CS 331 Midterm Exam 1

Wednesday, March 9th, 2016

Please bubble your answers in on the provided answer sheet. Also be sure to write and bubble in your student ID number (without the leading 'A').

1. What is the output of the following code snippet?

```
def f_1(x, y, z=10):
    return x+y+z

print(f_1(1, 2, 3))
```

(a) 6

(b) 13

(c) 15

(d) 30

2. What is the output of the following code snippet?

```
class C2:
   var = 100

    def __init__(self, val):
        self.var = val

c2inst = C2(200)
print(C2.var, c2inst.var)
```

(a) 100 100

(b) 200 200

(c) 100 200

(d) 200 100

3. What is the output of the following code snippet?

```
class C3:
    def __init__(self, val):
        self.val = val

    def m(x, y):
        x.val = y.val * 2

c3inst1 = C3(10)
c3inst2 = C3(20)
c3inst1.m(c3inst2)
print(c3inst1.val, c3inst2.val)
```

(a) 20 10

(b) 20 20

(c) 20 40

(d) 40 20

4. What is the output of the following code snippet?

```
def f4(fn, lst, init):
    res = init
    for i in range(len(lst)):
        res = fn(res, lst[i])
    return res

print(f4(lambda a,b: a + b**2, [1, 2, 2, 1], 0))
```

(a) 6

(b) 10

(c) 14

(d) 36

5. What is the output of the following code snippet?

```
def f5(init):
    l = [x for x in init]
    def rf(it=None):
        if it and it not in l:
            l.append(it)
        return l
    return rf

g5 = f5([2, 3])
h5 = f5([3, 4])
g5(2)
h5(5)
print(g5(), h5())
```

(a) [2, 3, 4, 5] []

(b) [2, 3] [3, 4, 5]

(c) [2, 3, 5] [3, 4, 2, 5]

(d) [2, 3, 4, 5] [2, 3, 4, 5]

6. What are the contents of lst after the following assignment?

```
lst = [(s, len(s)) for s in ('apple', 'banana', 'fish')]
```

(a) [5, 6, 4]

(b) [('apple', 5), ('banana', 6), ('fish', 4)]

(c) [('apple', 3), ('banana', 3), ('fish', 3)]

(d) [('apple', 'banana', 'fish'), (5, 6, 4)]

7. What are the contents of lst after the following assignment?

```
lst = [[(x,y) for x in range(3) if x<y] for y in range(3)]
```

(a) [(0, 1), (0, 2), (1, 2)]

(b) [(0, 1), [(0, 2), (1, 2)]]

(c) [[(0, 1)], [(0, 2)], [(1, 2)]]

(d) [[], [(0, 1)], [(0, 2), (1, 2)]]

8. What is the output of the following code snippet?

```
d = {}
for x in range(10):
    if x//2 in d:      # '//' performs integer division
        d[x//2] += x
    d[x] = 0

print(d[1], d[2])
```

(a) 0 0

(b) 2 4

(c) 5 9

(d) 9 8

9. What is the output of the following code snippet?

```
d = {i:[] for i in range(10)}
for a in range(10):
    for b in range(10):
        d[a].append(a*b)

print(d[4])
```

(a) []

(b) 16

(c) [0, 4, 8, 12, 16, 20, 24, 28, 32, 36]

(d) [0, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144]

10. What is the worst-case run-time complexity of retrieving the element in the *last* position of an array-backed list?

(a) O(1)

(b) O(log N)

(c) O(N)

(d) $O(N^2)$

11. What is the worst-case run-time complexity of inserting an element into the *middle* of an array-backed list?

(a) O(1)

(b) O(log N)

(c) O(N)

(d) $O(N^2)$

12. What is the worst-case run-time complexity when using binary search to search for an element *that does not exist* in a sorted, array-backed list?

    (a) O(1)

    (b) O(log N)

    (c) O(N)

    (d) $O(N^2)$

13. Recognizing that insertion sort is least efficient when its input is in reverse order, John P. Hacker modifies our implementation so that it performs an initial check to see if the input is in reverse order and, if so, simply reverses and returns the list instead of sorting it. Otherwise, insertion sort works as before. What is the worst-case run-time complexity of this modified version (across all inputs, sorted, reversed, or unsorted)?

    (a) O(1)

    (b) O(log N)

    (c) O(N)

    (d) $O(N^2)$

14. What is the worst-case run-time complexity of deleting the middle element of a singly-linked list, given that we already have a reference to the node we want to delete?

    (a) O(1)

    (b) O(log N)

    (c) O(N)

    (d) $O(N^2)$

15. What is the worst-case run-time complexity of insertion sort, if implemented on top of a doubly-linked list?

    (a) O(1)

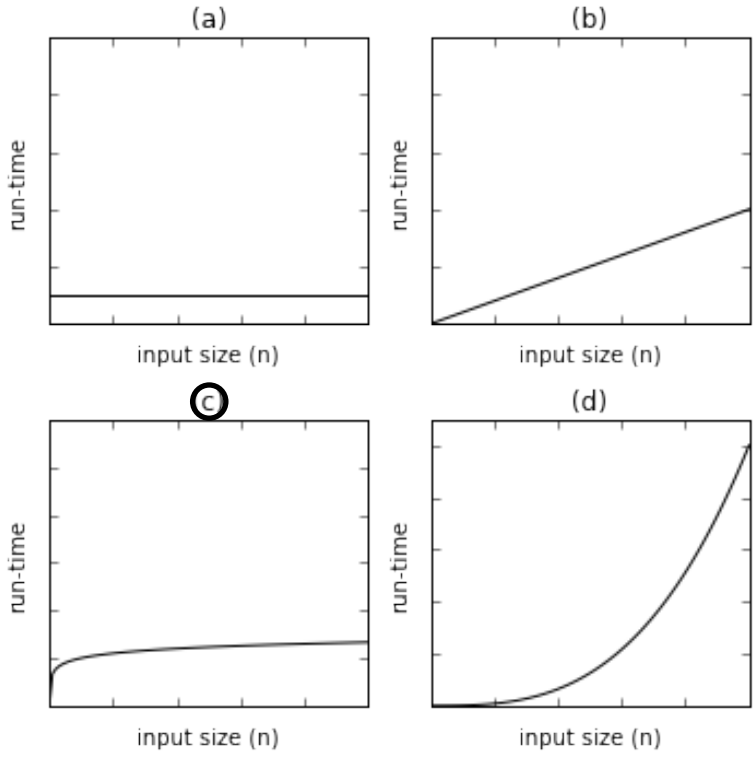
    (b) O(log N)

    (c) O(N)

    (d) $O(N^2)$

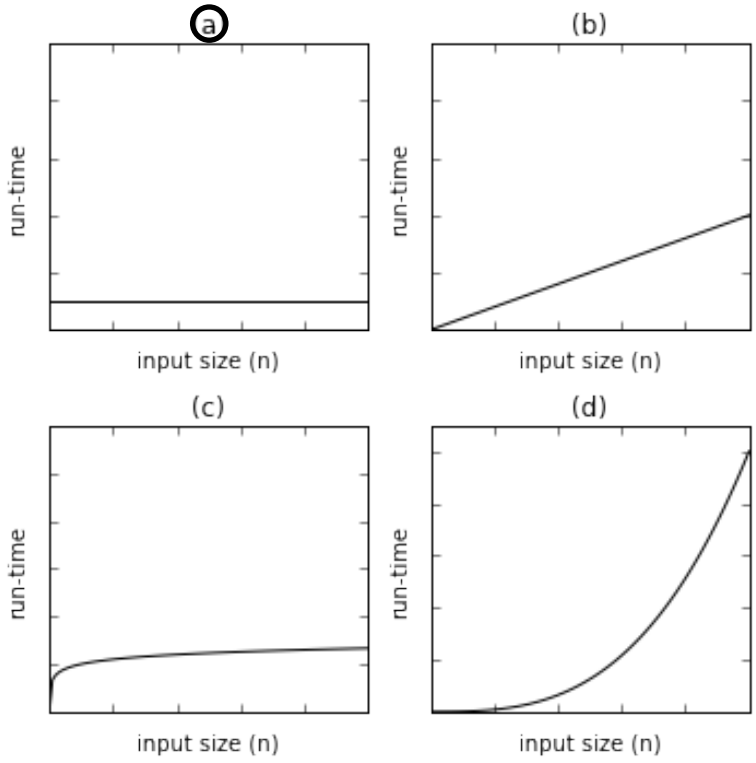
16. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_16(n):
    res = 1
    while n > 10:
        for i in range(10):
            res += i * n
        n = n // 2
    return res
```



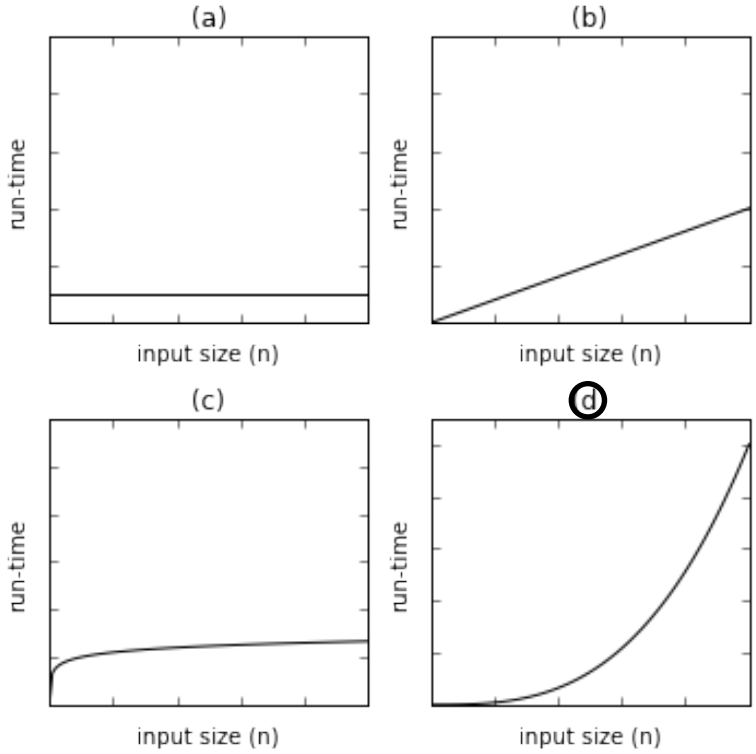
17. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_17(lst): # lst is an array-backed list
    n = len(lst)
    for i in range(100):
        a = random.randrange(n)
        b = random.randrange(n)
        lst[a], lst[b] = lst[b], lst[a]
    return lst
```



18. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_18(lst): # lst is an array-backed list
    n = len(lst)
    for i in range(2**n):
        lst[i%n] += i
    return lst
```



19. Which of the following could work as the body of `__str__` in a class that already has a functional `__iter__` method?

(a) `return self.join(', ')`

(b) `return [str(x) for x in self]`

(c) `return x.join(', ') for x in self`

(d) `return ', '.join(str(x) for x in self)`

20. Which snippet correctly completes the following binary search implementation for an array list which assumes the underlying data is sorted in *descending order* (e.g., [9, 7, 5, 3, 1])?

```
def bin_search_descending(self, val):
    bot = 0
    top = len(self) - 1
    while bot <= top:
        mid = (bot + top) // 2
        ____________________
        ____________________
        ____________________
        ____________________
        else:
            return True # val found in list
    return False # val not found in list
```

(a)
```
if val < self[mid]:
    bot = mid + 1
elif val > self[mid]:
    top = mid - 1
```

(b)
```
if val > self[mid]:
    bot = mid + 1
elif val < self[mid]:
    top = mid - 1
```

(c)
```
if val > self[mid]:
    top = mid + 1
elif val < self[mid]:
    bot = mid - 1
```

(d)
```
if val > self[mid]:
    mid = top - 1
elif val < self[mid]:
    mid = bot + 1
```

21. The following implementation of `__delitem__` (given that `idx` is a positive, valid index) in an array-backed list doesn't appear to work correctly. How would you go about fixing it?

```
1  for i in range(len(self.data)-1, idx, -1):
2     self.data[i-1] = self.data[i]
3  del self.data[len(self.data)-1]
```

(a) Change line 2 to: `self.data[i] = self.data[i+1]`

(b) Change line 1 to: `for i in range(idx+1, len(self.data)):`

(c) Change line 1 to: `for i in range(len(self.data), idx+1, -1):`

(d) Change line 2 to: `self.data[i], self.data[i-1] = self.data[i-1], self.data[i]`

22. Which snippet completes the following method so that it correctly implements an iterator that yields the values of an array-backed list in sorted order, without actually modifying the underlying array?

```
def sorted_iter(self):
    visited = []
    while len(visited) < len(self):
        min_idx = 0
        while min_idx in visited:
            min_idx += 1
        for i in range(len(self)):
            if ____________________________________
                min_idx = i
        visited.append(min_idx)
        yield self[min_idx]
```

(a) `i in visited and self[min_idx] < self[i]:`

(b) `visited[-1] < i and self[i] < self[min_idx]:`

(c) `i not in visited and self[i] < self[min_idx]:`

(d) `visited[-1] != i and self[min_idx] < self[visited[-1]]:`

23. Which snippet completes the following method so that it correctly removes all elements between indexes `start` and `end` (inclusive) from the underlying doubly-linked list (with a sentinel `head` node)?

```
def remove_range(self, start, end):
    assert(start <= end and end < len(self))
    n = self.head.next
    for i in range(len(self)):
        if i == start:
            start_node = n
        if i == end:
            end_node = n
        n = n.next
    ____________________________
```

(recall that when assigning to multiple targets, the expressions on the right side of the assignment operator are all evaluated first, then the values are assigned to the targets one at a time, *from left to right*)

(a) `start_node.next, end_node.prior = start_node.prior, end_node.next`

(b) `start_node.prior, end_node.next = end_node.next, start_node.prior`

(c) `start_node.prior.next, end_node.next.prior = start_node.next, end_node.prior`

(d) `start_node.prior.next, end_node.next.prior = end_node.next, start_node.prior`

24. Which snippet completes the body of the loop in the following method so that it correctly removes all occurrences of `value` from the underlying doubly-linked list (with a sentinel `head` node)?

```
def remove_all(self, value):
    n = self.head
    while n.next is not self.head:
        if n.next.val == value:
            ___________________________________
         else:
            n = n.next
```

(a) `n.next, n.prior = n.prior, n.next`

(b) `n.next, n.next.prior = n.next.next, n`

(c) `n.next.prior, n.next = n, n.next.next`

(d) `n.next.next, n.next.prior = n.next.next.next, n`

25. Which snippet completes the body of the loop in the following method so that it correctly reverses the order of the nodes in the underlying doubly-linked list (with a sentinel `head` node)?

```
def reverse(self):
    self.head.next, self.head.prior = self.head.prior, self.head.next
    n = self.head.next
    while n is not self.head:
        ______________________________
        n = n.next
```

(a) `n.next, n.prior = n.prior, n.next`

(b) `n, n.prior = n.prior, n.next.next`

(c) `n.next.prior, n.prior = n.next, n.prior`

(d) `n.next.prior, n.prior.next = n.prior, n.next`