Carleton University
School of Computer Science
Fall 2019
COMP 1405B 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. Note that succeeding on these test files does not necessarily indicate that your solution is correct.
5. There will be reference PDFs within the exam directory for 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 Final-Exam/archives directory. This zip file contains your most recent exam upload.**

Problem 1

Write a function called **median** that takes a single unsorted list of integers as an argument. The function must return the median of the given list. If there is no single median value, return the average of the two middle values. 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 the input list has *n* elements in general). **You may not use any additional modules or the built-in min/max/sort functions.**

Examples:
median([10, 1]) →5.5
median([20, 14, 16, 18, 0, 5]) →15
median([14]) → 14
median([10, 11, 14]) → 11

Problem 2

Write a function called **largestbox** that takes a single 2D list argument as input. Each of the items in this argument list will be a list with 3 numbers, representing the length, width, and height of a box. The function must return the 3-element list containing the length, width, and height of the box with the largest volume. Note that the volume of a box is calculated as *length x width x height*. You may assume that the input list will have at least one box. **For full marks, your solution must have a worst-case runtime complexity of O(n) or better. You may not use additional modules or the built-in min/max/sort functions**.

Examples:
largestbox([[2, 10, 2], [6, 4, 6], [7, 1, 8], [7, 10, 10]]) → [7, 10, 10]
largestbox ([[10, 9, 9], [7, 9, 1], [8, 9, 9], [1, 5, 10]]) → [10, 9, 9]
largestbox ([[2, 10, 7], [9, 8, 9], [4, 5, 8], [4, 4, 1]]) → [9, 8, 9]

Problem 3

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). **For full marks, your solution must have a worst-case complexity of O(n) or better.** You can assume that the lists will contain only integer values. **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 4

Write a function called **permutation_palindrome** that takes a single string argument. The function must return True if a permutation (i.e., any arrangement) of the given string is a valid palindrome (a string that is identical when spelled forwards or backwards). In other words, the function must return True if the characters in the given string can be rearranged to form a palindrome. Otherwise, the function must return False. Your function should ignore space characters within the string. Hint: this should be possible to do in O(n) time – think about what properties a palindrome string must have and don't overcomplicate it.

Examples:
permutation_palindrome("dog") → False
permutation_palindrome("car race") → True (racecar)
permutation_palindrome("acct oat") → True (tacocat)
Note: your function should only return True or False

Problem 5

Write a **recursive function** called **isprime** that accepts a single integer input argument. The function must return True to indicate the given number is prime (only evenly divisible by 1 and itself) or False to indicate the number is not prime. **Your function must operate recursively and cannot contain any for/while loops**. Note that you may add additional helper functions to your file. You may assume that the input argument will be a positive integer that is 1 or greater.

Examples:
isprime(1) → True
isprime(2) → True
isprime(3) → True
isprime(4) → False

Problem 6

Write a function called **largest_distance** that takes a single 2D list argument as input. Each element in this 2D list will be a 2-element list containing the X and Y coordinates of a location. The function must return the furthest Manhattan distance between any two of the locations contained in the list. In comments at the start of your solution, write an analysis of the runtime complexity of your solution. Note that the Manhattan distance between two points (x1, y1) and (x2, y2) can be calculated using the equation below, where |x| represents the absolute value of x. That is, the magnitude of x regardless of sign (e.g., |5| = 5 and |-5| = 5)

$$manhattan\_distance = |x1 - x2| + |y1 - y2|$$

Problem 7

Write a function called **sorted_students** that takes a single string input argument representing a filename. You can assume each line of the file specified by the input argument will have the pattern:

Student_Name,Test1_Grade,Test2_Grade,Test3_Grade,Exam_Grade

The student name will be a string and each grade will be a number. The function must return a list of student names sorted from the student with the lowest final grade to the student with the highest final grade. To calculate the final grade, use the following weighting: test #1=20%, test #2=20%, test #3=20%, exam=40%. **You may only read the contents of the file one time. You may not use additional modules or use any of the built-in min/max/sort functions**.

Problem 8

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. **To receive full marks for this question, your solution must have a worst-case runtime complexity of O(n), where n is the size of the list of integers. Up to half of the marks will be awarded for solutions with a runtime complexity larger than O(n).**

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**,5,5], 4, 19) → False
isvalidseries([5,5,5,5,5,5,5,5], 4, 20) → True

Problem 9

Write a function called **validate_brackets** that accepts a single string argument. The function must return True to indicate the string contains valid bracketing and False if the brackets are invalid. A string is considered to have valid brackets if there are an equal number of opening/closing brackets and each closing bracket – i.e.,), }, or] – matches the most recent, unmatched opening bracket – i.e., (, {, or [. The only types of brackets that your program must consider are round brackets ( ), square brackets [ ], and brace brackets { }. The string may contain other characters, which your function should ignore.

Examples:

validate_brackets("(({ }))") → True
validate_brackets("{ [{ ( ) }] }") → True
validate_brackets("(){}[]") → True
validate_brackets("((})") → False
validate_brackets("x = int(input('prompt: '))") → True
validate_brackets("({ (( ))))") → False

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, sign out with a proctor. If you believe there is an error with your upload, confirm with a proctor before logging out.