CSCI-SHU 210 Data Structures

Homework Assignment1 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 40 points out of 100)

** Question 6 is also manually graded. You can submit it on Gradescope together with other files.

Question 1 (Merge):

Write a **`merge(I1,I2)`** function that takes two iterable objects and merges them alternately, once one runs out it continues from the other. **Your algorithm must take O(len(I1) + len(I2)) time**. **(8pts for the runtime)** For example, it should work as follows:

```
print([i for i in merge(range(5),range(100,105))])
print([i for i in merge(range(5),range(100,101))])
print([i for i in merge(range(1),range(100,105))])
```

should output:

```
[0, 100, 1, 101, 2, 102, 3, 103, 4, 104]
[0, 100, 1, 2, 3, 4]
[0, 100, 101, 102, 103, 104]
```

Question 2 (Largest Ten):

Implement function, `largest_ten(list1)` for finding the **ten largest elements** in a list of size n.

```
Input: largest_ten([9,8,6,4,22,68,96,212,52,12,6,8,99,128])
Returns: [212, 128, 99, 96, 68, 52, 22, 12, 9, 8]   # Order
doesn't matter.
```

Important:

- For this question, you should avoid modifying the original sequence.
- For this question, you should avoid copying the entire original sequence.
- You can assume input list size is always greater than 10.
- You can assume list1 contains only integers.
- Your program must take O(N) time. Where N is the number of elements for the input list. **(8pts)**

Question 3 (Has_duplicate):

Implement function, `has_duplicate(list1)`, that determines whether list1 contains duplicate values.

```
Input: has_duplicate([1, 3, 6, 2, 4])
Returns: False
Input: has_duplicate([1, 3, 6, 2, 4, 3])
Returns: True
```

Important:

- You can assume list1 contains only integers.
- 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. (As comment, tightest big O) **(8pts)**

Question 4 (MinMax):

Implement function, **`minmax(list1)`**, that finds both the minimum and maximum of N numbers **using fewer than 3n/2 comparisons**.

Your function should return a tuple of two integers, the first integer is the minimum, the second integer is the maximum.

(Hint1: First, construct a group of candidate minimums and a group of candidate maximums.).

```
Input: minmax([150, 24, 79, 50, 98, 88, 345, 3])
Returns: (3, 345)
Input: minmax([678, 227, 764, 37, 956, 982, 118, 212, 177,
597, 519, 968, 866, 121, 771, 343, 561, 100])
Returns: (37, 982)
```

Important:

- For simplicity, input list sizes are even only.
- You can assume input list1 contains only integers.
- Number of comparisons must be less than 3N/2 times, where N is the number of elements in **`list1`**. **(8pts)**

Question 5 (Three-Way Disjoint problem):

Given three sets of items, A, B, and C, they are **Three-Way Set Disjoint** if there is no element common to all three sets, i.e., there exists no x such that x is in A, B, and C. In the text book, two solutions of **Three-Way Set Disjointness** is described which run time complexity is O(n^3) and O(n^2).

We will assume that no individual sequence contains duplicate values.

Implement an algorithm (Python Code) that solves the **Three Way Set Disjoint** problem using **O(nlogn)** time. (Hint: Use O(nlogn) sorting algorithm). **(8pts for the O(nlogn) runtime)**

```
l1 = [1,2,3,4,5]
l2 = [6,7,8,9,10,11,12]
l3 = [5,13,14,15,16]
l4 = [5,6,7,8,9,10,11]

Input1:three_way_disjoint(l1,l2,l3)  # No common element exist in
                                       all l1, l2, and l3.
Returns: True

Input2:three_way_disjoint(l1,l4,l3)) # 5 exists in l1, l3 and l4.
Returns: False
```

Question 6 (Why is O(n^2) faster than O(nlogn) sometimes?):

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(n log 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.