FALL 2019 - CS 303 Algorithms and Data Structures
Exam-2

50 points. 75 minutes. Closed book/notes.   Name: ___________________________

1. For each of the following program fragments, determine the time complexity using the $Big\ O$ notation. Explain your analysis next to each program fragment **[2*3 = 6 points]**

	Program Fragment	Time Complexity ($\mathrm{Big}\ O$ Notation)	Explanation

1	`for (int i=0; i<n; i++){`
	`    for (int j=0; j<n*n; j++){`
	`        a = a + i + j;`
	`    }`
	`}`		

2	`for (int i = 1; i < n; i *= 2)`
	`{`
	`a = a + i;`
	`}`		

3	`If (a==b){`
	`for (int i=0; i<3*n; i++){`
	`    for (int j=0; j<2*n; j++){`
	`        a = a + i + j; }`
	`    }`
	`}`
	`Else {`
	`  int a=0,i=n;`
	`   while(i>0){`
	`       a=a+i;`
	`       i=i/2;`
	`    }`
	`}`		

2. What is the Time Complexity of the following algorithms? **[10 points]**

Sorting Algorithm	**Best Case Time Complexity**	**Worst Case Time Complexity**
a. Insertion Sort		
b. Heap Sort		
c. Merge Sort		
d. Quick Sort		
e. Quick Sort Median of Three		

3. Given an input sequence of character keys: **[123,96,145,250,17,33,21,124,121,130]** illustrate the creation of a **binary search tree** if these keys are inserted in the above order. [**5 points**] *(You don't have to draw each steps, you can draw the final BST if you want)*

Show what the tree would look like after the following changes are made to this tree in the following order (make sure to include all the intermediate steps and explain each step): [**6 points**]

a. **Delete** node with value **33**
b. **Delete** node with value **145**
c. **Insert** node with value **129**

The algorithm for deleting a node in a binary search tree is given below:

```
TREE-DELETE(T, z)
  if z.left == NIL
      TRANSPLANT(T, z, z.right)          // z has no left child
  elseif z.right == NIL
      TRANSPLANT(T, z, z.left)           // z has just a left child
  else // z has two children.
      y = TREE-MINIMUM(z.right)          // y is z's successor
      if y.p ≠ z
          // y lies within z's right subtree but is not the root of this subtree.
          TRANSPLANT(T, y, y.right)
          y.right = z.right
          y.right.p = y
      // Replace z by y.
      TRANSPLANT(T, z, y)
      y.left = z.left
      y.left.p = y
```

```
TRANSPLANT(T, u, v)
  if u.p == NIL
      T.root = v
  elseif u == u.p.left
      u.p.left = v
  else u.p.right = v
  if v ≠ NIL
      v.p = u.p
```

4. Given the following character keys: **[120, 38, 165, 190, 185, 200, 250]** show the steps involved in creating a **red-black tree**. Make sure to draw the tree after each key is inserted in the given sequence. [**10 points**]

The pseudo-code for red-black tree insert is given below.

RB-INSERT(T, z)	LEFT-ROTATE(T, x)

$y = T.nil$
$x = T.root$
while $x \neq T.nil$
    $y = x$
    if $z.key < x.key$
        $x = x.left$
    else $x = x.right$
$z.p = y$
if $y == T.nil$
    $T.root = z$
elseif $z.key < y.key$
    $y.left = z$
else $y.right = z$
$z.left = T.nil$
$z.right = T.nil$
$z.color =$ RED
RB-INSERT-FIXUP(T, z)
	$y = x.right$ // set y
	$x.right = y.left$ // turn y's left subtree into x's right subtree
	if $y.left \neq T.nil$
	    $y.left.p = x$
	$y.p = x.p$ // link x's parent to y
	if $x.p == T.nil$
	    $T.root = y$
	elseif $x == x.p.left$
	    $x.p.left = y$
	else $x.p.right = y$
	$y.left = x$ // put x on y's left
	$x.p = y$

RB-INSERT-FIXUP(T, z)

```
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK                  // case 1
            y.color = BLACK                    // case 1
            z.p.p.color = RED                  // case 1
            z = z.p.p                          // case 1
        else if z == z.p.right
                z = z.p                        // case 2
                LEFT-ROTATE(T, z)              // case 2
            z.p.color = BLACK                  // case 3
            z.p.p.color = RED                  // case 3
            RIGHT-ROTATE(T, z.p.p)             // case 3
    else (same as then clause with "right" and "left" exchanged)
T.root.color = BLACK
```

5. What is a Binary Tree Traversal? **[2 points].** How many different traversal methods exist? Explain them **[4 points]**

6. Explain the **similarities** and **differences** between **BFS** and **DFS** in terms of *algorithms* and *data structures*. **[7 points]**