CSE101 Midterm-2 Review

Q. 1: Binary Search

A. Suppose a list is defined with this statement:

```
>>> consonants = ['b', 'c', 'd', 'f', 'g', 'h', 'j']
```

Show how binary search will proceed step by step and what result will be returned by the binary search algorithm. The brackets enclose the list to be searched, and the midpoint is half way between. An illustrative example is given.

```
>>> bsearch(consonants, 'h')
[b c d *f g h j]
b c d f [g *h j]
Output: 5
>>> bsearch(consonants, 'd')
[b c d *f g h j]
[b *c d] f g h j
b c [d] f g h j
Output: 2
>>> bsearch(consonants, 'e')
[b c d *f g h j]
[b *c d] f g h j
b c [d] f g h j
Output: (Nothing, None, -1)
```

B. Given a list a = [2,4,6,8,10,12] and target element to be 10; what are the mid values (corresponding array elements) in the binary search iterations?

a. 6 and 10
b. 8 and 10
c. 6 and 8
d. 4 and 12

C. Given a list a = [10, 20, 28, 55, 80, 113, 221, 344]. How many iterations required to find 20 using the binary search algorithm?

a. 1
b. 2
c. 3
d. 4

D. Estimate the number of comparisons required to search a list with n items using binary search technique.

Tip: Calculate $\log_2 n$ and round up to the nearest integer (ceiling).

1000 items	10 searches
10000 items	_____14 searches
100000 items	_____17 searches
1000000 items	_____20 searches

Q. 2: Mergesort and Quicksort

A. Below is a test list with 16 numbers. Show how this list would be sorted by a call to msort. The easiest way to do this is to show the groups before each round of merges.

14 52 86 29 20 95 91 72 2 98 4 46 51 38 78 72

The initial groups are given as below:

[14] [52] [86] [29] [20] [95] [91] [72] [2] [98] [4] [46] [51] [38] [78] [72]

After calling merge_groups with size 1

[14 52] [29 86] [20 95] [72 91] [2 98] [4 46] [38 51] [72 78]

After calling merge_groups with size 2

[14 29 52 86] [20 72 91 95] [2 4 46 98] [38 51 72 78]

After calling merge_groups with size 4

[14 20 29 52 72 86 91 95] [2 4 38 46 51 72 78 98]

After calling merge_groups with size 8

[2 4 14 20 29 38 46 51 52 72 72 78 86 91 95 98]

B. The quicksort function divides a list into two smaller regions by calling a helper function named partition and then making recursive calls to sort the two regions. The partition function is given as below where a is a list, p is index of pivot and r is the last index of region which will be partitioned:

```
def partition(a, p, r):
    x = a[p]
    i = p
    for j in range(p+1, r+1):
        if a[j] <= x:
            i += 1
            a[i], a[j] = a[j], a[i]
```

```
    a[p], a[i] = a[i], a[p]
    print ("After partition list = ", a)
    print ("index of pivot = ", i)
```

Explain what the result is of calling partition function for various lists.
An illustrative example is given below:

```
>>>nums = [55, 46, 89, 64, 93, 45, 15, 96]
>>>partition(nums, 0, len(nums)-1)
After partition list =  [15, 46, 45, 55, 93, 89, 64, 96]
index of pivot =  3
>>>nums = [37, 27, 90, 94, 78, 21, 14, 45]
>>>partition(nums, 0, len(nums)-1)
After partition list = [14, 27, 21, 37, 78, 90, 94, 45]
index of pivot = 3
>>>nums = [17, 32, 69, 23, 89, 53, 11, 71]
>>>partition(nums, 0, len(nums)-1)
After partition list = [11, 17, 69, 23, 89, 53, 32, 71]
index of pivot = 1
```

C. Suppose a list is defined with the following assignment statement:

```
elems = ['Rf', 'Sn', 'Au', 'Ge', 'Bh', 'Sr', 'Cn', 'Y']
```

Sort the list using mergesort (msort) technique.

>>>msort (elems)

```
[Rf] [Sn] [Au] [Ge] [Bh] [Sr] [Cn] [Y]
_____________________________________ [Rf Sn] [Au Ge] [Bh Sr] [Cn Y]
_____________________________________ [Au Ge Rf Sn] [Bh Cn Sr Y]
_____________________________________ [Au Bh Cn Ge Rf Sr Y]
```

Q. 4: Recursion and iteration

A. In a Fibonacci series, each number is the sum of the two preceding numbers. The first two numbers in the series are 1 and 1. The simplest is the series 1, 1, 2, 3, 5, 8, etc. Write an iterative and recursive function to return n-th Fibonacci number in a series.

```
#iterative version
```

```
def iterFib(n):
    if n < 2:
        return 1
    else:
        term1 = 1
        term2 = 1
        for i in range(2, n):
            temp = term2
            term2 = term2 + term1
            term1 = temp
        return term2
```

(Any suitable implementation is fine)

```
#recursive version
Def recFib(n):
```

```
# Returns the n-th Fibonacci number
def fib(n):
    if n == 0 or n == 1:  # two base cases
        return 1
    return fib(n - 1) + fib(n - 2)  # two recursive calls
```

B. Complete the following function that uses recursion to find and return the even elements in the list a.

```
# find_evens([1, 2, 4, 5]) returns [2, 4]
# find_evens([1, 2, 3, 4, 6, 5, 7, 9, 8, 10]) returns [2, 4, 6, 8, 10]
def find_evens(a):
    return None  # Replace this with your implementation
```

```
def find_evens(a):
    if len(a) == 0:
        return []
    elif a[0] % 2 == 0:
        return [a[0]] + find_evens(a[1:])
    else:
        return find_evens(a[1:])
```

C. Write a recursive function to find whether a given string is palindrome. A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., `madam` or `nursesrun`.

```
# Return True if the argument is a palindrome, and False if not
def is_palindrome(s):
    if len(s) <= 1:     # a string of 0 or 1 characters is a palindrome
                return True
  elif s[0] != s[-1]: # the first and last characters don't match
```

```
        return False
    else:
        return is_palindrome(s[1:-1])
```

Q. 5: Dictionaries

A. Write assignment statements that create dictionaries for the following sets of data:

- The numbers from 0 to 9, using the number as a key and its spelling as corresponding value.
- SWOT analysis is a study undertaken by an organization to identify its internal strengths and weaknesses, as well as its external opportunities and threats. SWOT stands for Strengths, Weaknesses, Opportunities, Threats. Create a dictionary analysis using the letters in the acronym SWOT as keys and the corresponding words as values.

```
days = { 0: 'Zero', 1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five',
6: 'Six', 7: 'Seven', 8: 'Eight', 9: 'Nine'}
```

```
analysis = {'S': 'Strengths', 'W': 'Weaknesses', 'O': 'Opportunities',
'T': 'Threats'}
```

B. Suppose a dictionary object of Egyptian numerals is defined with the following statement:

```
>>> d = {'Stroke':1, 'Yoke':10, 'Rope':100, 'Lotus':1000,
'Finger':10000, 'Frog':100000}
```

What will Python print as the value of the following expressions?

a. `>>> len(d)` **6**

b. `>>> d['Rope']` **100**

c. `>>> d['Frog']` **100000**

d. `>>> 'Yoke' in d` **True**

e. `>>> 10000 in d` **False**

f. `>>> 'Panda' in d` **False**

C. Write an assignment statement that create dictionary named continents for the seven continents on the earth which are: `Asia, Africa, North America, South America, Europe, Oceania, Antarctica`. In this dictionary, the first two characters of the continent name is the key and the name of the continent is value.

```
continents = {"As":"Asia", "Af":"Africa", "No":"North America",
"So":"South America", "Eu":"Europe", "Oc":"Oceania", "An":"Antartica"}
```

Q. 6: Output analysis and functions

A. Suppose a variable s has been defined with this assignment statement:

```
>>> s = "Peace begins with a smile."
```

What will Python print for each of the following statements?

a. >>> print(s) **Peace begins with a smile.**

b. >>> print(len(s)) **26**

c. >>> print(s.split()) **['Peace', 'begins', 'with', 'a', 'smile.']**

d. >>> print(s.split()[2]) **'with'**

e. >>>print(s.split()[1].upper()) **BEGINS**

f. >>>import string

>>> print(s.strip(string.punctuation)) **Peace begins with a smile**

B. Write a function, *acronym*, that creates an acronym from the first letter of each long word in a list, where a long word is any word with more than three letters.

```
>>> acronym('operating system')
'OS'
>>> acronym('association for computing machinery')
'ACM'
```

```
def acronym(phrase):
    result = ''
    words = phrase.split()
    for w in words:
        if len(w) > 3:
            result += w.upper()[0]
    return result
```

Q. 7: (Random numbers, Classes and OOP)

A. (Pseudo-Random Numbers) Study the random number generator code here:

```
a = 4
c = 11
m = 23
x = 3 % m

def rand():
    global x
    x = (a * x + c) % m
    return x

for i in range(10):
    print (str(rand()) + " ", end="")
```

What 10 values will this code generate?

____ 0 11 9 1 15 2 19 18 14 21 _____

B. Create a class called `Worker`. *Worker* holds information on a factory worker in a company. The information includes the worker's full name, hourly rate, hours in a standard week and hours in an extended week.

A worker earns their normal hourly rate for the number of hours in a standard week. If they work more hours, for the extra hours, they earn 1.5 times their hourly rate. Finally, if they work beyond the number of hours in the extended week, any hours over that number are paid at 2 times the hourly rate.

The class must have an __init__ method to build the object given the worker's name, hourly rate, standard hours, and extended hours.

You must also write a `calculatePay()` method that takes the number of hours worked that week and returns the amount of pay in US dollars.

Example: If Joe Cool has a standard work week of 40 hours, an extended week of 50 hours, and wage of 18.50 per hour, his pay for 55 hours would be:

40 * 18.50 + 10 * 18.50 * 1.5 + 5 * 18.50 * 2 = 1202.50

So creating a *Worker* object to compute this would look like:

```
w = Worker ("Joe Cool", 40, 50, 18.50)
print(w.name + "earned $" + str(w.calculatePay(55))+" for 55 hours. ")
```

Write the class along with the constructor and the calculatePay() method.

```
class Worker:

        def __init__(self, name, regular, extended, rate):
            self.name = name;
            self.regular = regular;
            self.extended = extended;
            self.rate = rate;

         def calculatePay(self, hours):
            if hours > self.extended:
                high = hours - self.extended
                over = self.extended - self.regular
                base = self.regular
            elif hours > self.regular:
                over = hours - self.regular
                base = regular
            else:
                base = hours
                over = 0
                high = 0
            pay = (self.regular * self.rate) + (over * self.rate * 1.5) + (high * self.rate * 2);
            return pay
```

C. The class `Clock` simulates the tick-tack of a clock. An instance of this class contains the time, which is stored in the attributes `self.hours`, `self.minutes` and `self.seconds`. Complete the methods `tick` and `__str__` of class clock.

```
    Examples:
    >>> x = Clock(12,59,59)
    >>> print(x)
    12:59:59
    >>> x.tick()
    >>> print(x)
    13:00:00
    >>> x.tick()
    >>> print(x)
    13:00:01
    """
```

```
def __init__(self, hours, minutes, seconds):
      """
      The parameters hours, minutes and seconds have to be integers and must
      satisfy the following equations: 0 <= h < 24, 0 <= m < 60, 0 <= s <
      60. An exception is thrown if the values are outside range.
       """
        if type(hours) == int and 0 <= hours and hours < 24:
            self._hours = hours
        else:
            raise TypeError("Hours have to be integers between 0 and 23!")
        if type(minutes) == int and 0 <= minutes and minutes < 60:
            self.__minutes = minutes
        else:
            raise TypeError("Minutes have to be integers between 0 and 59!")
        if type(seconds) == int and 0 <= seconds and seconds < 60:
            self.__seconds = seconds
```

```
        else:
            raise TypeError("Seconds have to be integers between 0 and 59!")
def tick(self):
      """
      This method lets the clock "tick", this means that the internal time
      will be advanced by one second.
       """
             if self.__seconds == 59:
            self.__seconds = 0
            if self.__minutes == 59:
                self.__minutes = 0
                if self._hours == 23:
                    self._hours = 0
                else:
                    self._hours += 1
            else:
                self.__minutes += 1
        else:
            self.__seconds += 1


def __str__(self):
       """
      Prints the time in the format HH:MM:SS.
       """
return "{0:02d}:{1:02d}:{2:02d}".format(self._hours,
                                                self.__minutes,
                                                self.__seconds)
```

Q. 8: Files

Write a function that takes as an argument a filename and calculates the number of lines, words, and characters in the entire file. These calculated numbers should be returned as a tuple in the form of (number_lines, number_words, number_characters).

```
def wc(filename):
    nlines = nwords = nchars = 0
    for line in open(filename):
        nlines += 1
        nwords += len(line.split())
        nchars += len(line)
    return nlines, nwords, nchars
```