COP 3503 Spring 2022 Section 1 Exam #2 Solutions

1) (5 pts) In a Bucket Sort of 100,000 real numbers, all in the range from [30, 130), if we decide to create 100,000 equally sized buckets, in which bucket number (in between 0 and 99,999 inclusive), would the number 40.2376 go? (Please show your work. The numbers have been chosen such that a calculator is not necessary.)

40.2376 – 30 = 10.2376, which is how far from the left of the range this value is.

The range is 130 – 30 = 100, and each bucket will have a range of $\frac{100}{10000} = 10^{-3} = .001$.

It follows that the desired bucket is $\left\lfloor \frac{10.2376}{.001} \right\rfloor = \mathbf{10237.}$

Grading: 1 pt getting relative value to 30.
2 pts calculating range/bucket size
1 pt division idea
1 pt correct final answer

2) (6 pts) Show the result of each iteration of a Radix Sort on the following values. The last column has been filled in.

Initial List	First Sort	Second Sort	Third Sort	Sorted List
8417	2341	3111	3111	2431
2431	3111	8417	8137	3111
8137	4461	2431	8417	4461
3111	8417	8137	2431	8137
4461	8137	4461	4461	8417

Grading: 2 pts first column, 2 pts second column, 2 pts last column, give partial if 3 or 4 of the values in a column are correct only.

3) (14 pts) Consider the following game: You flip a coin. On the first flip it has a ½ chance of landing heads and a ½ chance of landing tails. If it lands heads, you stop. If it lands tails, you flip again. Unfortunately, the coin isn't kind. On the second flip, it will only land heads 1/3 of the time (and land tails 2/3 of the time). The game continues until you toss your first head. On the kth toss, the probability of getting heads is $\frac{1}{k+1}$. (Thus, the game is stacked against you, as it gets a little harder each time to flip a head!) Using the following steps, prove that the expected number of coin flips when playing this game is $\sum_{i=2}^{\infty} \frac{1}{i}$. (Note: this sum diverges, so the expected outcome is the game to go on forever!)

a) (1 pt) What is the probability you stop playing after 1 turn? $\frac{1}{2}$ **(prob H)**

b) (2 pts) What is the probability you stop playing after exactly 2 turns? $\frac{1}{2} \times \frac{1}{3} = \frac{1}{6}$ **(prob TH)**

c) (2 pts) What is the probability you stop playing after 3 turns? $\frac{1}{2} \times \frac{2}{3} \times \frac{1}{4} = \frac{1}{12}$ **(probTTH)**

d) (4 pts) Generalize the result for k turns. What is the probability you stop playing after exactly k turns, where k is a positive integer? (Your work should include a product of fractions. You may informally express this product and simplify it to relatively simple looking fraction.)

This is the probability of k-1 failures followed by one success. Formally, we have the probability as $\left(\prod_{i=1}^{k-1} \frac{i}{i+1}\right) \times \frac{1}{k+1} = \frac{\prod_{i=1}^{k-1} i}{(k+1)\prod_{i=2}^{k} i} = \frac{1}{(k+1)k}$

e) (5 pts) Now, apply the formula derived in part (d) and plug it into the expectation formula to create an expression that is equal to the expected number of coin flips when playing this game. Show your work and simplify your expression to the summation shown in the question.

To get the desired expectation, multiply the terms above by the number of tosses for each scenario, so we have the following sum: $\sum_{k=1}^{\infty} \left(\frac{1}{k(k+1)}\right) k = \sum_{k=1}^{\infty} \frac{1}{k+1} = \sum_{k=2}^{\infty} \frac{1}{k}$, the desired sum.

Grading: (a) 1 pt all or nothing
(b) 2 pts for correct answer, 1 pt for anything you think is worth partial
(c) 2 pts for correct answer, 1 pt for anything you think is worth partial
(d) 2 pts for writing product of fractions for Ts, 1 pt for multiplying fraction for H, 1 pt for canceling and getting the reduced expression.
(e) 1 pt for setting up sum from 1 to infinity.
1 pt for grabbing answer from (d)
1 pt for multiplying answer from (d) with summation index
2 pts for simplification into final form.

Note: no summation or product signs are required. … terminology with clear patterns and cross outs is fine.

4) (15 pts) Consider the following problem: given a list of ***n*** integers, determine if it's possible to place ***k*** of those integers in a set **S** and ***m*** of the integers (all different from the ones chosen to be in **S**) in a set **T** such that $1 \leq k < m$ but the sum of the integers in **S** is strictly greater than the sum of the integers in **T**. Write a static method that solves this problem taking in an input array of integers. You may assume that the sum of the integers in the array doesn't exceed 10^9. To get full credit, your method must run in O(***n*lg*n***) time. The function prototype is below. You may assume the input array is at least size 3 (so you don't have to worry about an array out of bounds in the initial code.) **Note: The input array can be in any initial order.**

```
public static boolean canSplit(int[] a) {

    Arrays.sort(a);                          // 4 pts

    int leftS = a[0], rightS = 0;            // 1 pt
    int i = 1, j = a.length-1;               // 1 pt

    while (i < j) {                              // 2 pts
        leftS += a[i++];                         // 2 pts
        rightS += a[j--];                        // 2 pts
        if (leftS < rightS) return true;         // 2 pts
    }

    return false;                            // 1 pt

}
```

Solution Idea(s)

We want set S to sum to a number greater than set T, so greedily we want to choose the largest values of set S and the smallest values for set T. Secondly, since T has to have more elements than set S, but we want its value to be small, it makes sense that we only try situations where T has exactly 1 more element than S. Thus, in the solution above, we try putting the two smallest values in T and the largest in S, followed by trying the 3 smallest values in T and the 2 largest in S, and so forth, until we find overlap between the sets (which doesn't help). Thus, in the code, I keep two running sums, one from the left of the smallest values and one from the right of the largest values. I always make sure that leftS represents the sum of one more value than rightS. And if we ever find a situation where rightS is greater than leftS, it's possible. If we never found one at the end, we know it's not possible.

The solution above can be simplified with the following observation:

Let's say that after the array is sorted that a[0] + a[1] < a[a.length-1]. If this is true, and we choose to add one number to each side, the number we add to the left must be less than or equal to the number we add on the right! Thus, if the former is true, then the following:

a[0] + a[1] + a[2] < a[a.length-1] + a[a.length-2]

must also be true. Generalizing this logic, we find that there's no need to check the smaller set sizes. Rather, we can just check the maximum set sizes for both S and T.

If n is even, we just greedily put in set T the n/2 smallest values and put in set S the largest n/2-1 values. Note that when n is even, there is one value in the array that is not assigned to either set.

If n is odd, we assign the (n+1)/2 smallest values to T and the largest n/2 values to S. In code, we can avoid separate cases as follows:

```
public static boolean canSplit(long[] a) {

     Arrays.sort(a);                          // 4 pts

     long leftS = 0;                          // 5 pts total
     for (int i=0; i<(a.length+1)/2; i++)
          leftS += a[i];                 // 3 pts if off by 1

     long rightS = 0;                         // 5 pts total
     for (int i=a.length/2+1; i<a.length; i++)
          rightS += a[i];                // 3 pts if off by 1

     return rightS > leftS;                   // 1 pt
}
```

Notes about this Problem

Technically, Arrays.sort is NOT O(nln) because it uses Quick Sort with a known partition element. Thus, it's possible for someone to create a test case where Arrays.sort takes around n^2 simple operations. However, this is extremely rare and can be thwarted by randomizing the input array. I decided to go with this version because I had asked the class to memorize Arrays.sort and not Collections.sort. So, I thought students would complain if I passed in an ArrayList requiring Collections.sort which runs in O(nlgn) time because it uses Merge Sort.

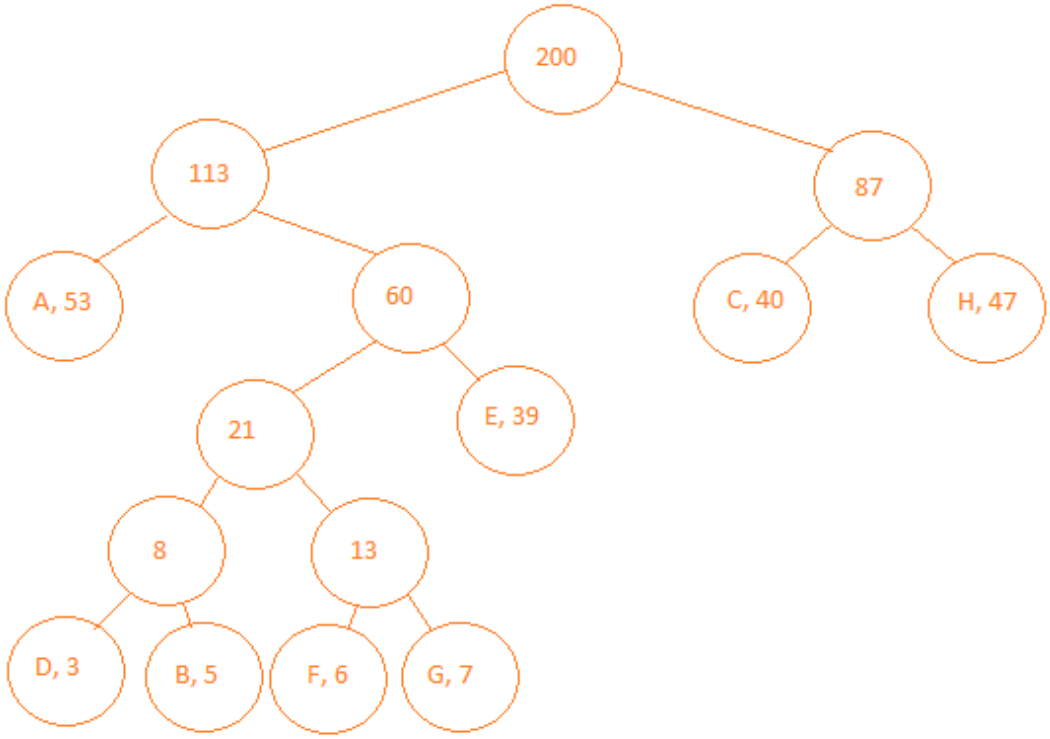
This problem is originally from Codeforces Division 2 Round 774 (https://codeforces.com/contest/1646).

I slightly modified this solution above because the real problem description requires longs. I didn't want you all worrying about that on the test, so I just guaranteed that the array values would be small enough that int works. To prove that this solution works, I submitted it on codeforces and the code is attached as a separate file (b_alt.java). You'll notice the array shuffling before calling the method. I also have posted the alternate solution which only checks the largest set split (b_alt2.java).

5) (10 pts) Draw a valid Huffman tree for a file that contains the following letter frequencies. Draw a circle around your final answer. In the interest of time, no need to give the codes for each letter or anything else. Just draw the tree. In each leaf node, write the letter and its frequency. In each internal node, just write the sum of frequencies in the corresponding subtree.

Letter	**Frequency**
A	53
B	5
C	40
D	3
E	39
F	6
G	7
H	47

Each time you remove two items from the Priority Queue, you can pick either of the two items to be on the left. Thus, there are many correct answers (128 in all since we do this 7 times…). Here is one of them:



Grading: 1 pt merge (D,B), 1 pt merge (F,G), 1 pt merge (8, 13),
2 pts merge (21, E), 2 pts merge (C, H), 2 pts merge (A, 60)
1 pt merge (87, 113)

6) (8 pts) In a breadth first search of 8 Puzzle positions, what are the four reachable board positions (in one move) from the following board state? Draw out each answer.

6	3	5
2		7
1	8	4

6		5
2	3	7
1	8	4

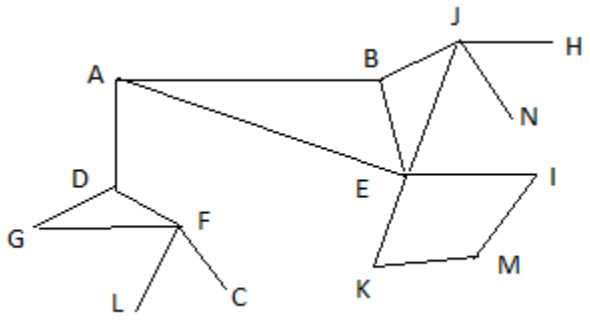
6	3	5
	2	7
1	8	4

6	3	5
2	7	
1	8	4

6	3	5
2	8	7
1		4

Grading: 2 pts per picture, all or nothing.

7) (7 pts) Show the order in which the vertices in the following graph get visited in a DFS starting at vertex A. Whenever looping through the neighbors of a vertex, always go through those in alphabetical order.



A, B, E, I, M, K, J, H, N, D, F, C, G, L

Grading: ½ pt per slot, round down.

8) (10 pts) Let G be a directed graph with ***n*** vertices which has precisely one possible topological sort.

(a) (1 pt) What is the fewest number of edges that could be in G, in terms of ***n***? n-1

(b) (2 pts) Give an informal proof that the answer in (a) is minimal.

With only n-2 edges, at most n-2 vertices will have incoming edges, meaning that a minimum of 2 vertices must have an in degree of 0. Either of these can be chosen for the first vertex in the topological sort, and since we assume a topological sort exists for the graph, there must be at least two topological sorts for any graph with at most n-2 edges which does have a topological sort. Thus, n-2 edges is insufficient. To achieve n-1 edges, if we label the vertices v_1 through v_n, just have edges of the form $v_i \rightarrow v_{i+1}$, for each i in the range [1, n-1]. This forces the ordering of the vertices in numerical order.

Grading: Judgement call, but the key idea is that the graph isn't connected and there must be at least two "components", each with a vertex with in degree 0. Give partial as you see fit.

(c) (3 pts) What is the maximum number of edges that could be in G, in terms of ***n***?

$\frac{n(n-1)}{2}$

Grading: All or nothing here. If you feel compelled to give partial, please run it by me.

(d) (4 pts) Give an informal proof that answer in (c) is maximal.

First, let's prove that any directed graph with $\frac{n(n-1)}{2} + 1$ edges has no topological sort. It's clear that if a graph has both vertices $v_i \rightarrow v_j$ and $v_j \rightarrow v_i$, it can not have a topological sort. There are a total of n(n-1) possible edges in a directed graph of n vertices. (The starting vertex is one of n choices, and the ending vertex can be anything but the starting vertex, so one of n-1 choices.)

Group our potential edges in $\frac{n(n-1)}{2}$ pairs, where each pair is of the form { (v_i, v_j), (v_j, v_i) }, for each distinct pair of i, j with $1 \leq i < j \leq n$. By the Pigeonhole Principle, if we choose $\frac{n(n-1)}{2} + 1$ edges from these $\frac{n(n-1)}{2}$ sets, we must choose 2 items from at least one of the sets. But if we do, then our graph has two edges that prevent a topological sort on the graph. To see that we can have a graph with $\frac{n(n-1)}{2}$ edges, just choose to add every edges of the form $v_i \rightarrow v_j$, where i < j. This is precisely $\binom{n}{2} = \frac{n(n-1)}{2}$ edges. This graph has a single possible topological sort of the vertices in numerical order.

Grading: 3 pts for somehow showing that a larger # of vertices implies that the graph has a cycle of size 2, 1 pt for completing the proof. Give partial as needed.

9) (10 pts) Show the contents of the distance array after each iteration of Dijkstra's algorithm for the directed graph G, with starting vertex A, whose edges are designated in the adjacency matrix below. (The row represents starting vertex of an edge and the column the ending vertex of that edge.) In addition, for each iteration, show the new vertex that gets pulled from the priority queue, indicating that the current distance stored to it is shortest. If an entry is missing, there is no edge between those vertices. **(Note: to get full credit you must fill in every square that is empty in the answer grid.)**

From/To	A	B	C	D	E	F
A		27	13	99	15	3
B			2	8	6	3
C		3		30	1	1
D		8	7		20	
E		9	30	2		10
F		20	12	27	9	

Place Your Answer Below

Add\To	A	B	C	D	E	F
None	0	∞	∞	∞	∞	∞
A	0	27	13	99	15	3
F	0	23	13	30	12	3
E	0	21	13	14	12	3
C	0	16	13	14	12	3
D	0	16	13	14	12	3

Grading: 2 pts per row, give 1 pt for a row if most things on the row are right.

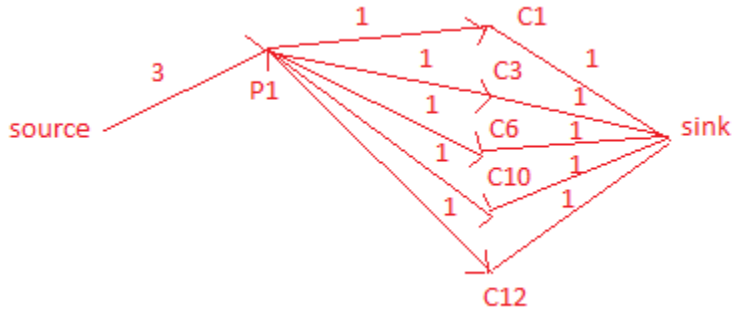
10) (10 pts) A university has a set of professors, $\{P_1, P_2, P_3, \ldots, P_n\}$ and a set of classes $\{C_1, C_2, C_3, \ldots, C_m\}$. Professor P_i must teach exactly a_i classes from a set S_i of classes they know how to teach. Note that the sum of the a_i's equals m (so the # of classes the professors are required to cover equals exactly the number of courses.) Describe how to set up a network flow graph that determines whether it's possible to assign the professors to cover all of the courses. Draw a picture of a portion of the graph for one professor, P_1 who must teach 3 courses from the set C_1, C_3, C_6, C_{10} and C_{12}. Also, explain how, if we know the max flow of the graph, we can determine whether or not all of the classes can be covered.

The vertices in the graph will be the source, one vertex for each professor, one vertex for each class, and the sink.

Connect the source to each professor. The capacity of the edge from source to the vertex for professor P_i is the number of classes they must teach, a_i. Then, put an edge from each professor to each class they can teach with a capacity of 1. Finally, draw an edge from each class to the sink with capacity 1.

If the maximum flow through this network equals m, then all the classes can be covered. Otherwise, they can not be.

Here is an illustration of a portion of a network with professor P_1 who needs to teach 3 classes out of C_1, C_3, C_6, C_{10} and C_{12}:



Grading: 1 pt source, 1 pt sink, 1 pt vertex for each prof, 1 pt vertex for each class
2 pts capacity equal to a_i from source to each prof
1 pt capacity of 1 from each prof to each class they teach
1 pt capacity 1 from each class to sink
2 pts for testing if max flow equals number of class
Note: Graph can run in other direction.

11) (5 pts) NASA's Space Launch System rocket with Orion spacecraft was rolled out to launchpad 39B at Kennedy Space Center last week for some tests. What acronym is used to describe the Space Launch System?

SLS (Grading: give to all)