```rust
use std::cmp::Ordering;

#[derive(Debug)]
pub struct Content {
    pub i: i32,
    pub s: String,
}

impl Content {
    pub fn new(i: i32, s: String) -> Content {
        Content { i, s }
    }
}

// Implement PartialOrd and PartialEq for Content
impl PartialEq for Content {
    fn eq(&self, other: &Self) -> bool {
        self.s.len() == other.s.len()
    }
}

impl PartialOrd for Content {
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {
        Some(self.s.len().cmp(&other.s.len()))
    }
}

// Define Node and Tree structs
#[derive(Debug)]
struct Node<T> {
    elem: T,
    left: TreeLink<T>,
    center: TreeLink<T>,
    right: TreeLink<T>,
}

impl<T> Node<T> {

    pub fn new(elem: T) -> Node<T> {
        Node {
            elem,
            left: None,
            center: None,
            right: None,
        }
    }
}

#[derive(Debug)]
pub struct Tree<T> {
    root: TreeLink<T>,
}

type TreeLink<T> = Option<Box<Node<T>>>;

impl<T: PartialOrd> Tree<T> {
    // [1] Create a new empty tree
    pub fn new() -> Self {
        Tree { root: None }
    }

    // [6] Add a node to the tree
    pub fn add(&mut self, el: T) {
        self.root = Self::add_to_node(self.root.take(), el);
    }

    fn add_to_node(node: TreeLink<T>, el: T) -> TreeLink<T> {
        match node {
            None => Some(Box::new(Node::new(el))),
            Some(mut boxed_node) => {
                if el < boxed_node.elem {
                    boxed_node.left = Self::add_to_node(boxed_node.left.take(), el);
                } else if el > boxed_node.elem {
                    boxed_node.right = Self::add_to_node(boxed_node.right.take(), el);
                } else {
                    boxed_node.center = Self::add_to_node(boxed_node.center.take(), el);
                }
                Some(boxed_node)
            }
        }
    }

    // [4] Count how many nodes have content < el
    pub fn howmany_smaller(&self, el: T) -> i32 {
        Self::count_smaller(&self.root, &el)
    }

    fn count_smaller(node: &TreeLink<T>, el: &T) -> i32 {
        match node {
            None => 0,
            Some(boxed_node) => {
                let mut count = 0;
                if boxed_node.elem < *el {
                    count += 1;
                }
                count += Self::count_smaller(&boxed_node.left, el);
                count += Self::count_smaller(&boxed_node.center, el);
                count += Self::count_smaller(&boxed_node.right, el);
                count
            }
        }
    }
}

pub fn main(){
    let mut t: Tree<i32> = Tree::new();
    println!("{:?}",t);
}
```