CS 331 Spring 2020

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. Consider the following code:

```
l = [[w, 1] for w in 'ob la di ob la da la la la la'.split()]
for i in range(len(l)):
    for j in range(len(l)):
        if i != j:
            if l[i][0] == l[j][0]:
                l[i][1] += 1
```

   After executing the above, what does the expression `l[:3]` evaluate to?

   (a) `['ob', 'la', 'di']`

   (b) `{'ob': 2, 'la': 6, 'di': 1}`

   (c) `[['ob', 2], ['la', 6], ['di', 1]]`

   (d) `[['ob': 8], ['la', 4], ['di', 9]]`

2. Consider the following code:

```
d = {}
l = 'ob la di ob la da la la la la'.split()
for i in range(len(l)-1):
    if l[i+1] in d:
        d[l[i+1]].append(l[i])
    else:
        d[l[i+1]] = [ l[i] ]
```

   After executing the above, what does the expression `d['la']` evaluate to?

   (a) `[1, 4, 6, 7, 8, 9]`

   (b) `[['ob'], ['di', 'ob'], ['da']]`

   (c) `['di', 'da', 'la', 'la', 'la']`

   (d) `['ob', 'ob', 'da', 'la', 'la', 'la']`

3. Given these variable definitions:

```
t = (1, 2, 3)
s = 'hello'
l = list(range(10))
```

   Which of the following will result in an exception being raised?

   (a) `s.append('!')`

   (b) `del l[2:5]`

   (c) `iter(t)`

   (d) `t[1:]`

4. Consider the following code:

```
class Foo:
  def bar(self, val):
      self.attrib = val

f = Foo()
```

After executing the above, which of the following is equivalent to the expression `f.bar(10)`?

(a) `Foo.bar(self, 10)`

(b) `Foo.bar(f, 10)`

(c) `self.bar(f, 10)`

(d) `f.__bar__(10)`

5. Given that the target of a `for` loop is an iterable object `it`, what is the first action performed on the target to commence iteration?

(a) `iter(it)`

(b) `next(it)`

(c) `yield(it)`

(d) `raise StopIteration`

6. When implementing a class-based iterator, what action should be taken in the `__next__` method when we are out of values to return?

(a) `return`

(b) call `next` recursively

(c) `yield`

(d) `raise StopIteration`

7. Which of the following is the closest analogue to the expression `l1 + l2`, where `l1` and `l2` both refer to built-in Python lists?

(a) `l1.append(l2)`

(b) `l1.extend(l2)`

(c) `l1.__add__(l2)`

(d) `l1.__iter__(l2)`—

8. Which of the following lists common runtime complexity classes in strictly worsening order (i.e., asymptotically fastest to slowest)?

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

9. Which of the following time functions based on input size n is bounded by the best (i.e., fastest) asymptotic runtime complexity class, relative to the others?

   (a) $5n^3$
   (b) $50n + 85$
   (c) $10n^2 + 100n$
   (d) $2^n - \log n + 50$

10. Which of the following lists operations/algorithms discussed in class in strictly worsening order (i.e., fastest to slowest) of asymptotic runtime complexity?

    (a) binary search, linear search, array indexing, insertion sort
    (b) binary search, array indexing, linear search, insertion sort
    (c) array indexing, linear search, insertion sort, binary search
    (d) array indexing, binary search, linear search, insertion sort

11. What describes the worst-case runtime complexity of deleting an arbitrary element from an array-backed list?

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

12. What describes the worst-case runtime complexity of assigning a new value to an arbitrary index in an array-backed list?

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

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):
    s = 0
    j = 0
    for i in range(100):
        s += lst[j]
        j += 1
        if j == len(lst):
            j = 0
    return s
```

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

14.
```
def fB(lst):
    it1 = iter(lst)
    it2 = iter(lst)
    next(it2)
    while True:
        try:
            if next(it1) == next(it2):
                return True
        except:
            return False
```

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

15.
```
def fC(lst):
    i = 0
    while i < len(lst)-1:
        if lst[i] == lst[i+1]:
            del lst[i]
        else:
            i += 1
```

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

Python data structures, Part I (8 points):

Suppose that we choose to represent a text document as a dictionary, with chapter names as keys and their textual contents as values. Here's a sample document using this representation:

```
{
  'c1': 'i like cake',
  'c2': 'cake is life',
  'c3': 'cake cake cake',
  'c4': 'i eat cake'
}
```

To assist in searching the document, it would be useful to construct an *index* where each word maps to a dictionary which in turn indicates how many times that word appears in a given chapter. Here is the index for the above document (note its structure — a dictionary containing keys that map to nested dictionaries):

```
{
  'cake': {'c1': 1, 'c2': 1, 'c3': 3, 'c4': 1},
  'eat':  {'c4': 1},
  'i':    {'c1': 1, 'c4': 1},
  'is':   {'c2': 1},
  'life': {'c2': 1},
  'like': {'c1': 1}
}
```

WP1 Implement the function `build_index`, which takes a dictionary representing a text document and returns a dictionary representing an index for that document, as described above.

To break a string down into a list of its component words, you should use the `split` method. E.g., `'i like cake'.split()` returns the list `['i', 'like', 'cake']`

Python data structures, Part II (8 points):

Given an index, as you constructed for the previous problem, we might wish to perform a search for matching chapters using a *query* consisting of one or more words. The result of a search would be a dictionary mapping chapter names to the total number of times all the query words appear in that chapter (only chapters with at least one match would appear in the result).

If we were to run the query `'you like cake'` against the sample index presented in the previous problem description, we would obtain the result:

```
{
  'c1': 2,
  'c2': 1,
  'c3': 3,
  'c4': 1
}
```

On the other hand, the query `'i like food'` would give us the result:

```
{
  'c1': 2,
  'c4': 1
}
```

WP2 Implement the function `search_index`, which takes an index (a dictionary mapping words to nested dictionaries) and a query string, and returns a result dictionary.

Again, you may use the `split` method to break the query down into its component words.

Array-backed list (8 points):

WP3 Implement the method `truncate` for the array-backed list data structure, which, when called with two parameters `start` and `stop`, removes all elements from the list *except* those between index `start` (inclusive) and index `stop` (exclusive).

E.g., given the list `lst` with contents `['a','b','c','d','e','f','g','h']`, calling `lst.truncate(2,6)` results in the list being truncated to `['c','d','e','f']`.

Your implementation may only use the following methods on the built-in list (referenced via `self.data`):

- Getting the length of the list
- Accessing (reading/writing) a valid index ≥ 0
- Deleting the last slot of the list

Your implementation should not use any other `ArrayList` methods nor create any additional lists, and should work in $O(N)$ time, where N is the number of elements in the list. You may assume that `start` and `stop` are valid index values ≥ 0, and the list is not empty.