A. CIRCUIT_SIZE as a function of MAX_DEPTH is:

```
#define CIRCUIT_SIZE ((1<<(MAX_FUNC_DEPTH))-1)
```

B. The index of the first element in the last row is:

```
int firstNodeIdxInLastRow = ((CIRCUIT_SIZE) >> 1);
```

C. The data structure to represent a node can be:

```
struct Node {
        int type;      // X, CONST, INV, XOR
        double constVal;  // used for CONST only
      } Circuit [CIRCUIT_SIZE];
```

We can also define the following for better readability:

```
enum {TYPE_X, TYPE_CONST, TYPE_INV, TYPE_XOR };
#define NUM_GATE_TYPES (TYPE_XOR+1)
```

D. The *perturbTree* function can be implemented as follows (function *random(int r)* generates an integer random number between 0 and r):

```
void perturbTree()
{
   int i;
   int firstNodeIdxInLastRow = (CIRCUIT_SIZE >> 1);

   i = random(CIRCUIT_SIZE);

   if (Circuit[i].type == TYPE_CONST) { // const node.
                          // only change val w/o changing type?
      if (random(2) == 1) {
         Circuit[i].constVal = random(11)*0.1;
         return;
      }
   }

   // change node type
   if (i < firstNodeIdxInLastRow) // internal node, any type OK
      Circuit[i].type = random(NUM_GATE_TYPES);
   else {
      int r = random(2);
      if (r==0)
            Circuit[i].type = TYPE_X;
      else Circuit[i].type = TYPE_CONST;
      }

   if (Circuit[i].type == TYPE_CONST) { // const input
      Circuit[i].constVal = random(11) * 0.1;
}
```

E. The *costOfTree* function can be implemented using a loop, in which X changes from 0.0 to 1.0 with a step of 0.1, and at each point, the tree is evaluated and compared against F_TARGET. The function to evaluate a point X is a recursive function that is called on the root node.

```
double costOfTree ()
{
   double x, mse=0.0;
   for (x=0.0 ; x<1.0001 ; x+=0.1) {
      double correctY = F_TARGET(x);
      double approxY = evaluateApproxFunc(x);
      double errorSq = (correctY - approxY) * (correctY - approxY);
      mse += errorSq;
      }
   return mse / 11.0;
}



// Evaluate the function at the subtree at "rootNodeIndex" which
// has level "level", with the value X locked at "x".
double evaluateApproxFunc(double x, int rootNodeIndex=0,
                          int level=0)
{
   if (Circuit[rootNodeIndex].type == TYPE_X) { // X
      return x;
   }
   if (Circuit[rootNodeIndex].type == TYPE_CONST) { // const
      return Circuit[rootNodeIndex].constVal;
   }

   int leftChildIndex = (rootNodeIndex << 1) + 1;
   int rightChildIndex = (rootNodeIndex << 1) + 2;

   if (Circuit[rootNodeIndex].type == TYPE_INV) { // inv
      double childVal = evaluateApproxFunc(x, leftChildIndex,
                                           level+1);
      return 1.0 - childVal;
   }

   // the remaining functions have two inputs
   double leftVal = evaluateApproxFunc(x, leftChildIndex, level+1);
   double rightVal= evaluateApproxFunc(x, rightChildIndex, level+1);

   if (Circuit[rootNodeIndex].type == TYPE_XOR) { // xor
      double retVal = leftVal*(1-rightVal) + rightVal*(1-leftVal);
      return retVal;
   }

   // node type error
   return -1000.0;
}
```