CSCI 141 Computer Science I

Exam: Midterm II Written Examination
Section 7
Duration: 110 minutes
Instructor: Monika Polak

11/14/2019

Full Name (printed): ________________________________

Instructions:

- The exam contains 17 pages. The last two pages are scrap paper. Please make sure you have all pages.
- The exam contains a total of 66 points.
- The exam is closed book and notes.
- If you require clarification of a question, please raise your hand.
- All coding questions pertain to the Python language.

True or False (9 points)

For each of the following statements, circle TRUE if the statement is true. Circle FALSE if the statement is false.

1. TRUE / FALSE Elements of a Python list must all be of the same data type.
2. TRUE / FALSE A *queue* is an example of a LIFO (Last In First Out) data structure.
3. TRUE / FALSE A dictionary may store the same key with different values.
4. TRUE / FALSE A *stack* is an example of a FIFO (First In First Out) data structure.
5. TRUE / FALSE A *dequeue* allows for removal of the first or last element in a queue.
6. TRUE / FALSE Insertion sort on an already sorted list runs faster than selection sort on a randomly ordered list.
7. TRUE / FALSE Accessing a value by index in a Python list is an $O(1)$ operation.
8. TRUE / FALSE A line at the grocery store is a real world example of a stack.
9. TRUE / FALSE Lists are mutable.

Multiple Choice and Short Answer (15 points)

Indicate the correct response for each question.

1. (1 point) What is the output from the following code?

```
ListA = [ 1, 2, 3 ]
ListB = ListA
ListA.append(4)
print(ListB)
```

(a) `[ 1, 2, 3, 4 ]`
(b) `[ ]`
(c) `[ 1, 2, 3 ]`
(d) `[ 1, 2, 4 ]`

2. (1 point) What is the output from the following code?

```
@dataclass(frozen=True)
class LinkNode:
    value: Any
    rest: Union[None, 'LinkNode']

lnk1 = LinkNode( 2, LinkNode( 4, None))
lnk1.rest = None
```

(a) `LinkNode(value=4, rest=None)`
(b) `LinkNode(value=2, rest=None)`
(c) `LinkNode(value=2, rest=LinkNode(value=4, rest=None))`
(d) `InstanceError`

3. (1 point) Which of the following statements (circle all that apply) create an empty Python list?

(a) `myLst = {}`

(b) `myLst = []`

(c) `myLst = None`

(d) `myLst = list()`

4. (1 point) What is the output of the following of code?

```
song = ''Strawberries, cherries and an angel kissing spring''
print( song.split() )
```

Output:

5. (1 point) What is the output of the following of code?

```
val = []
for i in range(2,6):
    val = val + [2*i]
print(val)
```

Output:

6. (1 point) Write code to change the "year" value from 1964 to 2018.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Code:

7. (1 point) What is the output of the following of code?

```
list = [ 5 , 6 , ‘‘abc’’, 8]
print(6 in list)
```

Output:

8. (1 point) What is the output of the following of code?

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Output:

9. (1 point) What is the output of the following of code?

```
thisdict = { "brand": "Ford", "model": "Mustang", "year": 1964}
for x in thisdict:
  print(x)
```

Output:

10. (1 point) What is the output of the following of code?

```
a=’’10’’
for i in range(4):
    for j in range(5)
        a+=1
print(a)
```

Output:

7. (5 points) For each operation listed below, give the complexity that most accurately describes the operation. Assume that N is the size of the list or stack or queue, whichever applies. All references to stacks and queues refer to our linked sequence implementation of these data structures. Note that the each complexity may be used more than once.

- $O(1)$
- $O(N \log N)$
- $O(\log N)$
- $O(N)$
- $O(N^2)$

(a) ____________ dequeueing the front element from a queue.

(b) ____________ popping the top element from a stack.

(c) ____________ pushing a new element onto a stack.

(d) ____________ sorting an already sorted Python list using insertion sort.

(e) ____________ sorting a reverse sorted Python list using insertion sort.

(f) ____________ sorting a randomly ordered list using merge sort.

(g) ____________ searching for a value in a sorted Python list using binary search.

(h) ____________ enqueueing a new element in a queue.

(i) ____________ sorting a list that is already in order using quick sort.

(j) ____________ sorting a randomly ordered list using quick sort.

Algorithm Tracing (14 points)

1. **(4 points)** Consider the following list of numbers:

```
[ 7, 2, 8, 4, 5, 3, 6 ]
```

Show the progression (the modified list) that insertion sort produces while sorting this list in ascending order. **Each iteration refers to one complete iteration of the *outer loop* (`for`) of the function**. (For a list of size N, the outer loop iterates $N-1$ times.)

Note: don't confuse this with other sorting algorithms. Make sure you show the progression of the **insertion sort** algorithm. Leave any unnecessary lines blank or add more lines if needed.

- After iteration 1: ____.
- After iteration 2: ____.
- After iteration 3: ____.
- After iteration 4: ____.
- After iteration 5: ____.
- After iteration 6: ____.

2. **(4 points)** Given the following sorted list of length 13:

```
numbers = [ 1, 3, 5, 6, 8, 9, 9, 11, 15, 16, 18, 20, 23 ]
```

Suppose we are using the **recursive binary search** algorithm to identify

the index containing the value 15

(or return `None` if it is not present). Assume the middle element is queried each step:

```
mid_index = ( start + end ) // 2
mid_value = data [ mid_index ]
```

where `start` is the starting index and `end` is the ending index of the interval being considered.

What is the sequence of **indices** (`start`, `mid_index`, `end`) used across all recursive calls made by the algorithm?

Remember that list indexing starts at 0.

If you think that not all of the listed recursive calls are necessary, leave any unnecessary lines blank. Add more lines if needed.

- Initial call: `start:` ________________ `mid_index:` ________________ `end:` ________________
- Next call: `start:` ________________ `mid_index:` ________________ `end:` ________________
- Next call: `start:` ________________ `mid_index:` ________________ `end:` ________________
- Next call: `start:` ________________ `mid_index:` ________________ `end:` ________________
- Next call: `start:` ________________ `mid_index:` ________________ `end:` ________________

3. **(6 points)** Consider text file text.txt containing a list of cryptographic hash functions, year designed and length of output in bits. Hash functions that produce output shorter than 224 bits are not considered secure.

```
MD4 1990 128
MD5 1992 128
BLAKE2b 2012 512
SHA-224 2002 224
SHA-256 2002 256
SHA3-224 2012 224
SHA3-512 2012 512
WHIRPOOL 2004 512
```

The following code creates a dictionary that for each year contains a list of secure hash functions that were designed.

```
hash_functions = {}
with open("text.txt") as f:
    while True:
        line = f.readline().split()
        if line==[]:
            break
        if ____________________________
            if line[1] in hash_functions:
                hash_functions[line[1]].append(line[0])
            else:
                hash_functions[line[1]]=[line[0]]
```

(a) Fill in the missing line (`if` statement) in the code above.

Part (b) in the next page.

(b) Show the progression (the modified dictionary) that this code produces while iterating. Each iteration refers to one complete iteration of the *while loop*.

Iteration#	Modified dictionary after iteration
1	
2	
3	
4	
5	
6	
7	
8	

Analysis (12 points)

1. **(8 points)** Using our implementation of stacks and queues from class, and using diagrams as we have done in class (rectangles or "boxcars" to represent objects, with an arrow to represent a reference from one object to another), draw the following diagrams:

   - Draw a diagram representing a stack data structure after the following sequence of statements. You need only show the final result.

     ```
     stk = make_empty_stack()
     push(stk, 'A')
     push(stk, 'B')
     pop(stk)
     push(stk, 'C')
     value = top(stk)
     pop(stk)
     push(stk, 'D')
     push(stk, 'E')
     ```

   - Draw a diagram representing a queue data structure after the following sequence of statements. You need only show the final result.

     ```
     que = make_empty_queue()
     enqueue(que, 'A')
     enqueue(que, 'B')
     dequeue(que)
     enqueue(que, 'C')
     enqueue(que, 'D')
     enqueue(que, 'E')
     dequeue(que)
     ```

2. **(4 points)** Test the function below:

```
def is_match(seq1, seq2):
    """
    Boolean function to check if linked sequences are identical
    :param seq1: first sequence
    :param seq2: second sequence
    :return: true if they are identical
    """

    if seq1 == None and seq2 == None:
        return True
    elif seq1 == None or seq2 == None:
        return False
    else:
        if seq1.value == seq2.value:
            return is_match(seq1.rest, seq2.rest)
        else:
            return False
```

Write six test cases for function `is_match` in the table below. For each test case:

- Identify the input values passed as arguments to the test;
- Identify the resulting return value; and
- Explain what distinct situation this test case checks.

Test#	Input values	Return values	Situation Tested
1			
2			
3			
4			
5			
6			

Design Problem (16 points)

1. **(7 points)** You work for IKEA, and are writing a program to keep track of some information about various products in the inventory of IKEA shop. For each product, you track:

   - Name (e.g. VINTER Candle holder)
   - Unit price (e.g. $3.99)
   - Quantity on hand (e.g. 100)
   - If the IKEA family disccount apply (True/False)

   (a) Define a dataclass `Product`, using `dataclasses`, that contains the type of information described above.

   (b) Write a statement that creates a new `Product` corresponding to a MINDE mirror (Name), $12.99 (Unit price), 100 (Quantity on hand), False (IKEA family disccount apply).

   (c) Given a Python list of `Product` objects, (`prodList`), write a function that iterates through the list and prints out the name and unit price of every product for which the IKEA family discount applies.

2. **(9 points)**

   Design a solution for maintaining a menu for a restaurant. For each meal, store its name and price. The program must provide the following functionality:

   - The `add` function takes the menu and a meal to insert. To `add` a meal means to append or insert the meal.
   - To `delete` a meal means to remove the meal from the menu. The `delete` function takes as input the menu and a meal to remove.
   - The `print_cheaper_than` function takes in input the menu and a number, `price`, and prints a list of the meals cheaper than `price` (together with prices).

   a) Describe the data structure(s) you will create to solve this problem. If using a dataclass, include the names and types of each field. For each data structure chosen, state why you chose that one versus other available choices.

b) Write the code for the `add` function.

c) Write the code for the `print_cheaper_than` function.

SCRAP PAPER

SCRAP PAPER