# COMP 1005/1405 – Summer 2017- Exam Practice Problems

Studying and solving these problems should give you an idea of what type and level of difficulty of problems to expect on the exam. You should still consider reviewing tutorial, assignment, and in-class problems as well.

## Problem 1

You should save the solution to this problem in a file called **mindistance.py** if you wish to use the tester program. Write a function called **mindistance** that takes a single 2-dimensional list argument as input. Each of the items in this argument list will be a list with 2 numbers representing the x and y location of a point in 2D space. The function must return the 2-element list representing the x and y location of the point that is closest to the origin (x=0, y=0). You can calculate the distance between a point (x,y) and the origin (0,0) with the equation:

$$Distance = \sqrt{x^2 + y^2}$$

## Problem 2

You should save the solution to this problem in a file called **setoperations.py** if you wish to use the tester program. Write two functions called **intersection** and **union**. Each of these two functions should accept two list arguments (called A and B here) as input and output a new list representing the intersection/union of the two lists A and B. The intersection of the lists is defined as all of the unique elements that are present in both A and B, while the union of the lists is defined as all of the unique elements that are present in either A or B.

## Problem 3

You should save the solution to this problem in a file called **maxk.py** if you wish to use the tester program. Write a function called **maxk** which should accept a list (mylist) argument and an integer (k) argument as input. The function should return a new list containing the k largest values from mylist. You may not modify the original list, but can make a copy of the original list using the slicing function and modify the copy. You can assume that k will always be less than the length of the list.

## Problem 4

You should save the solution to this problem in a file called **maxprofit.py** if you wish to use the tester program. Write a function called **maxprofit** that accepts a single list argument as input (referred to as mylist in this description). The function should determine the maximum profit that can be made by 'buying' a stock at one index in mylist and 'selling' it at a later index. In this case, you can calculate the profit of buying at index x and selling at index y as mylist[y] – mylist[x]. Note that the selling index must come after the buying index. Your function must return a list with the following 3 pieces of information: the buy index, the sell index, and the maximum profit. You may assume

that at least one combination of buy/sell prices that exist produce a positive profit. You can also assume that there will be only one maximum combination (i.e., no ties exist).

**Problem 5**

You should save the solution to this problem in a file called **sequencelength.py** if you wish to use the tester program. Write a function called **sequencelength** that accepts a single list argument. Your function must return the longest increasing sequence of numbers present in the argument list. An increasing sequence of numbers is one in which each number is larger than the one before it. For example, [4, 8, 13, 14, 21] is an increasing sequence, but [4, 8, 13, **12**, 21] is not.

**Problem 6**

You should save the solution to this problem in a file called **binarysearch.py** if you wish to use the tester program (note: the test program tests both iterative and recursive functions, in that order, so you may have to scroll up for iterative results). Implement an iterative (i.e., non-recursive) binary search function called **iterativesearch** and a recursive binary search function called **recursivesearch**. Each of these functions should accepts a list and an item as arguments and return any index where the item occurs in the list. If item does not occur in the list, the functions should return -1.

**Problem 7**

You should save the solution to this problem in a file called **removestring.py** if you wish to use the tester program. Write a function called **removestring** that takes two input arguments: a string called **text** and a string called **remove**. Your function must return the text string with each occurrence of the string remove taken out. You may not use the built-in string replace method, but can use other string methods.

**Problem 8**

Write a program that will repeatedly ask the user to enter strings, one at a time. If the user enters the string “print”, your program should print out the string that has been entered most frequently by the user. If the user enters the string “quit”, your program should quit. If the user entered any other string, your program must update its state in whatever way necessary to ensure the “print” request can be answered in O(1) time (i.e., it cannot iterate over the entire list of words that have been entered). Sample output for this question is available below:

**Sample Output for Word Frequency Problem**

Enter a string: hi
Enter a string: print
hi has been entered 1 times.
Enter a string: hi

Enter a string: print
hi has been entered 2 times.
Enter a string: hello
Enter a string: hello
Enter a string: hello
Enter a string: print
hello has been entered 3 times.
Enter a string: hi
Enter a string: hi
Enter a string: print
hi has been entered 4 times.
Enter a string: quit

Enter a string: a
Enter a string: a
Enter a string: b
Enter a string: b
Enter a string: c
Enter a string: c
Enter a string: a
Enter a string: print
a has been entered 3 times.
Enter a string: b
Enter a string: b
Enter a string: print
b has been entered 4 times.
Enter a string: c
Enter a string: print
b has been entered 4 times.
Enter a string: c
Enter a string: c
Enter a string: print
c has been entered 5 times.
Enter a string: quit