Midterm 2 Repeat Grading Guide

The purpose of the scoring guide is to provide a uniform evaluation of the tests by the graders. Therefore, the guide gives you the main ideas (and at least one possible solution) for each problem and the associated scores. This guide is not intended to be a detailed description of a complete solution to each problem; the steps described are sketches of a solution that achieves maximum score.
The partial scores given in the guide are only awarded to the solver if the related idea is included in the test as a step in a clearly described and reasoned solution. Thus, for example, the mere description of statements or definitions in the material without its application (even if any of the facts described do indeed play a role in the solution) is not worth the point.
A partial score is given for any idea or part solution from which the correct solution of the problem could be obtained with the appropriate addition of the reasoning described in the test. If a solver starts several solutions that are significantly different, then you can give score for at most one. If each described solution or part of solution is correct or can be extended to be correct, then the solution with the most points will be evaluated. However, if there are correct and incorrect solutions between several solution attempts and it is not clear from the test which one the solver considered to be correct, then the solution with fewer points will be evaluated (even if this score is 0). The scores in this guide may be further subdivided as necessary. Of course, a good solution other than that described in this guide is also worth the maximum score.
In the event of an arithmetic error, one point is deducted from each exercise. An exception to this is when computing the numbers significantly simplifies or modifies the structure of the task. In these cases, no points are awarded to the steps which follow the computation and are missing from the solution of the student.

---

1. (a) The numbers $3, 2, 1, 8, 5, 7, 6$ are inserted, in this order, into an empty binary search tree. Draw the binary search tree obtained after every insertion.
   (b) We add leaves without any value/key (NIL) to this binary search tree so that all internal nodes have values and two children and all leaves are nodes without values. Can we color the nodes of this tree so that we get a red-black tree?

   Grading guide:

   (a) Correct drawing of 7 trees. **3 points**
   (b) Correct drawing of the tree with NIL as leaves: **2 points**
   Root is black and leaves (NIL) are black in a RB-tree. **1 point**
   From the $3, 8$,NIL path, black height is at most 3. **1 points**
   From the $3, 8, 5, 7, 6$,NIL path, black height is at least 4 because a red node cannot have red children, so at least two of $8, 5, 7, 6$ must be black. **1+1 points**
   This is a contradiction so there cannot be such a RB-tree. **1 points**

2. A 23-tree contains the following numbers $2, 5, 10, 12, 14, 15, 19, 21$. We further know that the root is a 3-node.
   (a) Draw this tree. Give brief reasons for your answer. Please do not show insertion or tree building steps here.
   (b) Insert 11 into this tree. Please show the steps.

   Grading guide:

   (a) If the tree has only two levels, then maximum number of elements it can contain is $8 = 2 + 6$, if all nodes are 3-nodes. **1 point**
   If the tree has three levels, then the minimum number of elements it contains is $11 = 2 + 3 + 6$, where apart from the root, all other nodes are 2-nodes. **1 points**
   So, the tree must have two levels and all the nodes are 3-nodes. **1 points**
   To preserve the 23-trees property, there can be only one way to fill the numbers, correct drawing of the tree. **2 points**

(b) Correct position of 11 creating a temporary 4-node. **1 point**
Correct push up of 12 creating a temporary 4-node at the root. **2 points**
Correct push up of 12 creating a tree with one more level. **2 points**

3. We are given two arrays, each containing n distinct integers. Give an algorithm with running time $O(n \log n)$, that determines if there are at least $\frac{n}{2}$ numbers that occur in both arrays.

Grading guide:

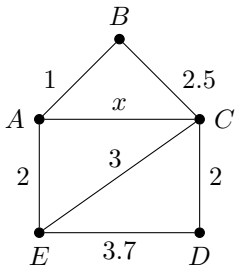
Sorting both arrays with mergesort or heapsort. RT is $O(n \log n)$ **4 points**
Merging the two arrays. RT is $O(n)$ **2+1 points**
Checking for duplicates. RT is $O(n)$ **2+1 points**
If any other algorithm is used for merging with worse RT, then maximum of 6 points. If no sorting is done then 0 points.

4. Design a data structure for storing distinct integers, such that the following operations can both be performed with $O(\log n)$ running time, where n is the number of elements stored in the data structure:
INSERT(i): insert number i into the data structure
FIND-SMALLEST(k): find the k^{th} smallest element in the data structure.

Grading guide:

A Red-black tree (or a 23-tree) is needed. **1 point**
Further a counter is required to store the number of items in the left subtree. **2 points**
Insert: usual insert of red-black tree, but if insert proceeds to the left subtree, then the counter of the root is increased by 1. **1 points**
Further, during rotations, if it is a left rotate, then the new root's counter is updated with the sum of its counter and the old root's counter. For a right rotate, we will subtract the counters. Essentially we have to make sure the counters hold correct values after rotate. **2 points**
Find-smallest: If k = counter of the root, then we return the value at the root. If $k <$ counter of the root, we will proceed to search in the left subtree. Otherwise we search for $k-$counter in the right subtree. **2 points**
RT of insert is $O(\log n)$ as RB-trees insertion has that RT, and we are only updating the counters of nodes we traverse during insert. RT of find-smallest is also $O(\log n)$ as we go as many steps as the height of the tree. **1+1 point**

5. In the graph given (where x is not necessarily an integer), Prim's algorithm was run from vertex A and in the process the edge of weight x was not included in the minimum spanning tree.
(a) Which edges were selected by the Prim's algorithm and in what order? As justification, briefly describe the general steps taken by Prim's algorithm (in 1-2 sentences, the Q table is not required).
(b) What can be the value of x and why?



Grading guide:

Prim's will expand a connected component from A, always picking the smallest weight edge out of the component. **2 points**
First edge is min{AB, AC, AE} which is same as min{AB, AC}. Since AC is not picked, AB is picked. And so $x \geq 1$ **1+1 points**
Second edge is min{AE, AC, BC } which is same as min{AE, AC}. Since AC is not picked, AE is

picked. And so $x \geq 2$ **1+1 points**
Third edge is min{ED, EC, AC, BC } which is same as min{BC, AC}. Since AC is not picked, BC is picked. And so $x \geq 2.5$ **1+1 points**
At this point AC induces a cycle, so it will never be picked. The fourth edge, therefore is CD. Then the required range is $x \geq 2.5$ **1+1 points**

6. Show that the following language is in coNP:

   DIFF-SUBSET-SUM = $\{(s_1, s_2, \ldots, s_n) | s_i$ are positive integers and the sum of the numbers in any two different subsets is different$\}$

   Grading guide:

   Ignoring inputs that don't match the syntax, the complement is = $\{(s_1, s_2, \ldots, s_n) | s_i$ are positive integers and there are two subsets such that their sum is the same$\}$ **1 point**
   A list of two subsets with the same sum is a witness. **1 point**
   If the maximum number in the list is s, then the size of the input is $O(n \log s)$. **1 point**
   Size of the witness is $O(2n \log s)$ which is atmost twice the size of the input, so it is poly size. **1+1 points**
   Verification requires:
   Checking if the two subsets are different. Any sorting algorithm can be used here. **1+1 points**
   Checking if the sums of both are the same. Addition take polynomial time. **1+1 point**
   Since the complement is in NP, the original language must be in coNP. **1 point**

7. Consider the following language:
   RED-BLACK-HAMCYCLE = $\{G | G$ is an undirected, connected graph on $2n$ vertices where n vertices are colored red, remaining n are colored black, and G has a Hamiltonian cycle such that along the cycle, the red and black vertices form a path $\}$
   (So if G is in the language, then it has a Hamiltonian cycle and there is a vertex from which if we walk along the Hamiltonian cycle, we will first encounter all red vertices, and then all black vertices.)
   Show that RED-BLACK-HAMCYCLE is NP-complete.

   Grading guide:

   We must check that this language is in NP. Such a hamcycle is a witness. Its size is atmost the size of the input. **1 point**
   Verification: check if the vertices are all there, the colors are in order and there are edges. All these checks are polytime. **1+1 points**
   If we give a Karp reduction of an NP-complete language to this, then that would prove that this language is also NP-complete. **1 point**
   st-HAMPATH or HAMPATH is a good choice for the NP-complete language. **1 point**
   Good reduction. **2 points**
   It is polynomial time to derive G' from G **1 point**
   Proof of if . **1 point**
   Proof of only if. **1 point**