CSCI-SHU 210 Data Structures

Assignment2 Complexity

Problem 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 should take O(n) time. 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]
```

Problem 2 Largest Ten

Implement an efficient algorithm (Python code) for finding the **ten largest elements** in a sequence of size n. What is the running time of your algorithm? Don't make any changes to the original input sequence.

Problem 3 Missing number

List S contains n – 1 unique integers in the range [0, n – 1], that is, there is one number from this range that is not in S. Implement an O(n)-time algorithm (Python code) for finding that number. You are only allowed to use O(1) additional space besides the List S itself.

Problem 4 Three-Way Set Disjointness 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).

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

Problem 5 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 ≥ 100, O(n log n)-time one runs faster. Explain how this is possible.

Problem 6 MinMax

Implement an algorithm (Python code) for finding both the minimum and maximum of n numbers using fewer than 3n/2 comparisons. Show that total number of comparisons is less than or equal to 3n/2. For simplicity, list sizes are even only.

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