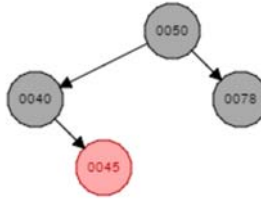Name: _Olivia Powers_   Date: _4/18/20_

Point values are assigned for each question.   Points earned: ____ / 74, ____ %

1. Show how the red-black tree would look after inserting a node with the key 0043. Use the document on Moodle that explains the insertion process succinctly. List the case you applied (i.e. 1, 2a, 3b), and write the steps you took to fix the tree (also listed in the document).

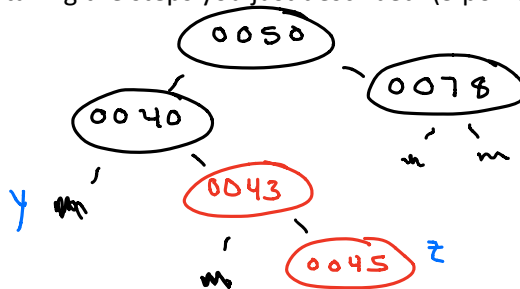a) Draw the tree after a regular binary search tree insertion. (3 points)
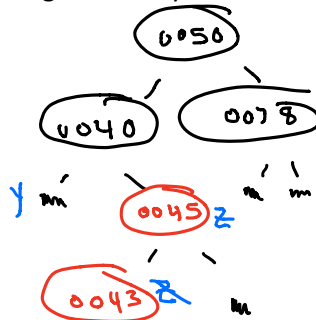
b) Which property is violated? (3 points) _property 4_
_____

Case seen after regular binary search tree insertion: (3 points) _2 b_

Steps taken to fix the tree: (3 points)
Set z to 0045 (the parent of 43), perform a
right-rotation of z (43)

Draw the tree after taking the steps you just described. (3 points)

c) Which property is violated now? (3 points) _property 4_
_____

Case seen after first fixup: (3 points) _3b_

Steps taken to fix the tree: (3 points) _Set the color of z's parent (43) to black, set z's grandparent (40) to red, perform a left rotation on z's grand parent (40)_

Draw the tree after taking the steps you just described. (3 points)

```
              (0050)
              /      ‾
          (0043)   (0078)
          /    \
      (0040)  (0045)
```
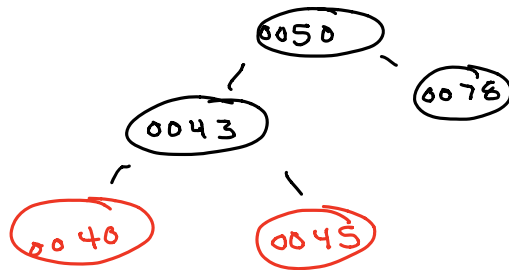
2. Draw the 2-3 tree after inserting each of the following keys. Redraw the tree for each part.
   a) 50 (1 point)

   (50)

   b) 76 (1 point)

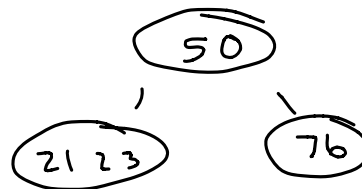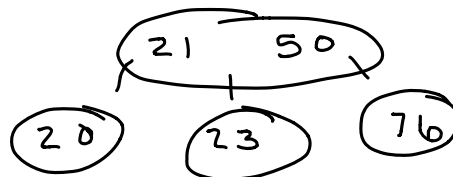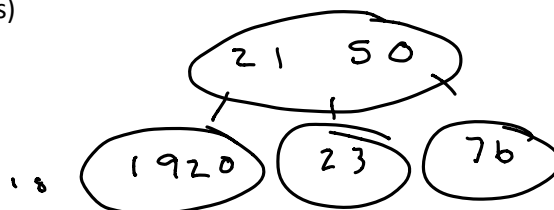   (50   76)

   c) 23 (3 points)

   ```
        (50)
        /    \
     (23)   (76)
   ```

   d) 21 (3 points)

   ```
         (50)
         /    \
     (21 23)  (76)
   ```

   e) 20 (3 points)

   ```
       (21   50)
       /   |    \
    (20) (23)  (76)
   ```

   f) 19 (3 points)

   ```
          (21   50)
          /   |     \
   (19 20) (23)  (76)
   ```

   g) 18 (3 points)

   ```
              (21)
              /    \
          (19)     (50)
          /   \      /  \
       (18) (20)  (23) (76)
   ```
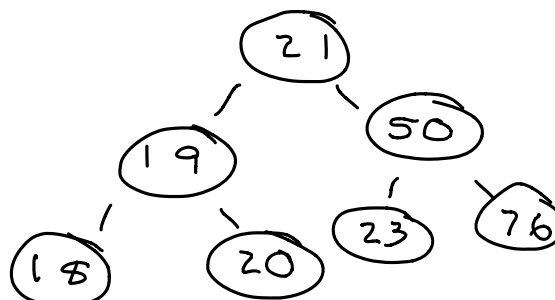
3. Read pages 241-242 in the textbook. Using that information, write pseudocode for computing the LCM of an array A[1..n] of integers. You may assume there is a working gcd() function. (6 points)

   ALGORITHM *LCM*(A[1..n]):

   // Computes the least common multiple of all the integer in array A

   ✳ written below ✳

4. Horner's method: *✳ work shown below ✳*
   $$p(x) = 4x^4 + 5x^3 - 2x^2 - 4x + 7$$

   a. Repeatedly factor out x in the following polynomial so that you can apply Horner's method. Write your expression for $p(x)$. (5 points)

   $$= x(x(x(4x+5)-2)-4)+7$$

   b. Show values of the array P[0..n] as needed to apply Horner's method. (3 points)

   $$[7, -4, -2, 5, 4]$$

   c. Apply Horner's method to evaluate the polynomial at $x = 2$. Make a table as we did in class showing the values x, p, n, and i, and then state your final answer for $p(2)$. (5 points)

x	p	n	i
2	4	4	
	13		3
	24		2
	44		1
	95		0

   $p(2) = \underline{95}$

   d. Use **synthetic** (not long) **division** to divide $p(x)$ by $x - 2$ to check your work. Be sure to show your work. (5 points)

   ✳ written below ✳

5. Rewrite the *LeftRightBinaryExponentiation* algorithm on page 237 in the textbook to work for $n = 0$ as well as any positive integer. *No credit will be given for answers that simply start with an if statement for n = 0.* (6 points)

   ALGORITHM *LeftRightBinaryExponentiation*(a, b(n)):

   // Computes a^n

   ✳ written below ✳

3) Algorithm LCM (A[1...n]) {
   product ← 1;
   for n ← 0 up until A. size do
   {
     product ← A[n] * product
   }
   }
   // LCM = product / gcd

   return product / gcd(A[1....n])

4) $p(x) = 4x^4 + 5x^3 - 2x^2 - 4x + 7$

a) = x(4x³ + 5x² - 2x - 4) + 7
    = x(x(4x² + 5x - 2) - 4) + 7
    = x(x(x(4x + 5) - 2) - 4) + 7

b) [7, -4, -2, 5, 4]

d)

$$x - 2 \overline{\smash{\big)}\, 4x^4 + 5x^3 - 2x^2 - 4x + 7}$$

$$2 \overline{\smash{\big)}\begin{array}{ccccc} 4 & 5 & -2 & -4 & +7 \\ \downarrow & 8 & 26 & 48 & 88 \\ \hline 4 & 13 & 24 & 44 & 95 \end{array}}$$

$$4x^3 + 13x^2 + 24x + 44$$

5) Algorithm LefRight Binary Exponentiation $(a, b(n))$:

1) checks to see if the #number is > 0

```
check ← false;
product ← a;
for i ← I-1 downto 0 do
{
    product ← product * product
    if b_i = 1 {
        check = true
        product ← product * a
    }
}
if check = false return 1
else return product
```