CS 331 Midterm Exam 1

Wednesday, June 14th, 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. Which of the following snippets is equivalent to the statement "`l1 = list(range(5,110,7))`"?

(a)
```
l1 = []
i = 7
while i <= 110:
    l1.append(i)
    i += 5
```

(b)
```
l1 = []
i = 0
while i < (110 + 5):
    l1.append(i)
    i += 7
```

(c)
```
l1 = []
i = 5
while i <= 110:
    l1.append(i)
    i += 7
```

(d)
```
l1 = []
i = 5
while i < 110:
    l1.append(i)
    i += 7
```

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

```
x = 0
for i in range(100):
    if i % 9 == 0:
        x = i
else:
    x = -1
print(x)
```

(a) 0

(b) -1

(c) 99

(d) 98

3. What are the contents of lst2 after the following two statements are carried out?

```
lst1 = [x*x for x in range(1, 5)]
lst2 = [y-1 for y in lst1 if y % 3 == 0]
```

(a) `[8]`

(b) `[9]`

(c) `[2, 5]`

(d) `[6, 24]`

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

```
def fold(fn, lst):
    res = lst[0]
    for x in lst[1:]:
        res = fn(res, x)
    return res

print(fold(lambda a, b: b - a, [1, 3, 5, 7]))
```

(a) 4

(b) -2

(c) -14

(d) 0

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

```
def gen(lim):
    print('Generating up to', lim)
    for i in range(lim):
        print('Yielding', i)
        yield i
        print('Yielded', i)

gen(10)
```

(a)
```
Generating up to 10
```

(b)
```
Generating up to 10
Yielding 0
```

(c)
```
Generating up to
Yielding
Yielded
```

(d) (No output)

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

```
def gen(lim):
    print('Generating up to', lim)
    for i in range(lim):
        print('Yielding', i)
        yield i
        print('Yielded', i)

it = iter(gen(10))
next(it)
```

(a)
```
Generating up to 10
Yielding 0
```

(b)
```
Generating up to 10
Yielding 0
Yielded 0
```

(c)
```
Generating up to 10
Yielding 0
Yielded 0
Yielding 1
```

(d) (No output)

7. Given that `iterable` is an iterable object, which of the following emulates the behavior of a `for` loop to iterate over its contents?

(a)
```
it = iterable
while True:
    i = iter(it)
    x = next(i)
    # do something with x
    if not i:
        break
```

(b)
```
it = iter(iterable)
while True:
    x = next(it)
    # do something with x
else:
    raise StopIteration
```

(c)
```
it = iter(iterable)
while True:
    try:
        x = next(it)
        # do something with x
    except StopIteration:
        break
```

(d)
```
it = next(iterable)
while True:
    try:
        x = iter(it)
        # do something with x
    except StopIteration:
        break
```

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

```
x0 = [0, None]
x1 = [1, None]
x2 = [2, x0]
x3 = [3, x2]

x3[1] = x3[1][1] = x1

print(x2[1][0])
```

(a) `0`

(b) 1

(c) 2

(d) 3

9. What is the worst-case run-time complexity of inserting a new element into an array-backed list?

(a) O(1)

(b) O(log N)

(c) O(N)

(d) $O(N^2)$

10. What is the worst-case run-time complexity of retrieving an element based on its provided index from 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 searching for an element with a given value in an unsorted 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 of prepending a new element to a circular, doubly-linked list?

    (a) O(1)

    (b) O(log N)

    (c) O(N)

    (d) O(N^2)

13. What is the worst-case run-time complexity of removing the last element from a circular, double-linked list?

    (a) O(1)

    (b) O(log N)

    (c) O(N)

    (d) O(N^2)

14. What is the worst-case run-time complexity of concatenating two circular, doubly-linked lists? (Assume that copying either list is not a requirement.)

    (a) O(1)

    (b) O(log N)

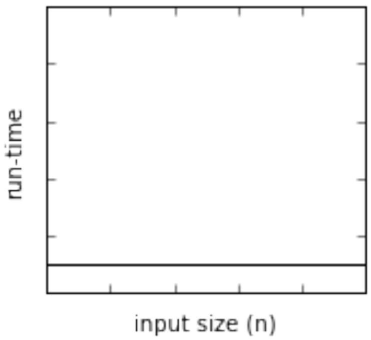
    (c) O(N)

    (d) O(N^2)

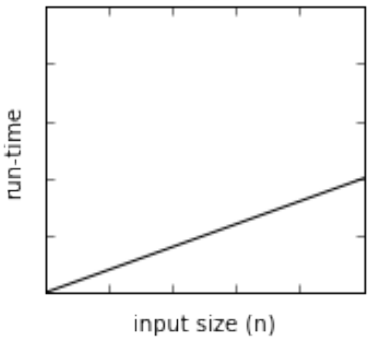
15. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_15(lst): # lst is a Python list
    res = 0
    for x in lst:
        res += res
    return res
```

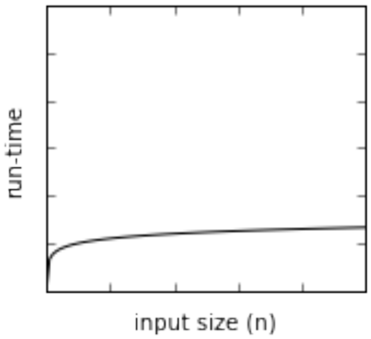
(a)



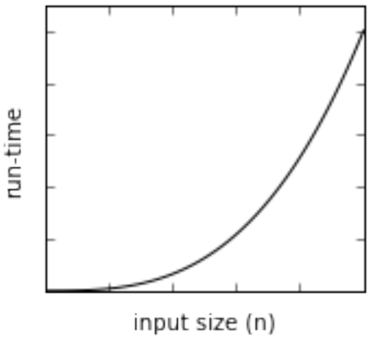
(b)



(c)



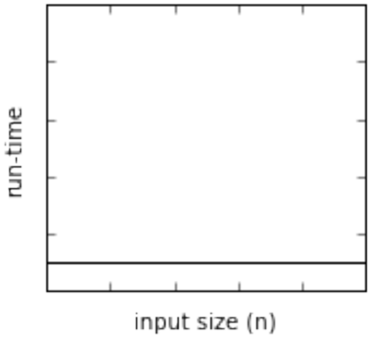
(d)



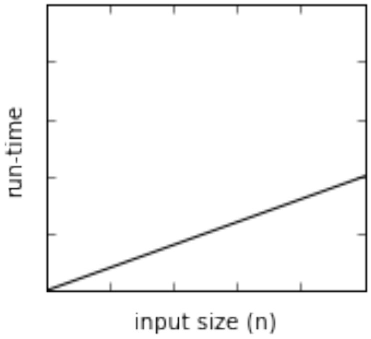
16. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_16(lst): # lst is a Python list
    res = 0
    for x in range(100):
        res += lst[randrange(len(lst))]
    return res
```

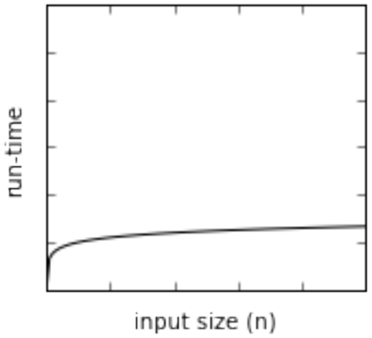
(a)



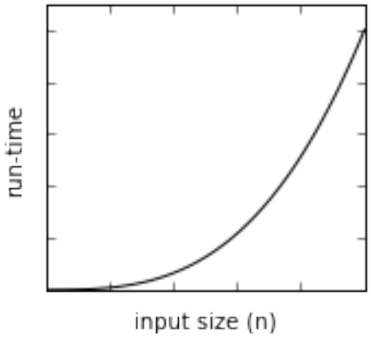
(b)



(c)



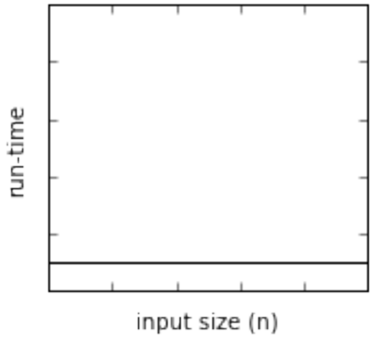
(d)



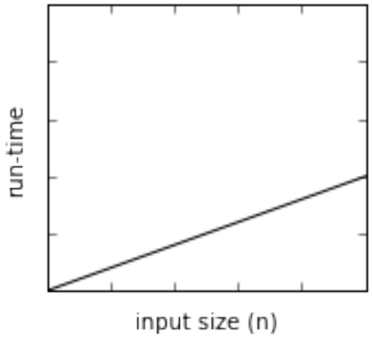
17. Which of the plots best depicts the worst-case run-time complexity of the following function?

```
def f_17(lst): # lst is a Python list
    res = 0
    bot, top = 0, len(lst)
    while bot < top:
        mid = (bot + top) // 2
        res += lst[mid]
        if res < 0:
            bot = mid + 1
        else:
            top = mid - 1
    return res
```

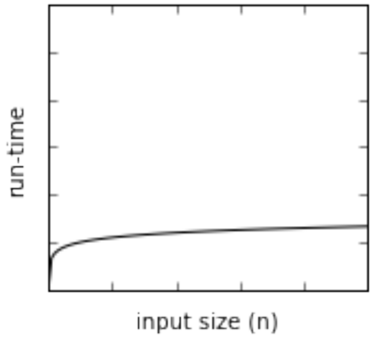
(a)



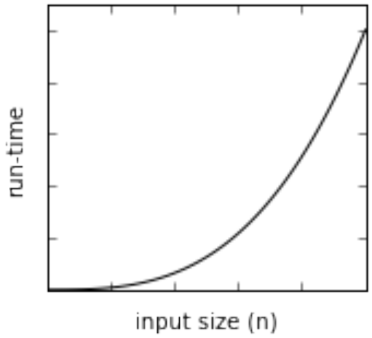
(b)



(c)



(d)



18. Which snippet provides a suitable implementation for `_normalize_idx` in a list implementation, in order to support both negative and positive indexes?

```
def _normalize_idx(self, idx):

    ___________________________
    return nidx
```

(a) `nidx += len(self)`

(b)
```
nidx = -idx
if nidx < 0:
    nidx += len(self)
```

(c)
```
nidx = idx
if nidx < 0:
    nidx += len(self)
```

(d)
```
nidx = idx
if nidx < 0:
    nidx += len(self)
else:
    nidx -= len(self)
```

19. Which snippet correctly completes the implementation of `__add__`, whose description is provided in the accompanying docstring below, in an array-backed list?

```
def __add__(self, other):
    """Implements `self + other_array_list`. Returns a new ArrayList
    instance that contains the values in this list followed by those
    of other."""
    assert(isinstance(other, ArrayList))

    _________________________________
```

(a)
```
self.extend(other)
return self
```

(b)
```
nlst = ArrayList()
nlst.extend(self)
nlst.extend(other)
return nlst
```

(c) `return self + self.extend(other)`

(d) `return self + other`

20. Which snippet correctly implements `remove_first` in an array-backed list, given that the underlying data storage mechanism is a `ContrainedList` (as provided in the `ArrayList` assignment)?

```
def remove_first(self):
    “””Removes and returns the first element in the list.”””

    _________________________________
    return val
```

(a)
```
val = self.data.pop(0)
```

(b)
```
val = self.data[0]
del self.data[0]
```

(c)
```
val = self[0]
del self.data[len(self.data)-1]
```

(d)
```
val = self[0]
del self[0]
```

21. Which snippet completes the following implementation of __iter__, to support iteration over all elements in the underlying circular, doubly-linked list (with a sentinel head node)?

```
def __iter__(self):
    n = self.head.next


    _________________________
```

(a)
```
while n.next is not self.head:
    yield n.val
    n = n.next
```

(b)
```
while n is not self.head:
    yield n.val
    n = n.next
```

(c)
```
while n.next:
    yield n.val
    n = n.next
```

(d)
```
while n:
    yield n.val
    n = n.next
```

22. Which snippet completes the body for the following method in a circular, double-linked list (with a sentinel head node)?

```
def __getitem__(self, idx):
    """Implements `x = self[idx]`"""
    ______________________________
    return n.val
```

(a)
```
n = self.head
while n.next < idx:
    n = n.next
```

(b)
```
n = self.head
for _ in range(idx):
    n = n.next
```

(c)
```
n = self.head.next
for _ in range(idx):
    n = n.next
```

(d)
```
n = self.head.next
for _ in range(idx+1):
    n = n.next
```

23. Which snippet completes the following implementation of `insert` in a circular, double-linked list (with a sentinel head node)?

```
def insert(self, idx, value):
    n = self.head.next


    _____________________________
    self.length += 1
```

(a)

```
for _ in range(idx):
    n = n.next
new = LinkedList.Node(value, n, n.prior)
n.prior = n.prior.next = new
```

(b)

```
for _ in range(idx+1):
    n = n.next
new = LinkedList.Node(value, n, n.next)
n.next.prior = n.next = new
```

(c)

```
for _ in range(idx-1):
    n = n.next
new = LinkedList.Node(value, n.prior, n)
n.next = n.prior
n = new
```

(d)

```
for _ in range(idx):
    n = n.next
new = LinkedList.Node(value, n.prior, n)
n.prior.next = n.prior = new
```