COMP1001 – Problem Solving in Information Technology
Final Examination (Solutions)

Please read the following instructions very carefully.

1. Before the commencement of the examinations
    a. Put away your phones, calculators, and other electronic devices.
    b. Put down your name, student ID and program name on the answer book.
    c. For the answer book:
        - Leave the first page empty.
        - Use the next four pages for Q1, the next four for Q2 and the next four to Q3.
    d. Login Blackboard (learn.polyu.edu.hk) and open the course page.
    e. Test also whether your PC has Python, IDLE, notepad, and 7-zip/winrar/winzip.
    f. Test the keyboard and mouse.
    g. If you have problems in (d)-(f), please raise your hand to seek help
2. About the questions and submissions
    a. You will find the softcopy of the exam paper in Blackboard in a few minutes after the commencement of the exam.
    b. Each question carries the same weight (4 marks).
    c. **Answer all 8 questions** and always attach succinct explanation.
    d. Write down your answers to Q1-Q3 in the answer book.
    e. Submit the *.py* files for Q4-Q8 to Blackboard. Name your files by *"your student ID"_Q4.py* for Q4 and so on. You should upload your *.py* file as soon as you finish one. There is no limit on the number of submissions.
    f. Some questions will take less time to answer, while some others more time.
3. During the examination
    a. You can use everything available in your PC and access everything under Blackboard and nothing else.
    b. You may use pencil/ball pen, eraser and ruler and nothing else.
    c. No communication with others in any form and no sharing of anything
    d. Bathroom policy: one at any time. Raise your hand and sign out before going. You also need to sign in after coming back.
    e. Raise your hand if you have any question about the examination paper, but we will not explain the questions to you.
4. At the end of the examination period.
    a. Pass your answer book towards the middle aisle in your respective room.
    b. Please logout your machine and clean up your table before leaving. You can keep the examination paper.

1. (**Binary representation of numbers**) Answer the following questions.
   a. [Weight = 0.4] Explain why 0.4_{ten} cannot be represented exactly in binary?
   b. [Weight = 0.3] How many different decimal numbers can be represented by $a_0.a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6}a_{-7}a_{-8}a_{-9}a_{-10}$, where $a_i = 0,1$ for $i = 0, -1, \ldots, -10$?
   c. [Weight = 0.3] For a binary number with 10 digits after the radix point (like the one in (b)), what is the precision of the decimal numbers that can be represented in terms of the number of digits after the radix point? For example, the precision of 0.123456_{ten} is 6.

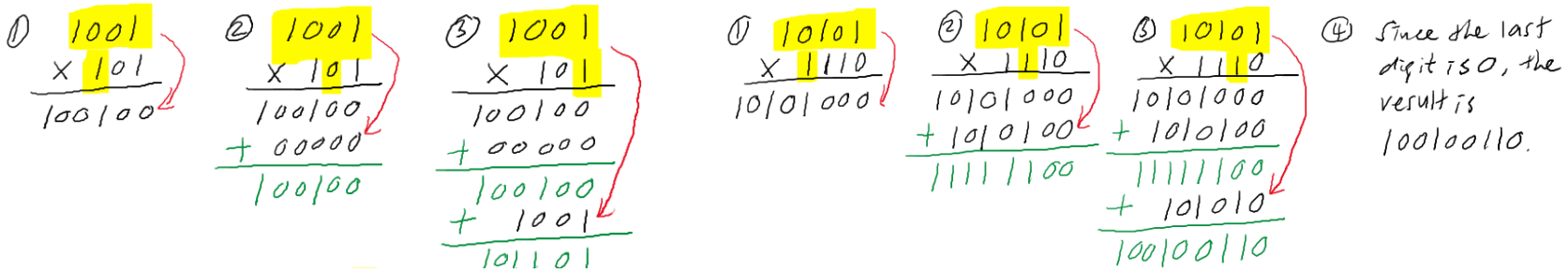
Solution:

a. [Weight = 0.4] By multiplying 2 and removing the integer part repeatedly, we have 0.4→0.8→0.6→0.2→0.4→ … (cycling back). Therefore, the numbers will be recycled in a loop, and as a result it cannot be represented by a finite number of bits.

b. [Weight = 0.3] There are 11 bits. Therefore, it can represent 2^{11} (2048) decimal numbers.

c. [Weight = 0.3] The precision is determined by the last bit (i.e., a_{-10}). Therefore, the precision is given by 2^{-10} (0.0009765625) which is 10.

2. (**Multiplication of two binary numbers**) In this question, you are going to explore the problem of multiplying two binary numbers. You are given the two functions below to tackle this problem. Consider the examples of $1001_{two} \times 101_{two}$ and $10101_{two} \times 1110_{two}$ below. Notice that the digits of the second number are considered one at a time. If it is 1, the first number is left shifted by the amount determined by the digit's position. There is no shifting for the last digit. These shifted binary numbers are then added to produce the result.




```
Function bitLeftShift(aBinaryNumber, n)
    Input:
    - aBinaryNumber: a binary number. As usual, the leftmost digit is the most significant bit.
    - n: the number of bits to be shifted and n >= 0.
    Output: return aBinaryNumber shifted left by appending n 0s to it.

Function PROC2(num1, num2)
    Input: num1 and num2 are two binary numbers. Their lengths do not have to be the same.
    Output: return the result of the binary addition of num1 and num2.
```

Write a function called *multiplyTwoBinary*() in pseudocode for multiplying two binary numbers.

```
Function multiplyTwoBinary(num1, num2)
    Input: two binary numbers
    Output: return the result of multiplying num1 and num2.
```

In terms of functions, you can use only *bitLeftShift*(), *PROC2*(), *and len*(*aBinaryNumber*) that returns the number of bits in *aBinaryNumber*. Of course, you could use *for* and *if* in the pseudocode. You could also use

- `for each i in [1,n]` for iterating *i* over 1, 2, …, *n* (*n* is included)
- `for each i in [n,1]` for iterating *i* over *n*, *n*–1, …, 0 (0 is included), and
- *aBinaryNumber*[*i*] for the *i*th digit.

Note that *aBinaryNumber*[0] is the most significant digit, and *aBinaryNumber*[*len*(*aBinaryNumber*)–1] is the least significant digit.

Solution:

```
Function multiplyTwoBinary(num1, num2)
   Input: two binary numbers
   Output: return the result of multiplying num1 and num2.
   sum ← 0
   for each digit d in [0,len(num2)-1]
      if num2[d] = 1
         sum ← PROC2(sum, bitLeftShift(num1,len(num2)-1-d))
   return sum
```

3. (**Approximation of e^x**) The exponential function e^x may be written as a Taylor series $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ Below is an implementation of two functions to approximate e^x.

```
def factorial(n):
# Input: n, a non-negative integer
# Output: Return n!
   if n == 0:
      return 1
   else:
      result = 1
      for i in range(1,n+1):
         result = result * i
      return result
```

```
def taylor_e(x,n):
# Input: x is a real number and
# n is a positive integer.
# Output: The first n+1 terms in the
# Taylor series of e^x
   sum = 1
   for i in range(1, n+1):
      sum = sum + x**i/factorial(i)
   return sum
```

a. [Weight = 0.3) What is the number of multiplications required for computing *factorial*(*n*) in terms of *n*?
b. [Weight = 0.3] What is the number of multiplications required for computing *taylor_e*(*x*,*n*) in terms of *n*? (useful formula: $1 + 2 + \ldots + n = n(n+1)/2$, and $1 + 2 + \ldots + n-1 = (n-1)n/2$). Also, *x***2 requires 1 multiplication, *x***3 requires 2 multiplications, and so on.
c. [Weight = 0.4] Implement the function in pseudocode that can achieve O(*n*) in time complexity.

Solution:

a. [Weight = 0.3] From the code, it is *n*.
b. [Weight = 0.3] There are two parts: (1) *x****i*: $0 + 1 + 2 + \ldots + n-1 = n(n-1)/2$ and (2) *factorial*(*n*): $n(n+1)/2$. Therefore, the total is $n(n-1)/2 + n(n+1)/2 = n^2$.
c. [Weight = 0.4]

```
def taylor_e(x,n):
    sum ← 1
    fact ← 1
    x_term ← 1
    for each i in [1, n]:
        fact ← fact * i
        x_term ← x_term * x
        sum ← sum + x_term/fact
    return sum
```

4. (**Two couples crossing the river**) In this question we consider a simplified version of the class project. There are only two couples crossing a river from the east to the west. Similar to before, we use 5-tuple to model the states of the problem: (position of the boat, position of the green husband, position of the green wife, position of the red husband, position of the red wife). From Content of Blackboard, you will find *threeCouples.py* which solves the class project problem.

   a. [Weight = 0.5] Modify *threeCouples.py* to *twoCouple.py* that solves the two-couple-river-crossing problem. You should remove the blue husband and blue wife from *threeCouples.py*.

   b. [Weight = 0.5] Note from (a) that some legal states that meet the constraints of the problem are not part of the solutions to the problem. What are they and why? Please answer them as Python comments at the top of *twoCouples.py*.

   The expected answer is

```
1 The red husband and red wife go from the east to the west.
2 The red husband goes from the west to the east.
3 The green husband and red husband go from the east to the west.
4 The green husband goes from the west to the east.
5 The green husband and green wife go from the east to the west.
```

   Solution: In the .py file

5. (**Keeping students' scores**) You may find *studentRecords.csv* and incomplete *sortedStudentRecords_student.py* from Content of Blackboard. The .csv file stores the last names of the students and the scores for the 10 assignments (the scores for assignments 8 and 9 are combined). The incomplete *.py* file takes *studentRecords.csv* and write a sorted order of their records to an output file called *sorted.csv*. The records are sorted based on the total scores of the 10 assignments in a non-decreasing order as you go down from the first line in the file (i.e., the record with the lowest total score will be at the first line in *sorted.csv*, and so on).

   The key idea of *sortedStudentRecords_student.py* is to sum up the scores and put it at the head of a list as a string. Since the total scores may take one or more digits, we could use a string method called *zfill*(*n*) to fill a string to *n* characters by prepending just enough 0s. For example, `"5".zfill(4)` will give `"0005"`. In our case you could use $n = 3$. By making all strings of scores to have the same length, we could simply use the *sort()* method to sort the records based on the total scores. Fill in the missing code to make this program work. Alternatively, you could write your own program from scratch. You will also find from Content of Blackboard the expected *sorted.csv*.

   Solution: In the *.py* file

6. (**Your overall test grade**) As you know, your overall test grade will be based on the best five questions out of the ten in the two tests. I told you that it is a better deal than the maximum of the two test marks. To convince you that I did not lie, in this question you write a program (no need to use function) to compute the % of increase in marks by using the best-5Q scheme over the best-test scheme. I have randomly generated a list of 10 integers each of which is in the range of [0,4] for the marks of the ten questions. The first five is for the first test, and the other half is for the second test. You could find the data in *data_bestFive.txt* from Content of Blackboard. Include them in your *.py* file.

   Your program should print out the minimum, maximum, and average of the % improvement in marks; and round the values to two decimal places. The expected answers are

   ```
   minimum = 0.0% maximum = 100.0% average = 36.12%
   ```

   The % improvement in marks is computed by (marks of the best-5Q – marks of the best-test) × 100 /marks of the best-test. For computing the average, you could use the built-in function *sum*(*list*) which returns the sum of the numbers in *list*, and *len*().

   Solution: In the *.py* file

7. (**Social network again**) We consider the friendship network problem again in Q3 of Assignment 7. Instead of generating the social network randomly, please use the one below.

   friends = {0: {9, 7}, 1: {11, 4, 13, 39}, 2: {32, 33, 34, 3, 36, 10, 11, 29}, 3: {2, 36, 37, 17, 19, 25, 26}, 4: {1, 36, 37, 8, 10, 20}, 5: {28, 31}, 6: set(), 7: {0, 38, 10, 14, 21, 22, 28}, 8: {19, 4, 23}, 9: {0, 28}, 10: {2, 4, 38, 7, 21}, 11: {1, 2, 35, 19, 24, 29}, 12: {24, 17, 34}, 13: {32, 1, 20, 36}, 14: {36, 31, 7}, 15: {22}, 16: {37, 17, 19, 26, 27}, 17: {32, 3, 35, 12, 16, 27, 29, 30}, 18: {32, 33, 37, 38, 20, 24, 26}, 19: {3, 38, 8, 11, 16, 25}, 20: {33, 4, 36, 13, 18, 25, 29, 30}, 21: {37, 38, 7, 10, 24}, 22: {36, 7, 15, 28, 29, 31}, 23: {8, 36}, 24: {11, 12, 18, 21, 30}, 25: {3, 39, 19, 20, 27, 31}, 26: {32, 33, 3, 16, 18}, 27: {38, 16, 17, 25, 28}, 28: {32, 5, 7, 9, 22, 27}, 29: {33, 2, 37, 11, 17, 20, 22, 30}, 30: {33, 17, 20, 24, 29}, 31: {25, 5, 22, 14}, 32: {2, 13, 17, 18, 26, 28}, 33: {2, 35, 18, 20, 26, 29, 30}, 34: {2, 35, 12}, 35: {33, 34, 11, 17}, 36: {2, 3, 4, 39, 13, 14, 20, 22, 23}, 37: {3, 4, 38, 16, 18, 21, 29}, 38: {37, 7, 10, 18, 19, 21, 27}, 39: {1, 36, 25}}

   Implement a function that accepts a person and the dictionary of all people; and returns a set of the person's friends and their friends. For example, it will return {7, 9, 10, 14, 21, 22, 28, 38} for person 0. It is because person 0 has friends of person 7 and person 9. In turn, person 7's friends are {0, 38, 10, 14, 21, 22, 28} and person 9's friends are {0, 28}. Therefore, the set of person 0's friends and their friends is {7, 9}∪{38, 10, 14, 21, 22, 28}∪{28}.

```
def friendsOfFriends(person,peopleDict):
    """
    Input: a person and a dictionary of all people.
    Output: Return a set of the person's friends of friends.
    """
```

   Include the statement below in your *.py* file.

```
print("The first person's friends of friends are: ",friendsOfFriends(0,friends))
```

   Solution: In the *.py* file

8. (**How many comparisons can we save?**) We have seen that the stack-based algorithm is much more efficient than the brute-force algorithm. We can compute the saving in comparisons by running the stack-based and brute-force algorithms separately. In this question, however, you are asked to implement a single function to return the number of comparisons saved by the stack-based method. This function will return the up periods as usual and the number of comparisons saved by the stack-based algorithm.

```
Function stockUp_saving(stock)
   Input: stock, a list of stock prices on a range of days starting from day 1.
   Output:
   - a list of up-periods for the days.
   - the total number of comparisons saved by the stack-based method
```

Include the statements below in your *.py* file for testing. The answer for this case should be 18.

```
stockPrice = [7, 12, 9, 6, 3, 9, 10, 11, 12]
result, saving = stockUp_saving(stockPrice)
print("The total number of saving is", str(saving)+'.')
```

To help you develop the code, we explain below why the answer is 18 for this stock price list.

- When computing day 7's up period, 6 comparisons are saved, because days 3, 4, and 5 are no longer in the stack.
- When computing day 8's up period, 8 comparisons are saved, because days 3, 4, 5 and 6 are no longer in the stack.
- When computing day 9's up period,
  - First of all, 10 comparisons are saved before comparing with day 2, because days 3, 4, 5, 6, and 7 are no longer in the stack.
  - Then, after comparing with day 2, 2 more comparisons are saved, because day 1 is not in the stack.

Therefore, 26 comparisons are saved. However, 8 comparisons are needed for the if-else statement. The total saving is therefore 26 – 8 = 18.

Solution: In the *.py* file

~~ END ~~