Panther ID (No name, please):

COP 5407: Intro. to Algorithms MIDTERM EXAM; Spring 2019

Max Score = 75 pts; Max Time = 75 minutes;

Read This: This is a "closed book" exam and has a total of 4 pages. To maximize partial credit, write down the basic ideas behind your solution and fill in the details later, if time permits. For a solution that uses DP, you can get partial credit for following the six steps outlined in class. However, the heart of the problem is in the step that gives the recurrence relation with a clear definition of what the notation means. Analyze the worst-case time complexity of all algorithms you are asked to design.

---

1. [38] Your friend has n toy cars arranged in a straight line on her shelf. Each car has one color on its front end and a different color on its rear end. For example, a car may have a red front and a blue rear. Unfortunately, your obsessive-compulsive personality is unable to tolerate this chaos of colors. You insist that the toy cars must be **color-matched**, i.e., they must be lined up so that the color on the rear end of any car must be the same as the color on the front of the next car behind it. To achieve this, you want to remove some cars from the lineup. For example, let the initial sequence of 7 cars have the following colors (R is red, B is blue, G is green, O is orange):

   (Head) [R–B] [R–O] [B–O] [O–G] [B–G] [G–B] [G–R] (Rear)

   Then you could remove the cars numbered 2, 3, 4, and 7 since the remaining cars satisfy the required "color-match" condition. However, removing cars 2, 5 and 7 from the sequence removes fewer cars and achieves the same "color-match" condition.

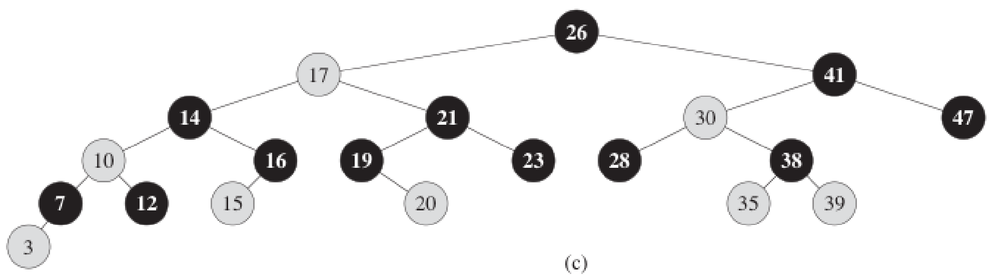
   Design an algorithm that takes as input an initial sequence of toy cars with their front and rear colors, and outputs the **least** number of cars to remove from this lineup so that the resulting sequence of cars has the required color property mentioned above. Note that you can only remove cars from the lineup, but your friend will not allow you to place them back in a different position or reorder the remaining cars in any way.

[Blank page.]

2. [37] You have been asked by your insane boss to design an augmented RB-tree that (a) stores integer key values in its nodes and (b) efficiently answers a sequence of queries from the following set:

- RB-INSERT(T, z) – inserts node z in tree T
- RB-DELETE(T, z) – deletes mode z from tree T
- OS-SOODD(x, y) – given node x, computes the number of odd values stored in the subtree rooted at y that are smaller than the key stored in node x.

You may assume that the tree has no duplicate key values. If your current tree looks like the RB-tree shown below and node x is the node with key value 20, then OS-SOODD$(x, \text{ROOT}(T))$ should return 5 because the key values 3, 7, 15, 17 and 19 are the only 5 odd numbers smaller than 20 stored in the tree. If y points to the node with key value 21, then OS-SOODD(x, y) should return 1 since only key value 19 is counted.



(c)

(a) [5] State what **augmented** piece(s) of information you will store in each node (besides KEY, LEFT, RIGHT, COLOR) in order to facilitate the above queries.

(b) [2] Explain very briefly why this information can be maintained in the presence of inserts and deletes in the RB-tree.

(c) [30] Write down an efficient algorithm for OS-SoOdd(x, y). The classic OS-Rank from class is provided for your benefit. Feel free to directly modify the pseudocode below.

OS-Rank(x, y)
▷ Finds rank of node x in subtree rooted at node y
▷ Recursive version discussed in class
▷ RB tree stores KEY, LEFT, RIGHT, COLOR in each node
▷ Every node is augmented with SIZE information

$r \leftarrow$ SIZE[LEFT[y]] $+ 1$
if $(x == y)$ **then return** r
else if (KEY[x] $<$ KEY[y]) **then return** OS-Rank(x, LEFT[y])
else return $r +$ OS-Rank(x, RIGHT[y])
end if
