result = [2*x+1 for x in range(1, 8)]`

    (a) `[3, 7, 11, 15]`

    (b) `[2, 4, 6, 8, 10, 12, 14]`

    (c) `[3, 5, 7, 9, 11, 13, 15]` *

    (d) `[3, 7, 15, 31, 63, 127]`

15. `result = [(x,y) for x in range(3) for y in 'hi']`

    (a) `[(0, 'hi'), (1, 'hi'), (2, 'hi')`

    (b) `[(0, 'h'), (0, 'i'), (1, 'h'), (1, 'i'), (2, 'h'), (2, 'i')]` *

    (c) `[(0, 'h'), (1, 'h'), (2, 'h'), (0, 'i'), (1, 'i'), (2, 'i')]`

    (d) `[(0, 'h'), (1, 'i'), (2, 'h'), (0, 'i'), (1, 'h'), (2, 'i')]`

16.
    ```
    result = []
    for s in 'hello how are you'.split():
        result.insert(0, s)
    ```

    (a) `'youarehowhello'`

    (b) `['hello', 'how', 'are', 'you']`

    (c) `['you', 'are', 'how', 'hello']` *

    (d) `['hello', 'hello', 'hello', 'hello']`

17. `result = {x+y: (x, y) for x in range(1, 3) for y in range(10, 30, 10)}`

    (a) `{11: 1, 12: 2, 21: 1, 22: 2}`

    (b) `{11: 10, 12: 20, 21: 20, 22: 40}`

    (c) `{11: (1, 10), 12: (2, 10), 13: (3, 10)}`

    (d) `{11: (1, 10), 12: (2, 10), 21: (1, 20), 22: (2, 20)}` *

18.
```
result = {}
for i in range(5):
    for j in range(10):
        result[i] = j
```

(a) `{0: 0, 1: 1, 2: 2, 3: 3, 4: 4}`

(b) `{0: 1, 1: 2, 2: 3, 3: 4, 4: 5}`

(c) `{0: 9, 1: 9, 2: 9, 3: 9, 4: 9}` *

(d) `{0: 0, 1: 10, 2: 20, 3: 30, 4: 40}`

19.
```
result = {}
for s in 'a man a plan a canal plan'.split():
    if s not in result:
        result[s] = 0
    result[s] += len(s)
```

(a) `{'a': 3, 'plan': 2`

(b) `{'a': 3, 'canal': 1, 'man': 1, 'plan': 2}`

(c) `{'a': 3, 'canal': 5, 'man': 3, 'plan': 8}` *

(d) `{'a': 'aaa', 'canal': 'canal', 'man': 'man', 'plan': 'planplan'}`

20.
```
result = {}
for x in range(1, 5):
    result[x] = 0
    for y in range(1, 21):
        if y % x == 0:      # note: A % B = remainder of A / B
            result[x] += 1
```

(a) `{1: 10, 2: 5, 3: 3, 4: 2}`

(b) `{1: 20, 2: 10, 3: 6, 4: 5}` *

(c) `{1: 210, 2: 110, 3: 63, 4: 60}`

(d) `{1: 20, 2: 20, 3: 18, 4: 20}`

Binary Search (6 points):

WP1 Complete the implementation of `bin_search_descending`, which takes a list of unique values sorted in descending order (i.e., largest value to smallest) and a value to search for, and returns either the index of the value or `None`, if the value isn't in the list.

Insertion Sort (6 points):

Consider the following insertion sort implementation with a line added to print the contents of the list at the start of each inner iteration:

```
def insertion_sort(lst):
    for i in range(1, len(lst)):
        for j in range(i, 0, -1):
            print(lst) # print list contents
            if lst[j-1] > lst[j]:
                lst[j-1], lst[j] = lst[j], lst[j-1]
            else:
                break
```

WP2 Show the list contents, in order, displayed by all calls to `print` when `insertion_sort` is called with the input list `[4, 3, 1, 0, 2]`. The first output is already filled in for you; you may not need all lines.

Array-backed List (6 points):

WP3 Complete the implementation of the array-backed list method `move_to_front`, which, when provided with the index of an existing element, will remove that element from its current position and insert it at the front of the list (at index 0).

Your implementation should not make use of any list operations besides subscript-based access (which use `__getitem__` and `__setitem__`) and `len`.