Carleton University
School of Computer Science
Summer 2019
COMP 1005/1405A Final Exam

Notes:

1. The exam will last 3 hours.
2. The exam contains 5 pages, including this cover page.
3. All answers should be written on the computer in the appropriate files.
4. Test files are included for each question. You should run these files to verify that your solutions are working correctly prior to finishing the exam.
5. There will be reference PDFs within the exam directory that include: lists, dictionaries, and strings, as well as the random and math modules.
6. If you are uncertain of a requirement, state your assumption in a comment in your answer.
7. **You may use a code editor and/or the command line to write and test your solutions. You may not use other programs (web browser, etc.).**
8. **You may not open or otherwise access any material that is not included within the exam directory.**

Instructions:

1. Follow the instructions provided on the projector screen.
2. Each question has an associated Python file included within the exam directory. Place your code for each question in the appropriate file.
3. The exam is divided into 3 parts consisting of 3 questions each: Part #1 (Problems 1-3), Part #2 (Problems 4-6), and Part #3 (Problems 7-9).
4. Each problem will be marked out of 10.
5. Your mark for each part of the exam will be calculated using the best two marks on problems within that section, for a total of 60 marks. For example, if you get 8/10, 10/10, and 3/10 on Problems #1-3, your mark for Part #1 of the exam will be 18/20.
6. You may use any standard Python code you want, unless the question specifically mentions that you may not.
7. **Before logging out or leaving the exam, it is your responsibility to verify that your exam has been uploaded successfully. You can check your most recent upload by extracting the firstname-lastname.zip file in the COMP-Exam/archives directory. This zip file contains your most recently uploaded exam files.**

Problem 1

Write a function called **nextprime** that accepts a single integer input argument *n*. The function must return the lowest prime number that is strictly larger than *n*. You may assume that *n* will always be a positive integer. For the purposes of this question, a prime number is any positive integer that cannot be evenly divided by any positive integers other than itself and 1. An alternative definition is any positive integer that cannot be formed by multiplying two smaller positive integer numbers. **You must write the code to check primality of the numbers yourself and may not import any additional modules**.

Examples:
nextprime(1) → 2
nextprime(2) → 3
nextprime(44) → 47
nextprime(47) → 53

Problem 2

Write a function called **minarea** that takes a single 2D list argument as input. Each of the items in this argument list will be a list with 2 numbers, representing the base length and height of a triangle. The function must return the 2-element list containing the base length and height of the triangle with the smallest area. Note that the area of a triangle is calculated as *0.5 x base length x height*. **You may not use additional modules or the built-in min/max/sort functions**.

Examples:
minarea([[20, 11], [7, 11], [25, 1], [24, 22]]) → [25, 1]
minarea([[7, 23], [15, 5], [11, 4], [18, 18], [2, 5]]) → [2, 5]
minarea([[3, 23], [3, 9], [7, 22], [18, 15], [14, 3], [14, 19]]) → [3, 9]

Problem 3

Write a function called **mode** that takes a single list of integers as an argument. The function must return the mode of the given list (i.e., the number that occurs most frequently). You can assume that there will be a single mode (no ties). This function should run in O(*n*) time, where *n* represents the length of the list. Solutions running in longer than O(*n*) time will be penalized. **You may not use any additional modules or the built-in min/max/sort/mode functions**.

Examples:
mode([8, 3, **2**, 8, 1, 7, **2**, 0, 4, **2**, 5]) → 2
mode([**0**]) → 0
mode([4, **1**, 4, 4, **1**, **1**, 4, **1**, **1**, 4, **1**]) → 1

Problem 4

The Jaccard index is a measure of similarity between two sets, computed as the size of the intersection of the sets divided by the size of the union of the sets. In other words, the Jaccard index of two sets A and B can be calculated as:

$$\frac{\#\text{ unique elements that are present in both A and B}}{\#\text{ unique elements present in A or B}}$$

Write a function called **jaccard** that computes the Jaccard index for two argument lists. **At the start of your code, in comments, provide an analysis of the runtime complexity of your solution. Your analysis should include the worst-case runtime complexity in big-O notation, as well as justification for your answer (assume that A and B have *n* elements in general)**. You can assume that the lists will contain only integer values. Note: don't forget that only unique elements should be counted in the above equation, any duplicate elements should be ignored. **You may not use list comprehensions, additional modules, or the built-in union/intersection functions**.

Examples:
A = [1, 2, 3, 4, 5] B = [2, 4, 6, 8]
intersection(A,B) → [2, 4]
union(A,B) → [1, 2, 3, 4, 5, 6, 8]
jaccard(A,B) → 2 / 7 → 0.2857

A = [0, 6, 1, 7] B = [1, 7, 6, 3]
intersection(A,B) → [6, 1, 7]
union(A,B) → [0, 6, 1, 7, 3]
jaccard(A,B) → 3 / 5 → 0.6

Problem 5

Write a function called **kthsmallest** that accepts a list (L) and an integer (K) as input arguments. The function must return the K-th smallest number in the list L. So, if K=1, the function returns the smallest number; if K=2, the function returns the second smallest number, etc.. You can assume that 1 <= K <= len(L). **You cannot: modify the original list, use list comprehensions, use additional modules, or use any of the built-in min/max/sort functions**.

Examples:
kthsmallest([9, 3, 1, 4, 6, 2, 8, 7, 5], 1) → 1
kthsmallest([9, 3, 1, 4, 6, 2, 8, 7, 5], 3) → 3
kthsmallest([9, 3, 1, 4, 6, 2, 8, 7, 5], 9) → 9
kthsmallest([2, 5, 2, 1, 5, 0, 2, 5, 4], 1) → 0
kthsmallest([2, 5, 2, 1, 5, 0, 2, 5, 4], 3) → 2
kthsmallest([2, 5, 2, 1, 5, 0, 2, 5, 4], 4) → 2

Problem 6

Write a recursive function called **sumdigits** that takes a single positive integer argument. The function must return the sum of the digits of the input argument (that is, the value of each digit in the number added together). **The function must operate recursively, without the use of loops. You may not use additional modules**.

Examples:
sumdigits(83991) → 30
sumdigits(100) → 1
sumdigits(755) → 17

Problem 7

Write a function called **sortvolumes** that takes a single 2D list argument. Each element in this argument list will be a 3-element list representing the length, width, and height of a box. The function must return a new 2D list containing the 3-element lists sorted in ascending order from smallest to largest volume. Note: the volume of a box can be calculated as *width x height x length*. **You may not use additional modules or use any of the built-in min/max/sort functions**.

Example:
sortvolumes([[3,5,1], [2, 2, 3], [1, 4, 2]) → [[1, 4, 2], [2, 2, 3], [3,5,1]]

Problem 8

Write a function called **xmostfrequent** that takes one string argument (S) and one integer argument (X). The function must return a list of the X most frequent words in the string S. Additionally, the returned list must have the X most frequent words sorted from highest to lowest frequency. You may assume all words are separated by spaces and that there is no additional punctuation. Part of the marks for this question will be based on minimizing the runtime complexity of your solution, relative to the number of words in the given string (N). Assume that X is small enough to be considered a constant (e.g., we know it will always be 10 at the most). In this case, it should be possible to solve this problem in O(N) time in the worst-case. **You may not use additional modules or use any of the built-in min/max/sort functions**.

Examples:
xmostfrequent(“peach apple peach orange apple peach”, 2) → [“peach”, “apple”]
xmostfrequent(“pea apple pea orange pea orange”, 2) → [“pea”, “orange”]
xmostfrequent(“b a b a c c a c a d b c c”, 3) → [“c”, “a”, “b”]

Problem 9

Write a function called **isvalidseries** that accepts a list of positive integers (L), an integer (X), and an integer (SUM) as arguments. A series of integers will be considered valid if there is no sequence of X consecutive numbers in the list that sum to larger than SUM. The **isvalidseries** function must return True if the given list is considered valid for the given X/SUM values, and False otherwise.

Examples (bold numbers show invalid consecutive numbers that exceed SUM):

isvalidseries([**8, 4, 8**, 3, 1, 2, 7, 9], 3, 19) → False
isvalidseries([2, 4, 8, 3, 1, 2, 7, 9], 3, 19) → True
isvalidseries([2, 4, 8, 3, 1, **2, 7, 9**], 3, 16) → False
isvalidseries([5,5,5,5,5,5,5,5], 3, 19) → True
isvalidseries([2,2,**5**,**5**,**5**,5,5], 4, 19) → False
isvalidseries([5,5,5,5,5,5,5,5], 4, 20) → True

Finishing the Exam

Within the COMP-Exam directory, there is an archives directory that contains several zip files. The zip file titled *yourfirstname-yourlastname*.zip contains the most recent upload of your work. Extract the files from this zip and open them in a code editor to verify that the files contain all your solutions in their final state. Once you have verified the contents, you can sign out with a proctor. If you believe there is an error with your upload, confirm with a proctor before logging out.