COP 3503 Spring 2024 Section 1 Exam #2 Solutions

Sheet 1: Algorithm Analysis, Extra Sorts

1) (6 pts) You've run an algorithm that processes an array of size ***n***, for several values of ***n*** and have recorded the execution times in the table below. Use the technique shown in class (blank columns have been provided for your use) and determine the most likely Big-Oh run time of the algorithm in terns of ***n***.

n	***Run time (ms)***	T(n)/n	$T(n)/n^2$	$T(n)/n^{1.5}$	
10000	98	.0098	9.8×10^{-7}	.000098	
50000	1162	.02324	4.6×10^{-7}	.000103	
100000	3016	.03016	3.0 x 10-7	.000095	
200000	9231	.046155	$2/3 \times 10^{-7}$	.000103	

We guess T(n) = O(n), but when we look at the numbers in the first column are steadily increasing, so the real run time must be larger. Next we guess T(n) = $O(n^2)$. But, the numbers in the second column are steadily decreasing. Finally, we try somewhere in between, $T(n) = n\sqrt{n}$. Looking at this column, this nearly stays constant. This is the most- likely run time of the algorithm.

$O(n\sqrt{n})$

Grading: 4 pts for filling in at least 2 columns (or 1 correct column) correctly.
2 pts for using the evidence to arrive at the correct answer (must have the column for $n^{1.5}$ to earn this credit.)

2) (8 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	Fourth Sort	Sorted List
32361	32361	34629	48147	32361	24169
48147	32661	48341	24169	32647	32361
32661	48341	48147	48341	32661	32647
58347	48147	58347	58347	24169	32661
34629	58347	32647	32361	34629	34629
24169	32647	32361	34629	48147	48147
48341	34629	32661	32647	48341	48341
32647	24169	24169	32661	58347	58347

Grading: 2 pts for each column, give 2 pts if completely correct. Give 1 pt for a column if it has at least 5 values in the correct slots.

3) (11 pts) Let T(n) represent the average number of nodes with two children in a binary search tree. Assuming that the root node of a binary search tree has probability of $\frac{1}{n}$ of being any particular rank (from 1 to n) in the sorted list of values stored in the tree, write down a recurrence relation that T(n) satisfies **and simplify this recurrence so that there's a single sum added to a single term on its right-hand side**.

There is a 1/n probability that there will be x nodes on the left subtree of a tree of n nodes and n – 1 – x nodes in the right subtree. If either x = 0 or n – 1 – x = 0 (which means x = n – 1), then the root node will NOT have two children. In all other cases, the root node will have two children, and should be counted. Here is a table showing the probability of each split and the number of nodes with two children in each case.

x	p(x)	T(n)
0	1/n	T(0) + T(n – 1)
1	1/n	T(1) + T(n – 2) + 1
2	1/n	T(2) + T(n – 3) + 1
…	1/n	
x	1/n	T(x) + T(n – 1 – x) + 1
…	1/n	
n – 2	1/n	T(n – 2) + T(1) + 1
n – 1	1/n	T(n – 1) + T(0)

To get the recurrence for T(n), we must multiply the value in the second and third columns of each row, and then add each of these together. When we do this, we get:

$$T(n) = \frac{1}{n}T(n-1) + \frac{1}{n}\sum_{i=1}^{n-2}[T(i) + T(n-1-i) + 1] + \frac{1}{n}T(n-1)$$

Note: Since T(0) = 0, these terms have been omitted above. Now, gather terms and pull out the portion of the summation that isn't a function of T to get:

$$T(n) = [\frac{2}{n}\sum_{i=1}^{n-1} T(i)] + \frac{n-2}{n}$$

Incidentally, as will be shown in class, this recurrence has a pretty clean closed-form solution:

$$T(n) = \frac{n-2}{3}$$

Grading: 4 pts for the rows in the table from 1 to n – 2,
2 pts for realizing that if x = 0 or x = n – 1, then there's no plus 1.
3 pts for writing down any form of the RHS from the table
2 pts to simplify the initial written form to the cleaner form with one sum

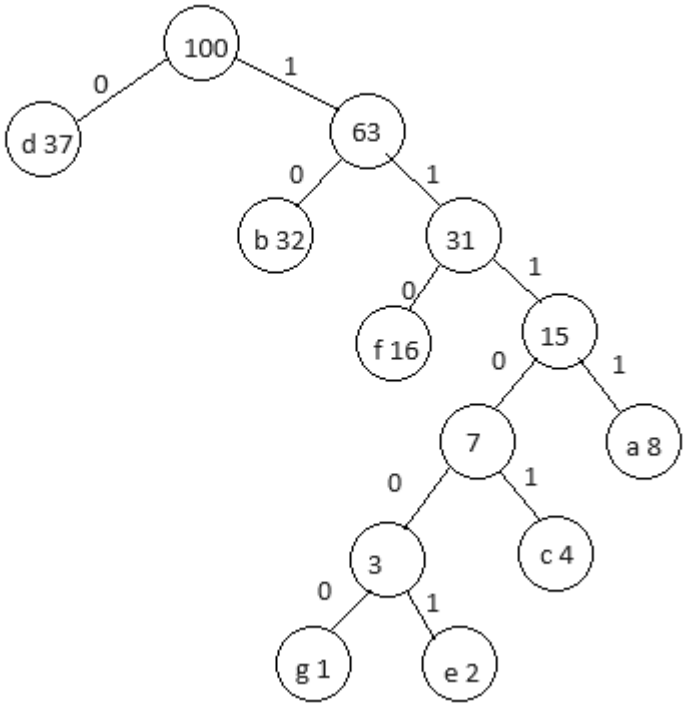
Sheet 2: Greedy Algorithms

4) (10 pts) A Huffman tree has been created for a particular input file comprised solely of the characters ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’ with the codes listed:

Letter	Huffman Code	Possible Letter Frequency
‘a’	1111	8
‘b’	10	32
‘c’	11101	4
‘d’	0	37
‘e’	111001	2
‘f’	110	16
‘g’	111000	1

(a) Draw the corresponding Huffman Tree below.

(b) Fill in possible letter frequencies, ***without any tie cases***, for each of the seven letters in both the tree and the table above that could create this set of Huffman codes. (Note: Both the frequencies of each letter must be distinct AND when comparing items in the PQ, there should never be a tie between the second and third smallest items in it.



Grading: 5 pts for a valid Huffman tree structure with letters drawn in. (Note: for each branch 0 and 1 can go either direction…) 5 pts for codes that fit the data without ties. If there are ties but it works, 3/5 on this. Give partial as needed.

5) (15 pts) Arup's World Wide Coffee Emporium sells cups of coffee. He sells several different types of coffee. Each type of coffee has the following attributes: cost (# of cents it costs Arup to make one cup of this type of coffee), sell (# of cents Arup sell's each cup of this type of coffee), and numcups (the number of cups he has in inventory of this type of coffee.) For this question: write the compareTo method for the coffeecup class AND write a function which takes in an array of type coffeecup storing Arup's total inventory and an integer, totalcups, and returns the maximum profit he can make by selling exactly totalcups number of cups of coffee.

```
class coffeecup implements Comparable<coffeecup> {
     public int cost;
     public int sell;
     public int numcups;

     // Constructor omitted.

     public int compareTo(coffeecup other) {

          int p1 = sell - cost;
          int p2 = other.sell - other.cost;

          return p2 - p1;
     }
}
public static long maxProfit(coffeecup[] list, int totalcups) {

    Arrays.sort(list);
    long res = 0;
    for (int i=0; i<list.length; i++) {
        int buy = Math.min(totalcups, list[i].numcups);
        totalcups -= buy;
        res = res + ((long)buy)*(list[i].sell-list[i].cost);
        if (totalcups == 0) break;
    }

    return res;
}
```

Grading: compareTo: 6 pts → 2 pts calc this profit, 2 pts calc other profit,
2 pts returning correctly based on these profits
maxProfit: 2 pts → sort
1 pt → accumulator variable set up
3 pts → figure out correct amount to buy (if stmt likely)
1 pt → update amount bought or left to buy
2 pts → update res, 1 pt if overflow

Sheet 3: Unweighted Graph Algorithms Solutions

6) (10 pts) For this problem, you must figure out the lexicographical first ordering of completing tasks 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, given the following constraints. Each constraint is given in the form (a, b) indicating that task a must be completed before task b.

(5, 2), (3, 1), (7, 4), (4, 3), (10, 2), (6, 3), (8, 10), (9, 5)

In degrees are on this chart:

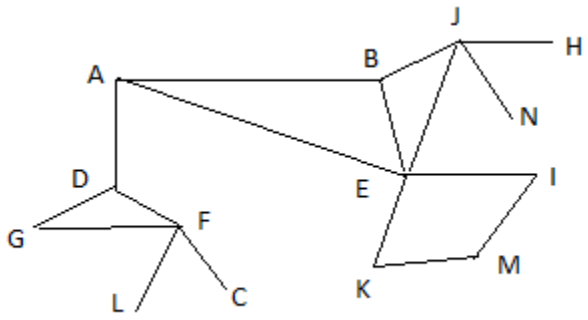
Node	1	2	3	4	5	6	7	8	9	10
In Deg	1	2	2	1	1	0	0	0	0	1

This we must start our top sort with vertex 6, the smallest of the possible options. Add this to the answer and then process all edges leaving 6: (6, 3) → indeg[3] = 1. Now, take the smallest possible item next, which is 7. Process the edge leaving 7: (7, 4) → indeg[4] = 0, add to priority queue (so that we always pull the minimum possible item!), So 4 goes next. Process the edge leaving 4: (4, 3) → indeg[3] = 0. Vertex 3 follows. Process edge (3, 1) → indeg[1] = 0. Add 1 to top sort, no edges leaving 1 to process. 8 is smallest possible item to add that's left. Process edge (8, 10) → indeg[10] = 0. Smallest item in queue is 9. Process edge (9, 5) → indeg[5] = 0. Process edge (5, 2) → indeg[2] = 1. At this point, only item in the queue is 10, add it to the top sort. Process edge (10, 2) → indeg[2] = 0. So finally, we can add 2 to the complete the top sort.

6, 7, 4, 3, 1, 8, 9, 5, 10, 2

Grading: 1 pt per slot, all or nothing for each slot.

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



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

Grading: ½ pt slot, round down to integer, all or nothing per slot.

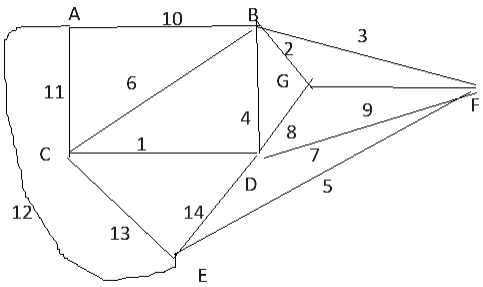
8) (8 pts) Imagine a building with ***n*** floors labeled 0 through ***n*** – 1, an elevator which can either move you ***a*** floors up or ***b*** floors down, and that you start on floor, ***s*** ($0 \leq s < n$). You can use a depth first search to mark all of the floors that are reachable. Complete the recursive dfs method below so that it fills in the boolean array, used, so that after the initial depth first search completes, used[i] will be set to true if and only if it's possible to reach floor i from floor ***s*** via a series of elevator moves, each of which go a floors up or b floors down. (Note: a and b are both guaranteed to be positive integers.)

```
public static void dfswrapper(int n, int a, int b, int s) {
    boolean[] used = new boolean[n];
    dfs(n,a,b,s,used);
}

// Marks all reachable floors in a building of n floors starting
// at the floor cur with an elevator that can go up a floors or
// go down b floors.
public static void dfs(int n, int a, int b, int cur, boolean[]
used) {

    used[cur] = true;

    if (cur+a < n && !used[cur+a]) dfs(n, a, b, cur+a, used);

    if (cur-b >=0 && !used[cur-b]) dfs(n, a, b, cur-b, used);


}
```

Grading: 2 pts for marking cur true
3 pts for if and recursive call up
3 pts for if and recursive call down

Sheet 4: weighted Graph Algorithms

9) (10 pts) Trace through Kruskal's Algorithm on the graph drawn below. Show, in order, each edge that is considered to be added to the minimum spanning tree and then put either "yes" or "no" in the second column to indicate whether or not that edge actually gets added to the minimum spanning tree.



Edge to Consider	Is it Added? (yes/no)
CD (1)	yes
BG (2)	yes
BF (3)	yes
BD (4)	yes
EF (5)	yes
BC (6)	no
DF (7)	no
DG (8)	no
GF (9)	no
AB (10)	yes

Grading: 1 pt per row, full row must be correct to get the point.

10) (10 pts) Let **G** be a directed graph with **n** vertices, where **n** < 1000. Each edge in **G** is assigned a color: red, green or blue. We are given a start vertex in the graph, **s**, and a destination vertex in the graph, **e**. Devise an algorithm (describe in words) to find a path from **s** to **e**, which minimizes the number of red edges on the path. If two paths contain the same number of red edges, the number of green edges should be minimized. If two paths contain the same number of red and green edges, then the number of blue edges should be minimized. The goal of the algorithm is to simply output the number of red, green and blue edges on a path that achieves this objective. Be as clear in your description as possible. You may state the use of any of the algorithms taught in class, but must clearly and unambiguously state the input you're using to that algorithm. (Hint: Any shortest path in a graph of **n** vertices with positive edge weights only can not contain more than **n** – 1 edges.)

Assign each red edge a weight of 10^6. Assign each green edge a weight of 10^3. Assign each blue edge a weight of 1. Run Dijkstra's Algorithm on this graph with a start vertex of s to get all of the shortest paths from s to each other vertex. Let D be the shortest distance from vertex s to vertex e in this graph. Then, we can solve for the number of red, green and blue edges on the desired path as follows (note: all divisions are integer divisions)

Red Edges = D/1000000
Green Edges = (D%1000000)/1000
Blue Edges = D%1000

Since n < 1000, a single red edge is more costly than the sum of a path with the maximal number of green and blue edges. (Without a red edge the maximal shortest distance would be 998,000 since the most number of edges on a graph with 999 vertices is 998.) Thus, we can be confident that when we divide D by a million, this will uncover the number of red edges on the path. More importantly, by weighting red edges this high, their number will naturally be minimized by Dijkstra's algorithm. We repeat this logic to show that for all paths with the same number of red edges, green edges are minimized. Finally, in order to minimize blue edges in the case that there's a tie between both red and green edges, we must assign a positive weight to blue edges, so those get minimized in that case.

We are using a data compression scheme to keep track of three pieces of data (number of red, green and blue edges) in a single value. Knowing that each of these numbers is less than 1000, we can simply store the equivalent of a 3 digit number in base 1000 to encode any possible number of red, green and blue edges we might encounter on an individual path.

Grading: 3 pts for any mention of Dijkstra's
2 pts for any weighting scheme for edges (assigning each color to a weight0
3 more pts for any VALID weighting scheme
2 pts for the clarity of the explanation

11) (5 pts) March 14th is often called Pi Day because the irrational value of Pi, rounded to 2 decimal places is 3.14. Those who are not mathematically inclined often use this day to celebrate a common dessert. What is that dessert?

Pie (Grading: Give to All)