**CSI 33** **Midterm Exam In-class Practice**

**Part 1** Answer True/False and Multiple Choice questions

**1.** Which of the following is a Θ(n) operation?
 (a) Sorting a list with Selection sort
 (b) Finding the $i^{th}$ item in a Python list.
 (c) Re-assigning the element at the end of a Python list.
 (d) Deleting an item from the middle of a Python list.

Answer: (d)
Explanation: Selection Sort is $\Theta(n^2)$ operation;
random access in Python list (finding $i^{th}$ item, `myList[i]`) is Θ(1) operation;
re-assigning an element at the end of a Python list ( `myList[len(myList)-1] = ...`) is also Θ(1) operation;
When a middle element is deleted, the "right half" of the values, about $\frac{n}{2}$ of them, must be shifted one space to the left, which gives Θ(n) running time.

**2.** Which of the following is **not** true of Python dictionaries?
 (a) They are implemented as hash tables.
 (b) Lookup is very efficient.
 (c) Values must be immutable.
 (d) All of the above are true.

Answer: (c)
Explanation: Indeed, Python dictionaries are implemented as hash tables, the lookup, insertion, deletion are all Θ(n) operations. Keys must be immutable (as this is the way to "access" the associated value with it). Values can me mutable.

**3.** How many iterations will the while loop of the *Binary Search* do when searching for 21 in the sequence [1, 5, 12, 14, 17, 21, 28]? Use the Binary Search algorithm I presented in class.

(a) 5 (b) 4 (c) 3 (d) 2

Answer: (d)
Explanation: look at the algorithm of the *Binary Search* for the key information:
1) the middle value accessed by the index $\left\lfloor \frac{high+low}{2} \right\rfloor$ , where *low* = *0* and *high* = *len(myList) – 1* initially
2) the while loop stops as soon as *low* > *high*
3) when the middle element checked for equality with the target value:
- if it is equal, then the index is returned, and
- if not, left half (stepping one left for the high index) or the right half (stepping one to the right for the low index) is "chosen"
21 is present in the sequence, therefore, the exit condition from the loop will be the location of this element.
We will begin by selecting index $\left\lfloor \frac{6+0}{2} \right\rfloor = 3$ , the value at the 3rd position is 14, not 21.
Since 21 is greater than 14, the *low* index is adjusted to *low = 3+1 = 4*.

$2^{nd}$ iteration of the while loop: the "middle index" is $\left\lfloor \frac{6+4}{2} \right\rfloor = 5$ , the value at the $5^{th}$ position is 21. It is the target value, therefore, the position 5 is returned and the while loop is terminated.
The Binary Search algorithm performed 2 iterations of the while loop.

Here is what you can present as an explanation, if asked for:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 5 | 12 | 14 | 17 | 21 | 28 |

$1^{st}$ iteration: mid = (0+6)//2 = 3, not it
21 > 14 or 14 < 21, hence low = 3+1 = 4

$2^{nd}$ iteration: mid = (4+6)//2 = 5, found it!
return 5

## **Part 2.** Answer short-answer questions

**1.** Consider the following code fragment:

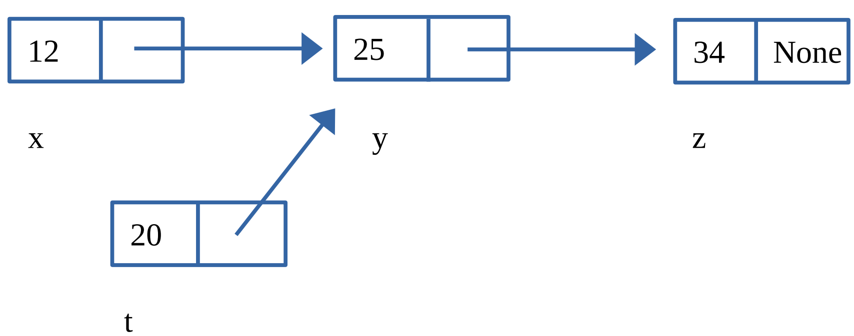
```
from ListNode import *

z = ListNode(34)
y = ListNode(25,z)
x = ListNode(12,y)
t = ListNode(20,y)
```

What will be produced by this code fragment (draw a pictorial representation)?

For your reference, the definition of the `ListNode` class:

… `skipped`

Answer: graphical



**2.** Give a theta analysis of the time efficiency of the following code fragment. Provide explanations.

```
n = int(input("Enter a positive integer:"))     3 steps
myList = []                                     1 step
while n > 1:                     n/3 iterations, 1 step for comparison
     myList.insert(0, n)         insertion is Θ(n) operation, but see
     n -= 3           1 step                          more details below.
```

**More details:** note that initially the list is empty, and the next element is inserted into the "head" position (0th position), so we cannot claim that at each iteration the list has n elements and all of them are shifted 1 space to the right.

So it is better to see what is going on at each iteration from the very beginning:

1st iteration, the value of n is inserted into an empty list : 1 step

2nd iteration: the value of myList[0] is shifted one space to the right, and the n-3 is placed into the 0th position: 2 steps

3rd iteration: the values of myList[1] and the myList[0] are shifted one position to the right, and the value of n-6 is inserted into the 0th position: 3 steps

4th iteration: the values of myList[2], myList[1], and myList[0] are shifted to the right, and the value of n-9 is inserted into the 0th position: 4 steps

…

the loop will stop when $n-3k \le 1$… so there will be about $\frac{n}{3}$ iterations:

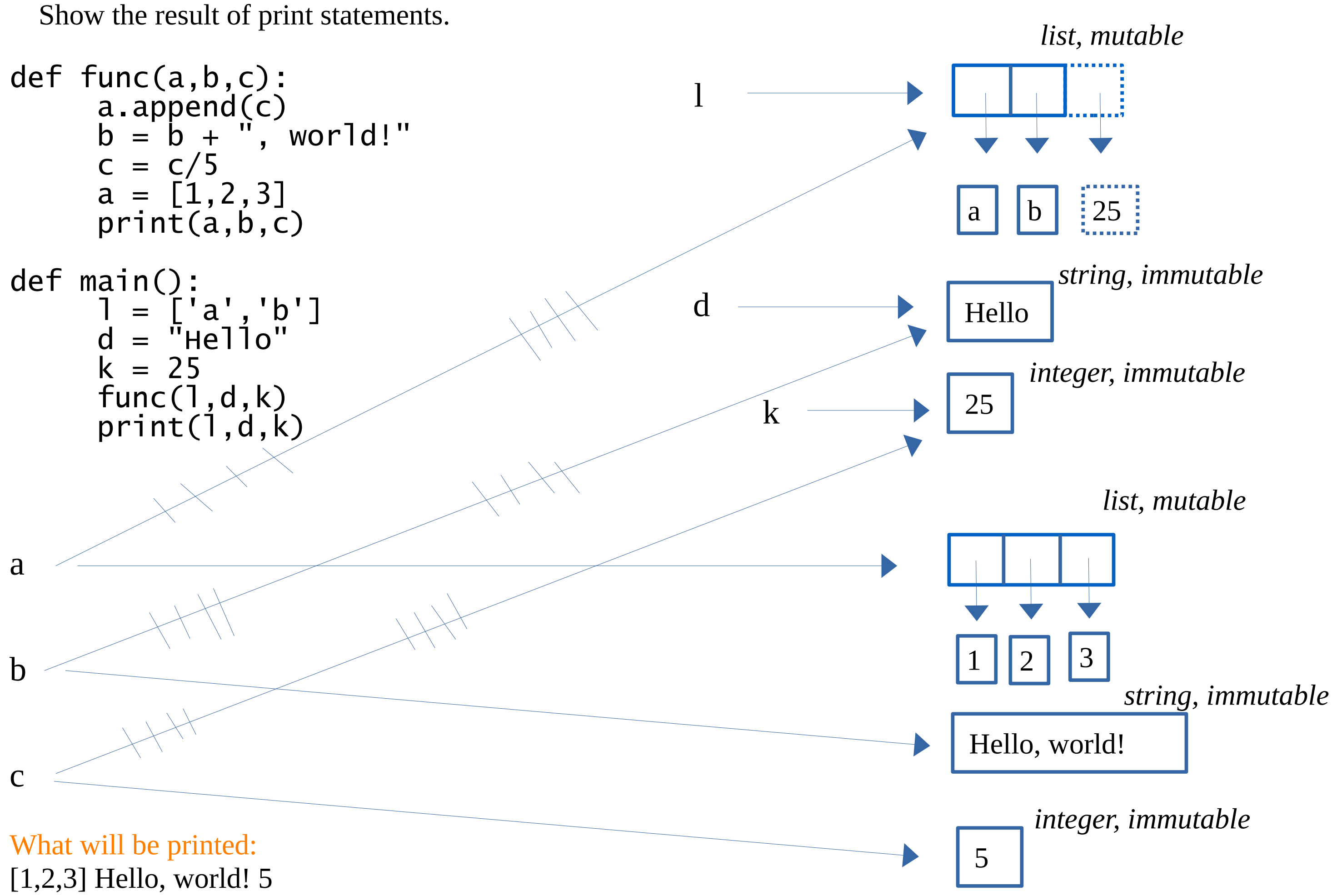
1 step + 2 steps +3 steps + 4 steps + … + $\frac{n}{3}$ steps = arithmetic sequence = $\frac{\left(1+\frac{n}{3}\right)\left(\frac{n}{3}\right)}{2} = \ldots = \frac{n}{6}+\frac{n^2}{6}$

Hence T(n) = $4+2\cdot\left(\frac{n}{6}+\frac{n^2}{6}\right)=4+\frac{n}{3}+\frac{n^2}{3}=\Theta\left(n^2\right)$

Answer: $T(n) = \Theta(n^2)$

**3.** Give pictorial representation of the Python's memory during execution of the code given below.
Show the result of print statements.

```
def func(a,b,c):
    a.append(c)
    b = b + ", world!"
    c = c/5
    a = [1,2,3]
    print(a,b,c)

def main():
    l = ['a','b']
    d = "Hello"
    k = 25
    func(l,d,k)
    print(l,d,k)
```




What will be printed:
[1,2,3] Hello, world! 5
[‘a’,’b’, 25] Hello 25

**Part III.**
Here is the running time for each of four requested operations:

| operations | (a) an unordered Python list | (b) a sorted Python list | (c) a Python dictionary (elements of the set are keys, None or True are values) |
|---|---|---|---|
| **add** | $\Theta(n)$<br>we need to check if the element is already in the set, and since the list is unordered, we will have to apply linear search | $\Theta(\log n)$<br>since the elements are ordered, we need to find a position to insert the new record, search can be done with *log n* time (recall binary search on sorted arrays), then append operation on average takes $\Theta(1)$ time | $\Theta(1)$<br>almost all basic operations on dictionaries are $\Theta(1)$, since hash tables with hashing function are used. |
| **remove** | $\Theta(n)$<br>operations of insertion and deletion are $\Theta(n)$ for Python's lists + we need to find an element, and shift all the ones to the right of it one space to the left | $\Theta(n)$<br>first we will need to locate the element with the given name ( $\Theta(\log n)$ operation), then we will need to delete is ($\Theta(n)$ operation on Pyton's lists), hence the result is $\Theta(n)$ | $\Theta(1)$<br>using hashing function the record will be accessed in constant time, and deleted |
| **clear** | $\Theta(1)$<br>if we reassign the data attribute to empty list, e.g. data=[] | $\Theta(1)$<br>if we reassign the data attribute to empty list, e.g. data=[] | $\Theta(n)$<br>either deleting all elements or changing the values of the keys to None |
| **__contains__** | $\Theta(n)$<br>since the list is unordered, we will have to apply linear search | $\Theta(\log n)$<br>search can be done with *log n* time (recall binary search on sorted arrays) | $\Theta(1)$<br>one of the basic operations of Python dictionaries |
| **intersection** | $\Theta(nm)$<br>$\|set1\| = n$ and $\|set2\| = m$<br>Every element from the set1 will be “searched for” in set2 | $\Theta(n \log m)$<br>$\|set1\| = n$ and $\|set2\| = m$<br>Every element from the set1 will be “searched for” in set2, w can use binary search | If n is number of elements in set1 and m is the number of elements of set2, and n < m, then we say the asymptotic running time is $\Theta(n)$<br>$\|set1\| = n$ and $\|set2\| = m$, and $n <= m$ |
| **intersection**<br>the idea is to grab the smaller size set and check if its elements are present in the other, if present, then the element is added to the new set. Worse case scenario – all the elements are to be added | | | |
| **union** | $\Theta(nm)$<br>We will start by adding all the elements of one set to the new set; then we will be | $\Theta(n+m)$<br>Two sets are ordered, so we compare the two first elements of the sets, and | $\Theta(n+m)$<br>All the keys from one dictionary are added to the new dictionary, then each |

| | | | |
|---|---|---|---|
| | grabbing one by one elements from the second set, checking if its presence in the new set – if it is not present, we will add it, if it is – we will move on to the next. | grab the smallest – add it to the new set; then we compare the "next" front/first elements : take the smallest and add it to the new set, etc. | key from the second dictionary is checked for presence in the new dictionary: if is already there, don't add it, otherwise, add it. |
| **difference** | Θ(nm) Set1-Set2: grab an element from Set1 (n elements): if it is not in Set2 (m elements to compare to), add it to the new set, and so forth. | Θ(nm) Set1-Set2: grab an element from Set1 (n elements): if it is not in Set2 (m elements to compare to), add it to the new set, and so forth. | Θ(n) Set1-Set2: grab an element from Set1 (n elements): if it is not in Set2 (Θ(1) operation), add it to the new set, and so forth. |
| | | | |

About operations and their cost on Python's list see pages 99
About operations and their cost on dictionaries see pages 94-95