CSCI-SHU 210 Data Structures

Assignment 2 Analysis of Algorithms

For this assignment, the correctness of your program's output can be graded by Gradescope.

However, your program's runtime will be manually checked and graded after due. (Which is about 64 points out of 100)

** Question 5 is also manually graded. You can submit it on Gradescope together with other files.

Problem 1 Merge Generator (24 points)

Write a merge_generator(I_1, I_2, I_3) function that takes three iterable objects and merges them alternately, once one runs out it continues to merge the other two, and lastly yields the remaining elements in the longest iterable.

- Your algorithm must take **O(len(I_1) + len(I_2) + len(I_3)) time**. (8pts for the runtime)
- You should not store all the data. In other words, the **memory complexity would be O(1)** (8pts for the space complexity)

For example, it should work as follows:

Example 1:

```
all = merge_generator( range(3), range(95,98), range(-3,0) )
input: next(all), next(all), next(all), next(all), next(all), next(all), next(all), next(all), next(all)
Return: 0, 95, -3, 1, 96, -2, 2, 97, -1
```

Example 2:

```
all = merge_generator( range(1), range(95,98), range(-2,0) )
input: next(all), next(all), next(all), next(all), next(all), next(all)
Return: 0, 95, -2, 96, -1, 97
```

Example 3:

```
all = merge_generator( range(4), range(95,96), range(-2,0) )
input: next(all), next(all), next(all), next(all), next(all), next(all), next(all)
Return: 0, 95, -2, 1, -1, 2, 3
```

Problem 2 Check if Sorted Without Using Sort (20 points)

Dr X claims to have an algorithm that takes an input sequence *S* and produces an output sequence *T* that is a sorting of the *n* elements in S. However, you are in doubt about its correctness. Write a python function is_sorted(T, S), that tests in **O(nlogn) time** in the worst case if *T* is a sorting of *S* (8pts for the runtime).

- Assume both *S* and *T* are lists of integers, and they have the same length n
- Assume that there are no duplicated elements in S or T.
- No sorting functions are allowed in your solution.
- Hint: write your own binary search function.

Example 1:

```
Input: is_sorted([1, 2, 3, 4, 5], [2, 4, 3, 1, 5])
Return: True
```

Example 2:

```
Input: is_sorted([1, 2, 3, 5, 9], [2, 4, 3, 1, 5])
Return: False
```

Example 3:

```
Input: is_sorted([1, 2, 3, 5, 4], [2, 4, 3, 1, 5])
Return: False
```

Example 4:

```
Input: is_sorted([5, 4, 3, 2, 1], [2, 4, 3, 1, 5])
Return: True
```

Problem 3 All Possible Sum Finder (28 points)

You are given two Python lists *A* and *B* both of length *n*, each storing integers **in increasing order**. Given an integer ***m***, implement a Python function find_sum_m(A, B, m) such that it returns **all** pairs of elements *(a,b)*: a ∈ *A* and b ∈ *B*, such that $a + b = m$, as a python list of tuples.

- If there are duplicates of $a \in A$ that all match to one (or duplicates of) $b \in B$, then return one pair (a,b) (see example below):
- If there is no such pair, then return an empty list.
- Your program must have **runtime complexity: O(n)** in the worst case (8pts for the runtime),
- and **memory complexity: O(1)** (8pts for the memory complexity).

Example 1:

A = [-1, 4, 5, 6, 8, 10, 12], B = [0, 1, 2, 4, 9, 10, 20]
Input: find_sum_m (A, B, 14)
Return: [(4, 10), (5, 9), (10, 4), (12, 2)]

Example 2:

A = [-1, 4, 5, 6, 8], B = [0, 1, 2, 4, 10]
Input: find_sum_m (A, B, 100)
Return: []

Example 3:

A = [1, 2, 2, 3, 5], B = [1, 98, 98, 99, 99]
Input: find_sum_m (A, B, 100)
Return: [(1, 99), (2, 98)]

Problem 4: Logarithm Calculator (28 points)

If $m^k = n$ for integers m, $n > 1$, and $k \geq 0$, we say that k is the logarithm base m of n. In general, this logarithm need not necessarily be an integer. Let us define the **integer logarithm base** *m of n* to be the greatest integer $k \geq 0$, such that $m^k \leq n$, and denote it by $k = lg_m n$. In the following exercise, we will calculate the integer logarithm function.

The only arithmetical operations you may use are + and *. When analyzing the time complexity of these algorithms, you are to count arithmetical operations only, assuming that every operation takes a single time unit.

a) Write a python function LG1(m,n) with input integers m, n > 1, that calculates $lg_m n$ by repeatedly calculating the powers $m^0, m^1, \ldots, m^k$, until a number k is found satisfying $m^k \leq n < m^{k+1}$.

There is no runtime complexity limit for this question. However, you should analyze your implementation's runtime complexity by yourself. Write your program's big ***O*** complexity **within your .py file in term of *k***. (As comment, tightest big O in worst case) (8pts for the runtime),

b) It is well known that each positive integer k can be written uniquely as a sum of integer powers of 2, i.e., in the form $k = 2^{l_1} + 2^{l_2} + \cdots + 2^{l_j}$, where $l_1 > l_2 > \cdots > l_j \geq 0$. For example,

$$12 = 2^3 + 2^2$$
$$31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

Hence, $m^k = m^{2^{l_1}} \cdot m^{2^{l_2}} \cdots m^{2^{l_j}}$, , and if we need to calculate $k = lg_m n$, it is enough to find the appropriate exponents $l_1, l_2, \ldots, l_j$.

Design an iterative (i.e., non-recursive) algorithm LG2(m,n) to calculate $lg_m n$ by first finding an integer l_1 satisfying $m^{2^{l_1}} \leq n < m^{2^{(l_1+1)}}$, then finding an integer $l_2 < l_1$, satisfying $m^{2^{l_1}} \cdot m^{2^{l_2}} \leq n < m^{2^{l_1}} \cdot m^{2^{(l_2+1)}}$, and so on.

- Your program must have **runtime complexity: O(logk)** in the worst case (8pts for the runtime), in which $k = logn$
- and **memory complexity: O(logk)** (8pts for the memory complexity).

Problem 5 Why is $O(n^2)$ faster than O(nlogn) sometimes? (8 points)

Al and Bob are arguing about their algorithms. Al claims his O(nlogn)-time method is always faster than Bob's $O(n^2)$-time method. To settle the issue, they perform a set of experiments. To Al's dismay, they find that if $n < 100$, the $O(n^2)$-time algorithm runs faster, and only when $n \geq 100$, O(nlog n)-time one runs faster. Explain how this is possible.

Important:

- You should submit a .txt file for this question.
- If you are submitting this question on gradescope, simply upload the text file, the text file will not get auto graded, TA will manually grade your text file after the deadline.