**Fall 2026 COP 3503H Exam #1 Solutions**
**Date: 10/8/2026**

1) (12 pts) Write a function that takes in an ArrayList of String, not necessarily distinct, and returns a HashMap<String,Integer> that stores a mapping between each **unique string** in the array list and a distinct integer identifier starting at 0. In addition, the identifiers should be assigned in alphabetical order. So if the original list stored ["pear", "orange", "pear", "apple", "strawberry"] then the map should be: "apple" → 0, "orange" → 1, "pear" → 2, "strawberry" → 3.

```
public static HashMap<String,Integer> makeMap(ArrayList<String> list) {

    TreeSet<String> ts = new TreeSet<String>();  // 1 pt
    for (String s: list) ts.add(s);              // 2 pts

    HashMap<String,Integer> res = new HashMap<String,Integer>();// 2 pts
    int idx = 0; // 1 pt
    while (ts.size() > 0)                    // 1 pt
        res.put(ts.pollFirst(), idx++);      // 4 pts

    return res;                              // 1 pt
}
```

**Grading: Note, there are many solutions, so map the points accordingly.**

2) (8 pts) What is the minimum number of comparisons that are necessary to sort 6 items using a comparison sort? (Note: The answer is a positive integer.)

6 items can be ordered in 6! ways 6! = 720. A single comparison can distinguish between at most 2 separate lists. More generally k comparisons can distinguish between at most $2^k$ different lists. It follows that we desire the minimum integer k for which $2^k > 720$. Since $2^9 = 512$ and $2^{10} = 1024$, the desired answer is **10.**

**Grading: 3 pts for 6!, 1 point for 720, 2 pts for $2^k$, 2 pts for the final answer.**

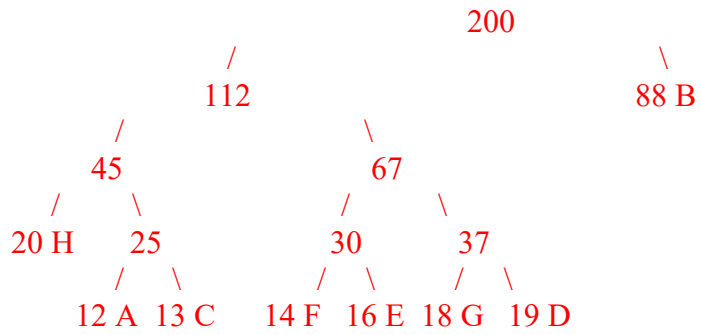
**Credit for 6 lg 6 is 1 point out of 8**

3) (12 pts) Use the Huffman Coding algorithm to build a Huffman tree for the data below and then compute the number of bits the Huffman code saves. Assume the original encoding uses 3 bits per character:

| Letter/Character | Frequency | Assigned Bit Code |
|---|---|---|
| A | 12 | 0010 |
| B | 88 | 1 |
| C | 13 | 0011 |
| D | 19 | 0111 |
| E | 16 | 0101 |
| F | 14 | 0100 |
| G | 18 | 0110 |
| H | 20 | 000 |

Draw the process of building the tree, showing the substeps that are necessary for you to build the correct final tree and put a box around the final tree below. After building the tree, put the corresponding bit code for each letter in the last column of the chart above.

One possible final tree:




The codes above represent 0 for going left and 1 for going right.

There are many possible correct answers. These nodes have to be siblings in any correct answer: (A, C), (F, E), (G, D), (H, 25), (30, 37), (45, 67) and (112, 88). Every pair of siblings can be in either order so there's 128 possible trees and 128 possible Huffman codes.

Bits saved = 2 x 88 – 1 x (12 + 13 + 19 + 16 + 14 + 18)
= 176 – 92
= **84**

**Note: When many of the characters appear a similar number of times, the number of bits saved tends to be less...**

**Grading: 8 points for the tree, 4 pts for the codes being consistent with the tree, grader determines partial as they see fit.**

4) (12 pts) Determine the following sum, in terms of x, assuming that |x| < 1:

$$1 + 3x + 5x^2 + 7x^3 + 9x^4 + \ldots$$

(Note: Formally, the sum to find is $\sum_{i=0}^{\infty}(2i + 1)x^i$.) Put a box around your final result.

Let S equal the sum:

$$\begin{aligned} S &= 1 + 3x + 5x^2 + 7x^3 + 9x^4 + \ldots \\ xS &= \quad x + 3x^2 + 5x^2 + 7x^4 + \ldots \\ \text{---} & \\ S - xS &= 1 + 2x + 2x^2 + 2x^3 + 2x^4 + \ldots \end{aligned}$$

$S(1 - x) = 1 + 2x(1 + x + x^2 + x^3 + \ldots)$

$$S(1 - x) = 1 + 2x\left(\frac{1}{1-x}\right)$$

$$S(1 - x) = \frac{1-x}{1-x} + \left(\frac{2x}{1-x}\right)$$

$$S(1 - x) = \frac{1+x}{1-x}$$

$$S = \frac{1+x}{(1-x)^2}$$

**Grading: Writing down sum and multiplying it by x – 2 pts**
**Subtracting xS from S – 2 pts**
**Factoring out 2x or something similar – 2 pts**
**Correctly plugging in formula for infinite geometric sum – 3 pts**
**Finishing the problem getting a simplified form for S – 3 pts**

5) (9 pts) Trace through a Radix sort of the numbers below showing the intermediate results:

| Original Numbers | First Iteration | Second Iteration | Third Iteration | Final Result |
|---|---|---|---|---|
| 1234 | 1671 | 1615 | 2136 | 1234 |
| 2298 | 1234 | 2324 | 1234 | 1378 |
| 3256 | 2324 | 1234 | 3256 | 1615 |
| 1615 | 1615 | 2136 | 2298 | 1671 |
| 2136 | 2655 | 2655 | 2324 | 2136 |
| 2324 | 3256 | 3256 | 1378 | 2298 |
| 1671 | 2136 | 1671 | 1615 | 2324 |
| 2655 | 2298 | 1378 | 2655 | 2655 |
| 1378 | 1378 | 2298 | 1671 | 3256 |

6) (16 pts) You're attempting to break a lock code. The code is ***n*** letters long. You know that the code has all distinct letters and you have a reference string, ***s***, where you know that each letter in ***s*** is incorrect, but each letter is within 3 of the correct letter in the code. For example, if the reference string = "SPAN", then you know that the first letter of the code has to be P, Q, R, T, U or V. Note that the third letter must be B, C or D since there are no letters before A. Complete the method below so that the initial method call (also shown below) prints out all possible strings that could be the correct code for the lock based on this information. Below a description of each formal parameter is given as well:

Method call to print all possibilities, assuming that s is correctly filled:

```
printPossible(new char[s.length], 0, s, new boolean[s.length]);

// curString - current String being built. k - how many letters
// have already been filled in for curString, s - reference string
// used is an array of size 26 where used[i] is set to true if
// letter i has already been filled in curString (one of the first
// k letters). When i=0, that's 'A', i=25 is 'Z'.
public static void printPossible(char[] curString, int k,
              char[] s, boolean[] used)

    if (k == s.length) {
        System.out.println(new String(curString));
        return;
    }

    // Fill in code below.
    for (int i=-3; i<=3; i++) {                    // 2 pts

        if (i == 0) continue;                      // 1 pt

        char c = (char)(s[k] + i);                 // 1 pt
        if (c < 'A') continue;                     // 2 pts
        if (c > 'Z') break;                        // 2 pts, cont ok
        if (used[c-'A']) continue;                 // 2 pts

        used[c-'A'] = true;                        // 1 pt
        curString[k] = c;                          // 2 pts
        printPossible(curString, k+1, s, used);    // 2 pts
        used[c-'A'] = false;                       // 1 pt
    }
}
```

7) (15 pts) Given an integer array, **a**, of size ***n***, consider the problem of finding the number of ordered triplets (i, j, k) where i < j < k and a[i] < a[j] < a[k]. You may assume that all the values in the array are in the range [**1**, $\mathbf{10^6}$]. The goal is to solve the problem in O(nlgn) time. A binary index tree can help in this task. Assume that you have a class that already implements a binary index tree with the following methods:

```
// Creates a new binary index tree with indexes 1 though n,
// inclusive with all elements initially set to 0.
public bit(int n)

// Adds value to the index idx in this binary index tree.
public void add(int idx, long value)

// Returns the sum of the values in this binary index tree in
// all indexes from index 1 through index idx, inclusive.
public long query(int idx)
```

Complete the method below that takes in an integer array ***a*** and returns the desired number of triplets in O(nlgn) time. **(Note: This is a challenging question. For 12 pts partial credit, count the number of pairs i < j where a[i] < a[j] in O(nlgn) time. My solution uses 3 bit objects.)**

```
public static long countTriplets(int[] a) {

    long res = 0;                              // 1 pt
    bit items = new bit(1000001);              // 1 pt
    bit pairs = new bit(1000001);              // 1 pt
    bit trips = new bit(1000001);              // 1 pt

    for (int i=0; i<a.length; i++) {           // 1 pt
        long cur = items.query(a[i]-1);        // 2 pts
        items.add(a[i], 1);                    // 2 pts
        pairs.add(a[i], cur);                  // 2 pts
        long curT = pairs.query(a[i]-1);       // 1 pt
        trips.add(a[i],curT);                  // 1 pt
        res += curT;                           // 1 pt
    }

    return res;                                // 1 pt
}
```

**Grading note: Attempt to do pairs will be graded differently.**

8) (15 pts) Design a greedy algorithm to solve the following problem (no code). Be as precise as possible and intuitively argue for the correctness of your algorithm: There are **n** + **m** children, each of which will receive one candy for Halloween (sorry, times are tough!) There are **n** Nestle Crunch bars and **m** Milky Way bars to distribute to the children. Each child has various preferences for these two candies. Specifically, child i gets $\mathbf{a_i}$ units of happiness from a Nestle Crunch bar and $\mathbf{b_i}$ units of happiness from a Milky Way bar. Naturally, you want to distribute the candy such that the total sum of the children's happiness is maximized. Determine an algorithm that determines this maximum sum of happiness (and one way of distributing the candy to achieve this level of happiness.)

Frame the problem a bit differently: Attempt to minimize the loss of happiness. A child loses $\max(a_i, b_i) - \min(a_i, b_i)$ happiness if they are given the candy they like less, since we know that all children will receive candy. The issue of course is that we might not have enough of the right type of candy to maximize each child's happiness. But, it's more important, for example, to give Joey the "right candy" if his $\max(a_i, b_i) - \min(a_i, b_i) = 1000$ than Sarah if her $\max(a_i, b_i) - \min(a_i, b_i) = 5$, since it's better for us to lose 5 happiness points collectively than 1000.

So, the greedy algorithm is as follows:

1. Sort the data by $\max(a_i, b_i) - \min(a_i, b_i)$, from greatest to least.
2. Loop through the children, giving them their preferred candy, unless it's run out. If it has run out, then you are forced to simply give the candy type that you have left.

The informal proof of correctness is as follows: Consider a different assignment of candy that is supposedly better than what the algorithm produces. This algorithm must give a less preferred candy to a child when our algorithm gives them their preferred one because our algorithm gives everyone their preferred one until it only has one type left...if the competing algorithm copied ours for each choice until one type ran out, it would not differ from ours. Thus, the competing algorithm gave child x their less preferred candy losing some level of happiness. This means that someone later down the list received their more preferred candy in the different assignment once our assignment ran out of their preferred candy. **BUT, our loss of happiness, by definition of our algorithm is less than or equal to the competing algorithm's loss of happiness. This contradicts the assumption that the competing algorithm was better than ours.**

Here's a little example with n = 4, m = 3

| Child | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $a_i$ | 22 | 1 | 24 | 10 | 14 | 5 | 7 |
| $b_i$ | 5 | 23 | 25 | 6 | 20 | 3 | 10 |

Here is the sorted list listed as triples $(i, a_i, b_i)$, and below which candy and score they get:

[(2, 1, 23), (1, 22, 5), (5, 14, 20), (4, 10, 6), (7, 7, 10), (6, 5, 3), (3, 24, 25)]
**M 23 N 22 M 20 N 10 M 10 N 5 N 24 Total = 114**

This is ONLY 1 off of the maximum possible happiness (sum of maxes) of 115, since we are only unable to provide the best candy at the very end! **(Grading: Grader's discretion!!!)**

Note: Apparently you can also sort by $a_i - b_i$ from largest to smallest, and just give the first ***n*** children in this list Nestle Crunch bars. Here we are minimizing the loss of the ***n*** Nestle bars. Let's say that someone gets a Nestle bar and it's non-ideal. So, let $a_x - b_x < 0$ and person x receives a Nestle bar. If we were to give this person a Milky Way bar instead, we'd have to swap the Milky Way given to child y, where

$a_y - b_y \leq a_x - b_x < 0$

Note that equivalently, this is

$0 < b_x - a_x \leq b_y - a_y$

In the old arrangement, Child Y was getting ideal happiness since he got his preferred Milky Way. But if we take his Milky Way away and give him a Nestle, the arrangement LOSES $b_y - a_y$ happiness and gains $b_x - a_x$ happiness.

The net change is:

$b_x - a_x - (b_y - a_y) \leq b_x - a_x - (b_x - a_x) = 0$, using the inequality above.

Thus, with the exchange argument, we prove that this alternative algorithm also works.

9) (1 pt) The book 438 Days is about a true story of Salvador Alvarenga's survival at sea on a small boat. How many days was he on his boat before he landed on a remote Pacific island?

**438 (Grading: Give to All)**