CS 201 Data Structures Library Phase 2 Due 11/1

Phase 2 of the CS201 programming project, we will be built around a balanced binary search tree. In particular, you should implement a left-leaning red-black tree for the class RBTree.

The public methods of your class should include the following (elmtype indicates the type from the template):

Function	Description	Runtime
RBTree();	Default Constructor. The tree should be empty	O(1)
RBTree(keytype k[], valuetype V[], int s);	For this constructor the tree should be built using the arrays K and V containing s items of keytype and valuetype.	O(s lg s)
~RBTree();	Destructor for the class.	O(n)
valuetype * search(keytype k);	Traditional search. Should return a pointer to the valuetype stored with the key. If the key is not stored in the tree then the function should return NULL.	O(lg n)
void insert(keytype k, valuetype v);	Inserts the node with key k and value v into the tree.	O(lg n)
int remove(keytype k);	Removes the node with key k and returns 1. If key k is not found then delete should return 0.	O(lg n)
int rank(keytype k);	Returns the rank of the key k in the tree. Returns 0 if the key k is not found.	O(lg n)
keytype select(int pos);	Order Statisics. Returns the key of the node at position pos in the tree. Calling with pos = 1 should return the smallest key in the tree, pos = n should return the largest.	O(lg n)
void split(keytype k, RBTree<keytype,valuetype>& T1, RBTree<keytype,vlauetype>& T2);	Splits the tree into T1 and T2 based on key k This function will be worth 10 bonus points if implemented on O(lg n) time.	O(lg n)
int size();	returns the number of nodes in the tree.	O(1)
void preorder();	Prints the keys of the tree in a preorder traversal.	O(n)
void inorder();	Prints the keys of the tree in an inorder traversal.	O(n)
void postorder();	Prints the keys of the tree in a postorder traversal.	O(n)

Your class should include proper memory management, including a destructor, a copy constructor, and a copy assignment operator.