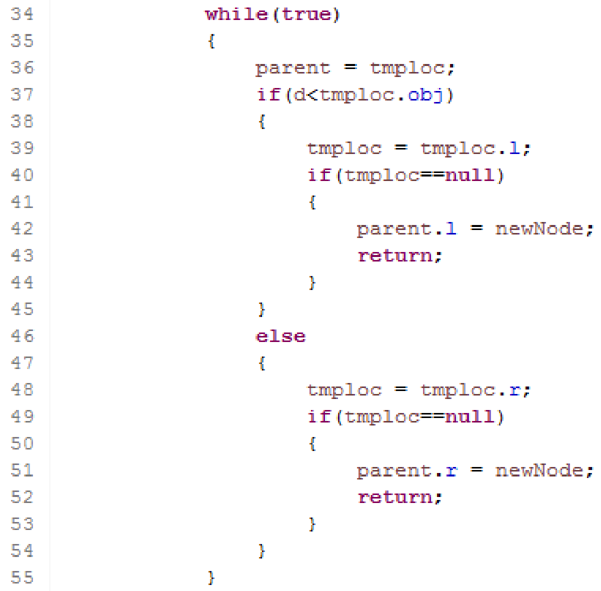
Problem 2

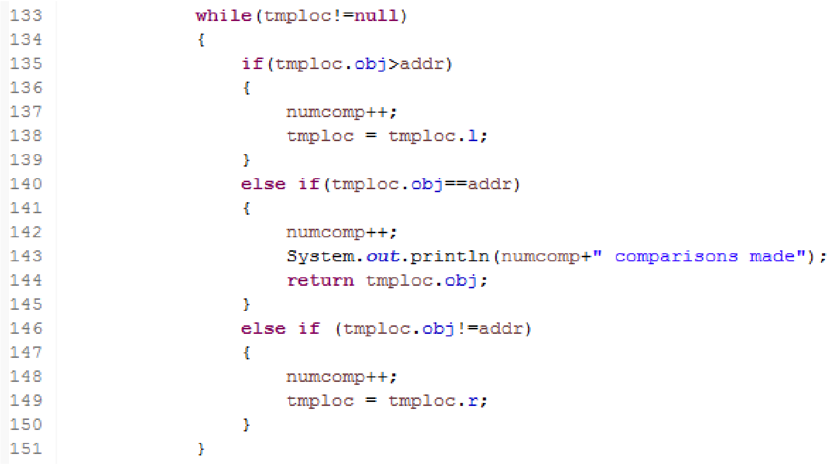
The primary functions in the BST implementation are put (both integer and integer array versions), search, balanceTreeTwo, and sortedTree. Other methods are fairly minimal in their complexity.

Put depends on the height of the tree, which is logn for a standard BST. The array version depends on the height and the size of the input array, a. So the complexity of the array version of the put method is alogn.



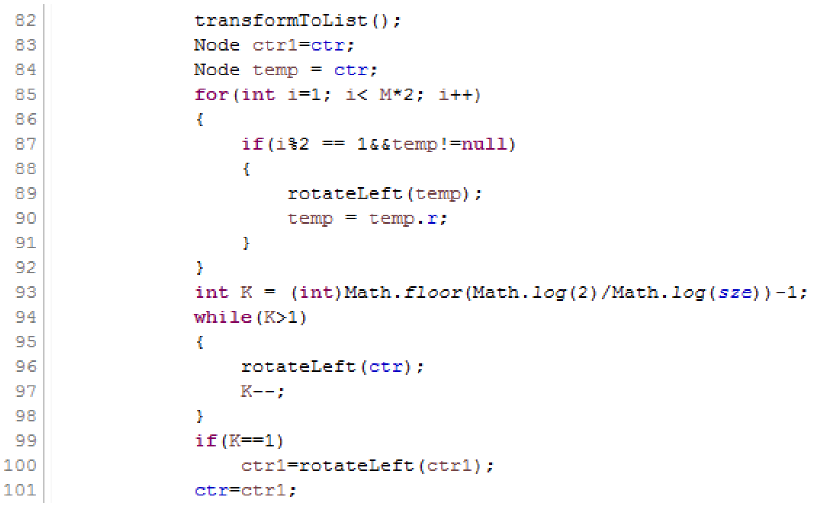
```
while(true)
{
    parent = tmploc;
    if(d<tmploc.obj)
    {
        tmploc = tmploc.l;
        if(tmploc==null)
        {
            parent.l = newNode;
            return;
        }
    }
    else
    {
        tmploc = tmploc.r;
        if(tmploc==null)
        {
            parent.r = newNode;
            return;
        }
    }
}
```

Search depends on the height, as logn. Once the element's position is arrived at, the operations that follow are O(1).



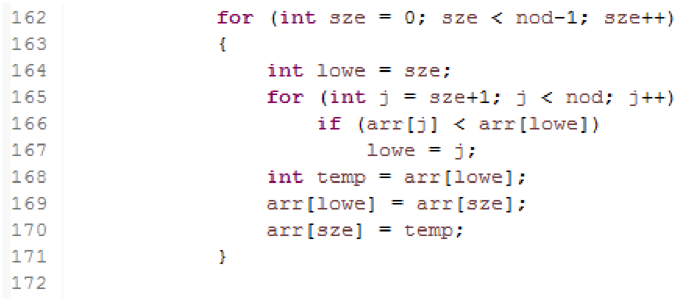
```
while(tmploc!=null)
{
    if(tmploc.obj>addr)
    {
        numcomp++;
        tmploc = tmploc.l;
    }
    else if(tmploc.obj==addr)
    {
        numcomp++;
        System.out.println(numcomp+" comparisons made");
        return tmploc.obj;
    }
    else if (tmploc.obj!=addr)
    {
        numcomp++;
        tmploc = tmploc.r;
    }
}
```

balanceTreeTwo has a loop that depends on value M, which is an integer, so complexity is O(1). It does however call transformToList which has nested loops, both of which depend on the height of the tree – converting it into a list of length k, where k is the number of elements in the BST. Thus balanceTreeTwo has complexity O(k).



```
transformToList();
Node ctr1=ctr;
Node temp = ctr;
for(int i=1; i< M*2; i++)
{
    if(i%2 == 1&&temp!=null)
    {
        rotateLeft(temp);
        temp = temp.r;
    }
}
int K = (int)Math.floor(Math.log(2)/Math.log(sze))-1;
while(K>1)
{
    rotateLeft(ctr);
    K--;
}
if(K==1)
    ctr1=rotateLeft(ctr1);
ctr=ctr1;
```

sortedTree similarly has nested loops, which both depend on the size of the BST, thus giving complexity O(k^2), where k is the number of items in the BST. Additionally, helper method drilDn is called, which is recursive and will run the length of the BST, adding another k, thus O(k^2+k), but since k is inconsequential compared to k^2, we leave O(k^2).



```
for (int sze = 0; sze < nod-1; sze++)
{
    int lowe = sze;
    for (int j = sze+1; j < nod; j++)
        if (arr[j] < arr[lowe])
            lowe = j;
    int temp = arr[lowe];
    arr[lowe] = arr[sze];
    arr[sze] = temp;
}
```