Solutions for COP 3503 Exam 1 Review Questions for Fall 2026

Part A (Java API) – From Spr 24 Exam 1

1) (10 pts) What is the expected run-time of each of the following operations in the Java API? For each of these questions, assume that the data structure in question currently has ***n*** items in it.

(a) Collections: sort(List<T> list) **O(nlgn)**

(b) ArrayList: add(E e) **O(1)**

(c) ArrayList:add(int index, E element) **O(n)**

(d) TreeSet: lower(E e) **O(lg n)**

(e) HashMap: get(Object key) **O(1)**

(f) PriorityQueue: poll() **O(lg n)**

(g) PriorityQueue: remove(Object o) **O(n)**

(h) ArrayList: contains(Object o) **O(n)**

(i) HashSet: add(Object o) **O(1)**

(j) ArrayList: size() **O(1)**

Grading: 1 pt all or nothing on each.

2) (15 pts) The rolling median problem is as follows: As you read in ***n*** integers, after reading in each one, output the current median of the values. (Recall, that the median of a list of an odd number of values is the middle value, and that the median of a list of an even number of values is the average of the two middle values.) For this question, you'll complete the code on the back of this page so that it prints out **twice the current median value**. (This is so we can avoid floating point numbers…)

The strategy to solve the problem efficiently is as follows: Keep two priority queues, one a max queue which will store the "smaller half" of values (called left in the code) and another priority queue (a min queue), which will store the "larger half" of values. After reading in each number, insert it into the appropriate side (for example, if the left has [6, 2, 9] and right has [15, 13, 22] and we're inserting 19, it will go on the right.) Then, in the worst case, one of the two priority queues will have 2 more items than the other. If this is the case, then remove one item from the larger sized queue and put it into the smaller sized queue. Once this is done, the median is easy to calculate. (We'll let you figure it out from here.)

The Java API methods you need will be described at the end of the next page.

```
import java.util.*;

public class rollingmedian {
    public static void main(String[] args) {

        Scanner stdin = new Scanner(System.in);
        int n = stdin.nextInt();

        PriorityQueue<Integer> left = new
            PriorityQueue<Integer>(Collections.reverseOrder());
        PriorityQueue<Integer> right = new PriorityQueue<Integer>();

        for (int i=0; i<n; i++) {

            int val = stdin.nextInt();

            if (i == 0) {
                left.offer(val);
                System.out.println(2*val);
                continue;
            }

            // Decide which side to add val to.
            if (val > left.peek())  // several ways, > or >= okay… // 1 pt
                right.offer(val);                                 // 1 pt
            else                                                  // 1 pt
                left.offer(val);                                  // 1 pt

            // Transfer value from left to right or right to left if necessary.

            if (left.size() > right.size()+1) right.offer(left.poll());// 2pts
            if (right.size() > left.size()+1) left.offer(right.poll());// 2pts

            // Print out twice the current median (3 cases)
            if (left.size() > right.size())                       // 1 pt
                System.out.println(2*left.peek());                // 1 pt
            else if (right.size() > left.size())                  // 1 pt
                System.out.println(2*right.peek());               // 1 pt
            else                                                  // 1 pt
                System.out.println(left.peek()+right.peek());     // 2 pts


        }
    }
}
```

Backtracking – Spr 24 Exam (except question 3)

4) (5 pts) In class, code to solve the Tentaizu problem was executed, which ran fairly quickly. A single line of code was commented out and then the solution ran quite slowly (it solved one puzzle in several seconds, while we waited.) Conceptually, what did that line of code do?

If we completed square (r+1, c+1), but the number of bombs surrounding square (r, c) was not correct, that line of code said the board was not viable or possible to produce any valid solutions. This works because no new bombs adjacent to (r, c) will ever be placed after we complete our selection for square (r+1, c+1).

Grading: Holistic, up to grader based on clarity of explanation. Note that there are may ways to express this idea.

5) (5 pts) In class, we visualized a solution to the Eight Queens puzzle as a permutation of the integers from 1 to n, where the i^{th} integer in the permutation represented which row to place the queen in column i. Using this same visualization technique, draw the solution that corresponds to the first permutation lexicographically that correctly solves the five queens problem below.

Q				
			Q	
	Q			
				Q
		Q		

Grading: 2 pts for any valid 5Q solution, 0 pts if Queens attack each other or there aren't 5 of them, 5 pts if it's perfectly correct and 5 pts for the transpose of this picture (placing [1,3,5,2,4] in the rows instead of the columns.

6) (15 pts) A prefix composite number is one such that each prefix of it is also a composite number. For example, 44052 is a prefix composite number that is 5 digits long because 4, 44, 440, 4405 and 44052 are all composite numbers. Complete the program on below so that it prints out all prefix composite numbers that are 5 digits long.

```
import java.util.*;

public class prefixcomposite {

     public static void main(String[] args) {
          go(0, 0, 5);
     }

     // cur is a valid prefix-composite of k digits
     // n is the # of digits in the prefix-composites to be printed
     public static void go(int cur, int k, int n) {

          if (k == n) {                          // 1 pt
               System.out.println(cur);          // 1 pt
               return;                           // 1 pt
          }

          for (int i=0; i<10; i++)               // 1 pt
               if (!isprime(10*cur+i))           // 2 pts
                    go(10*cur+i, k+1, n);        // 3 pts
     }

     // Returns true if n is prime or if n is less than 2.
     // Must run in O(sqrt(n)) time and not use any doubles for
     // full credit.
     public static boolean isprime(int n) {
         for (int i=2; i*i<=n; i++)              // 2 pts
             if (n%i == 0)                       // 2 pts
                 return false;                   // 1 pt
         return true;                            // 1 pt
     }
}
```

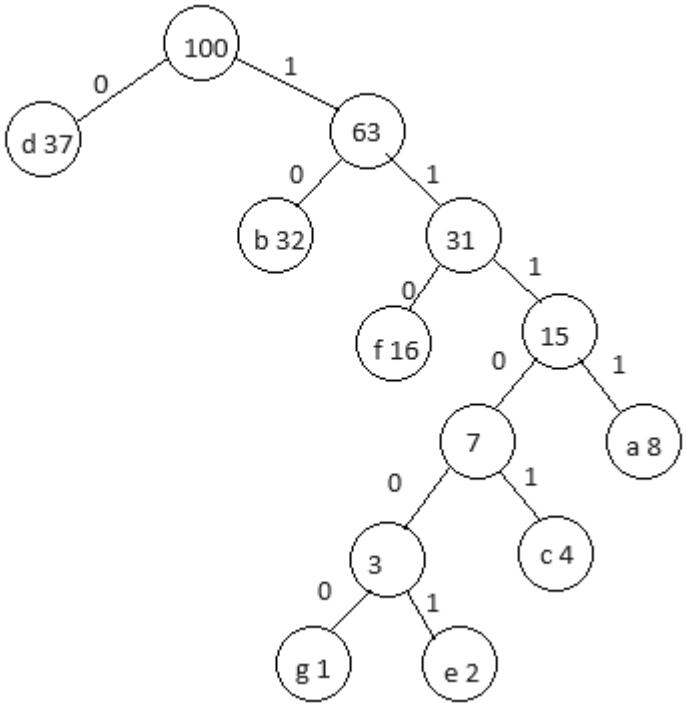
Greedy Algorithms – Spr 24 Exam 2

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

Sorting, Alg Analysis – Spr 24 Exam 2

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