



AP[®] Computer Science AB 2004 Sample Student Responses

The materials included in these files are intended for noncommercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program[®]. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. This permission does not apply to any third-party copyrights contained herein. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here.

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,500 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT[®], the PSAT/NMSQT[®], and the Advanced Placement Program[®] (AP[®]). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit www.collegeboard.com

Copyright © 2004 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at apcentral.collegeboard.com.

- (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()).incrementCount();
    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.

- (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();
}
```

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)
```

```
{
    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;
}
```

GO ON TO THE NEXT PAGE.

- (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();
}
```

GO ON TO THE NEXT PAGE.

C₂

- (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.setLeft(addHelper(obj, t.getLeft()));
    }
    else
    {
        t.setRight(addHelper(obj, t.getRight()));
    }
    return t;
}
```

GO ON TO THE NEXT PAGE.