HOMEWORK 5
SHAHAD ALMUHIZI
436201525

Problem 1:

1.

```
private boolean areMirror(BTNode<T> t1, BTNode<T> t2){
if((t1== null) != (t2== null))
return false;
if(t1== null)
return true;
return(t1.data.equals(t2.data) && areMirror(t1.left, t2.right)
&& areMirror(t1.right, t2.left));}
```

2.

```
private void swap( BTNode <T > t){
if(t ==null)
return;

if(t.left!=null){
T val=t.data;
t.data=t.left.data;
t.left.data=val;}

else if(t.right!=null){
T val2=t.data;
t.data=t.right.data;
t.right.data=val2;}
swap(t.left);
swap(t.right);}
```

Problem 2:

1.

```
public static <T > LinkedList <T > collectLeaves(BT <T > bt){
LinkedList <T > list=new LinkedList<T>();
if (!bt.empty()){
bt.find(Relative.Root);
recCollectLeaves(bt,list);}
return list;}

private static <T> void recCollectLeaves(BT <T> bt, LinkedList
<T>list){
boolean flag=true;
if(bt.find(Relative.LeftChild)){
flag=false;
recCollectLeaves(bt,list);
bt.find(Relative.Parent);}
if(bt.find(Relative.RightChild)){
flag=false;
recCollectLeaves(bt,list);
bt.find(Relative.Parent);}
if(flag){
list.insert(bt.retrieve());}}
```

2.

```
public LinkedList<T> collectLeaves(){
LinkedList<T> list= new LinkedList<T>();
BTNode<T> p= root;
return collectLeaves(list,p);}

public LinkedList<T> collectLeaves(LinkedList<T> list, BTNode<T>
p){
if(p== null)
return list;
if(p.left==null&& p.right== null)
list.insert(p.data);
collectLeaves(list,p.left);
collectLeaves(list,p.right);
return list;}
```

Problem 3:

1.

```
public static boolean isBST(BT<Integer> bt){
bt.find(Relative.Root);
LinkedList<Integer> list= new LinkedList<Integer>();
isBST(bt,list);
list.findFirst();
Integer curr= null,p=null;
while(!list.last()){
curr=list.retrieve();
if(p!= null)
if(curr< p)
return false;
list.findNext();
p=curr;}
curr=list.retrieve();
if(p!= null)
if(curr<p)
return false;
return true;}

public static void isBST(BT<Integer> bt, LinkedList<Integer>
list){
if(bt.find(Relative.LeftChild)) {
isBST(bt, list);
bt.find(Relative.Parent);}
list.insert(bt.retrieve());
if(bt.find(Relative.RightChild)) {
isBST(bt, list);
bt.find(Relative.Parent);}}
```

2.

```
public static boolean find(BT<Integer>bt,int k ){
if(bt.empty())
return false;
Relative rel;
bt.find(Relative.Root);
do{
if(k==bt.retrieve())
return true;
if(k<bt.retrieve())
rel=Relative.LeftChild;
else
rel=Relative.RightChild;}
while(bt.find(rel));
return false;}
```

problem 4:

1.

```
private void swapData(intk){
BSTNode<T> p=root;
BSTNode<T> q=null;
while(p!= null&& p.key!= k) {
q=p;
if(k< p.key)
p= p.left;
else
p= p.right;}

if(p== null|| p== root)
return;

T temp= p.data;
p.data= q.data;
q.data= temp;}
```

2.

```
public int nbInRange(int k1, int k2) {
BSTNode<T> p= root;
return nbInRange(k1, k2, p);}

public int nbInRange(int k1, int k2, BSTNode<T> p){
if(p== null)
return 0;
if(p.key> k1&& p.key< k2)
return 1+nbInRange(k1, k2, p.left)+nbInRange(k1, k2, p.right);
if(p.key<= k1){
return 1+nbInRange(k1, k2, p.right);
else
return nbInRange(k1, k2, p.right);}

if(p.key==k2)
return 1+nbInRange(k1, k2, p.left);
else
return nbInRange(k1, k2, p.left);}
```

3.

```
public int deepestKey(BSTNode<T> t) {
int tmp= 0;
int k= 0;
int val= 0;
BSTNode<T> p= t;
LinkedStack<Integer> Levels= new LinkedStack<Integer>();
LinkedStack<BSTNode<T>> list= new LinkedStack<BSTNode<T>>();

while(p!= null) {
if(p.left== null&& p.right== null)
if(tmp< k) {
tmp= level;
val= p.key;}

if(p.right!= null) {
Levels.push(k+ 1);
list.push(p.right);}

if(p.left!= null) {
p= p.left;
k++;}
else{
if(!list.empty()) {
p=list.pop();
k= Levels.pop();}
else
p= null;}}
return val;}
```