CS210A: Data Structures and Algorithms
Mid semester exam
17 February, 2016

Note: For the questions where two choices are given, attempt **exactly one** of the two choices. If you attempt both, you will get zero marks for that question. Therefore, if you are not fully sure about the answer for the difficult one, you should attempt the easier one. Please keep this in mind and be very careful while deciding about the choice you should attempt. There is neither any need to analyse time complexity nor any need to prove correctness of any algorithm that you design in this exam.

1. (*marks=10*) Provide tightest possible upper bounds for $T(n)$ for each of the following recurrences for asymptotically large value of n. There is **no need** to provide any explanation. Just write the answer in big-Oh notation for each recurrence in your answer sheet.

   (a) $T(n) = 4n + n^{3/4}T(n^{1/4})$
   (b) $T(n) = n^2 + 16T(n/4)$
   (c) $T(n) = 100 + 3T(n/3)$
   (d) $T(n) = n + T(n/4) + T(3n/5) + T(n/7)$
   (e) $T(n) = n/\log^2 n + 2T(n/2)$

2. (*marks = 6*)
   We have to carry out implementation of a queue of positive numbers using an array $Q[0..n]$ of size $n+1$. Though the number of queue operations need not be bounded, you may assume that at any moment of time, there will never be more than n elements in the queue. You have to make use of **only** two global variables F and E of type integers defined as follows. If the queue is non-empty, F should store the index of first element of the queue. E should store the index of the vacant slot where the next element will be enqueued. In the beginning, queue is empty and we set $E = F = 0$. Write **neat pseudocode** for the following queue operations. Each queue operation should take $O(1)$ time only.

   (a) Enqueue(Q, x) //This function should place x at the end of the queue.
   (b) Dequeue(Q) // This function should return the first element from the queue if queue is non-empty, and 0 otherwise.
   (c) IsEmpty(Q) // This function should return 1 if the queue is empty and 0 otherwise.

   [Note: The implementation asked in this question might be a bit different from the one we discussed in the class. So please use your intelligence instead of just pasting the answers from your memories of the lecture slides.]

3. (a) (*marks = 3*) Write a **neat pseudocode** for an algorithm $BHeight(r)$ that computes the black height of a red-black tree rooted at r. The algorithm has to be most efficient asymptotically.
   (b) (*marks=3*) Write a **neat pseudocode** for an algorithm $Inorder(r)$ that prints the keys stored in a binary search tree rooted at r in the increasing order of their values. The algorithm has to be most efficient asymptotically.

4. Attempt exactly one of the following questions. You may describe the algorithm as a pseudocode or in plain English. However, it must be clear, unambiguous, and complete.

   (a) (*marks =8*) There is a sorted array A storing n distinct positive integers. You are given three positive integers a, b, and c. Design an $O(n)$ time algorithm to determine if there exist any two distinct integers $y, z \in A$ such that $a^2 = by + cz$.

   (b) (*marks =5*) There is a sorted array A storing n distinct positive integers. You are given a positive integer x. Design an $O(n \log n)$ time algorithm to determine if there exist any two distinct integers $y, z \in A$ such that $x = y + z$.

5. Attempt exactly one of the following questions.

Note: In the answer sheet, you should write the answer for each blank against its line number. Each blank is a single statement or a single (arithmetic or Boolean) expression. Please keep this in mind while writing the answers in the answer sheet.

(a) (*marks=10*)

There is a $n \times n$ Matrix $M[0..n-1, 0..n-1]$ such that each entry is 0 or 1. Moreover, in each row, all zeros appear before all ones. The numbers of zeros may be different in different rows. The following is the pseudocode of the fastest possible algorithm for computing the number of zeros in the row with maximum number of zeros. You have to fill in the blanks suitably.

Algorithm 1: Max-zeros(M)

```
1  i ← .....;
2  j ← .....;
3  max ← .....;
4  while ..... do
5  |  while ..... and ..... do
6  |  |  .....;
7  |  end
8  |  if (..... < j) then
9  |  |  .....;
10 |  end
11 |  i++;
12 end
13 return max;
```

(b) (*marks=5*)

Fill in the blanks the following pseudocode of computing any local-minima in an array $A[0..n-1]$ of n distinct numbers. This algorithm was discussed in the class. Assume for simplicity that $A[0] = A[n-1] = \infty$ and $n > 3$. The algorithm is supposed to return an index of A storing a local minima.

Algorithm 2: LocalMinima(A)

```
1  L ← .....;
2  R ← .....;
3  found ← FALSE;
4  while ..... do
5  |  mid ← .....;
6  |  if ..... and ..... then
7  |  |  found ← TRUE;
8  |  else
9  |  |  if ..... < ..... then
10 |  |  |  .....
11 |  |  else
12 |  |  |  .....;
13 |  |  end
14 |  end
15 end
16 return .....;
```

6. Please see the red-black tree shown in Figure 1. Attempt only one of the following problems on this red black tree. There is no need to show intermediate steps. Just draw the final red-black tree. Do not forget to assign color to the nodes suitably in your answer.

(a) (marks=10) Draw neatly the final red-black tree after deleting key 75.

(b) (marks=6) Draw neatly the final red-black tree after inserting 41.

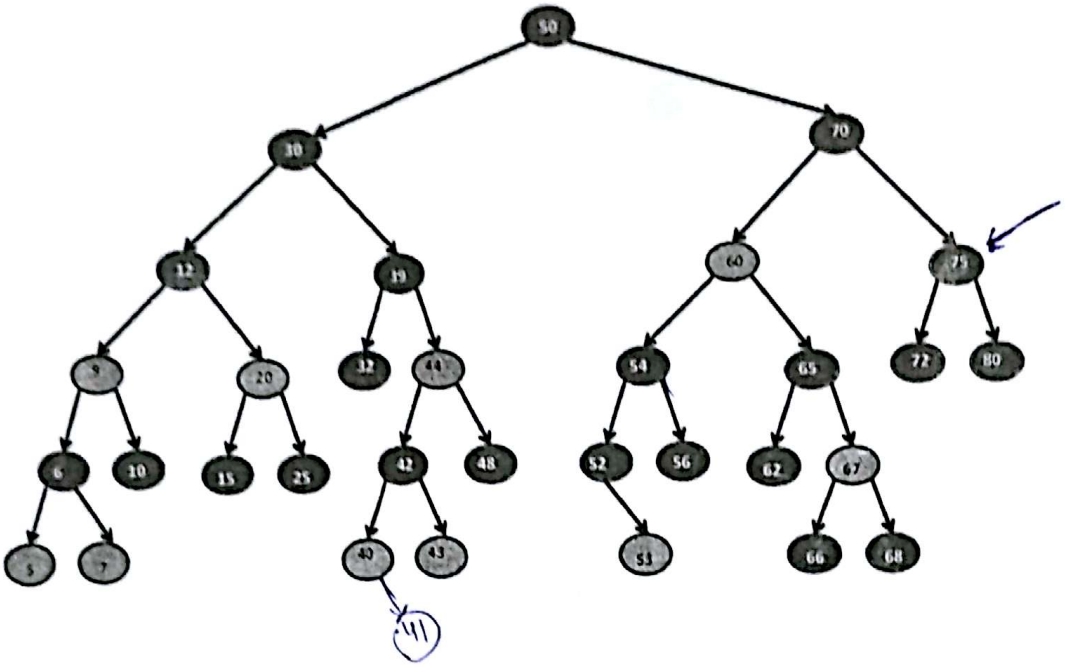

Figure 1: The red-black tree for Question 6. The leaf nodes, which are null nodes, are not shown here. The grey colored nodes are the red-nodes. Of course, the black colored nodes are indeed the black-nodes.