Question - 1

RBT

SCORE: **6.25 points**

Which of the following is true about Red Black Trees?

- ( ) At least one child of every black node is red
- ( ) The root may be red
- ( ) A leaf node may be red
- (•) None of the above.

Question - 2

True or False

SCORE: **6.25 points**

Is the following statement true? A Red-Black Tree which is also a perfect Binary Tree can have all black nodes.

- (•) True
- ( ) False

Question - 3

Time Complexity

SCORE: **6.25 points**

What is the worst case time complexity guarantee for search, insert and delete operations in a Binary Search Tree?

- (•) O(n) for all
- ( ) O(log n) for all
- ( ) O(log n) for search and insert, O(n) for delete
- ( ) O(log n) for search, O(n) for insert and delete

Question - 4

2-3 Trees

SCORE: **6.25 points**

Which of the following is true for 2-3 trees?

- [ ] Every node can have 2 or 3 keys
- [x] Each node can have 2 or 3 children depending on number of keys in node.
- [x] Every path from root to null link has same length.
- [ ] Every path from root to null link has same number of keys.

Question - 5

2-3 trees traversal

SCORE: **6.25 points**

Complexity of search, insert or delete in 2-3 trees is in order of:

- [x] lg N
- [ ] N
- [ ] N* lg N
- [ ] 0.5* N

Question - 6

Count the nodes

SCORE: **6.25 points**

A balanced 2-3 tree of depth **h** can store

- [x] between 2^h-1 and 3^h-1 nodes.
- [ ] only less than 2^h-1 nodes.
- [ ] between 2^h and 3^h nodes.
- [ ] only less than 3^h-1 nodes.

Question - 7

RBT rules

SCORE: **6.25 points**

Which of the folloing is true for RBT:

- [x] No node has two red links connected to it.
- [x] In the red link larger key is root.
- [x] Red links lean left.
- [ ] Number of red link is equal to number of black links.
- [x] Every path from root to null link has the same number of black links.
- [ ] All of the above.

Question - 8

SCORE: **6.25 points**

Operation on RBT

What is the below pseudo code trying to do, where pt is a node pointer and root pointer.

```
redblack(Node root, Node pt) :
  if (root == NULL)
     return pt

  if (pt.data < root.data)
  {
      root.left  =   redblack(root.left, pt);
      root.left.parent = root
  }
  else if (pt.data > root.data)
  {
      root.right = redblackt(root.right, pt)
      root.right.parent = root
  }
 return root
```

- (●) insert a new node
- ( ) delete a node
- ( ) search a node
- ( ) count the number of nodes