AP® Computer Science AB
2004 Sample Student Responses

AB 4 A,

(a) Write the `TreePriorityQueue` method `peekMin`. Method `peekMin` returns the minimum data value in the priority queue.

Complete method `peekMin` below.

```
// precondition:  the priority queue is not empty
// postcondition: a data value of smallest priority is returned;
//                the priority queue is unchanged
public Object peekMin()
{  if(root==null)
     return null;

   TreeNode current= root;
   while(current !=null)
   {
      if(current.getLeft()!=null)      //move as far left as poss.
         current=current.getLeft(); //to find the smallest value

   }
   return ((Item)current.getValue()).getData();
}
```

GO ON TO THE NEXT PAGE.

(b) Write the `TreePriorityQueue` private method `addHelper`, which adds `obj` to the subtree rooted at `t`. If a node containing an `Item` with data `obj` is already in the subtree, `addHelper` increments the count; otherwise, `addHelper` adds a new node containing an `Item` with data `obj` and a count of 1 to the subtree, maintaining the binary search tree property. Method `addHelper` returns a reference to the root of the resulting subtree.

Complete method `addHelper` below.

```
// postcondition: obj has been added to the subtree rooted at t;
//                the resulting subtree is returned
private TreeNode addHelper(Comparable obj, TreeNode t)
{
    if(t==null) //check for null reference
        return t;
    Comparable data= ((Item)t.getValue()).getData();
    if(obj.equals(data)) //if equal, increment t's item
        ((Item)t.getValue()).increment(count();
    else if(obj.compareTo(data)<0)//obj goes left of data
    {
        if(t.getLeft == null)
            t.setLeft(new TreeNode(new Item(obj)));
        else
            addHelper(obj, t.getLeft);
    }
    else //if(obj.compareTo(data)>0) //obj goes right
    {
        if(t.getRight==null)
            t.setRight(new TreeNode(new Item(obj)));
        else
            addHelper(obj, t.getRight);
    }
    return t;
}
```

GO ON TO THE NEXT PAGE.

AB4 B1

(a) Write the `TreePriorityQueue` method `peekMin`. Method `peekMin` returns the minimum data value in the priority queue.

Complete method `peekMin` below.

```
// precondition:  the priority queue is not empty
// postcondition: a data value of smallest priority is returned;
//                the priority queue is unchanged
public Object peekMin()
{
    return ((Item) root.getValue()).getData();
}
```

(b) Write the `TreePriorityQueue` private method `addHelper`, which adds `obj` to the subtree rooted at `t`. If a node containing an `Item` with data `obj` is already in the subtree, `addHelper` increments the count; otherwise, `addHelper` adds a new node containing an `Item` with data `obj` and a count of 1 to the subtree, maintaining the binary search tree property. Method `addHelper` returns a reference to the root of the resulting subtree.

Complete method `addHelper` below.

```
// postcondition: obj has been added to the subtree rooted at t;
//                the resulting subtree is returned
private TreeNode addHelper(Comparable obj, TreeNode t)
{
    Item newItem = new Item(obj);

    if(t == null)
        t=new TreeNode(obj, null, null)
    else if(((Item) t.getValue()).getData().compareTo(obj) == 0)
      ((Item) t.getValue()).incrementCount();
    else
     {
        if( obj.compareTo(((Item) t.getValue()).getData()) < 0 )
            addHelper(obj, t.getLeft());
        else
            addHelper(obj, t.getRight());
     }

    return t;
}
```

AB 4 C1

(a) Write the `TreePriorityQueue` method `peekMin`. Method `peekMin` returns the minimum data value in the priority queue.

Complete method `peekMin` below.

```
// precondition:  the priority queue is not empty
// postcondition: a data value of smallest priority is returned;
//                the priority queue is unchanged
public Object peekMin()
{
   while( !isEmpty(root]
     root.getLeft();

   return  root.getData().getValue();
}
```

(b) Write the `TreePriorityQueue` private method `addHelper`, which adds `obj` to the subtree rooted at `t`. If a node containing an `Item` with data `obj` is already in the subtree, `addHelper` increments the count; otherwise, `addHelper` adds a new node containing an `Item` with data `obj` and a count of 1 to the subtree, maintaining the binary search tree property. Method `addHelper` returns a reference to the root of the resulting subtree.

Complete method `addHelper` below.

```
// postcondition: obj has been added to the subtree rooted at t;
//                the resulting subtree is returned
private TreeNode addHelper(Comparable obj, TreeNode t)
{   if(t == null)
      return null;
    if(obj.compareTo(t.getData().getValue()) == 0)
    {   t.incrementCount();
        return t;
    }

    else if(obj.compareTo(t.getData().getValue()) < 0)
    {t = addHelper(obj, t.getLeft());
    }

    else
    {t = addHelper(obj, t.getRight());
    }
    return t;
}
```