



## AP<sup>®</sup> Computer Science AB 2001 Sample Student Responses

**The materials included in these files are intended for non-commercial 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. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.**

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT<sup>®</sup>, the PSAT/NMSQT<sup>™</sup>, the Advanced Placement Program<sup>®</sup> (AP<sup>®</sup>), and Pacesetter<sup>®</sup>. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2001 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

Complete function ItemsToQueues below.

```
apvector<apqueue<int>> ItemsToQueues(const apvector<int> & L, int k)
// precondition: all values in L are positive;
//              k ≥ 0
// postcondition: Step 1 of the Radix sort algorithm for the digit
//              at position k has been completed
{
    apvector<apqueue<int>> QA(10);
    int i;
    for (i=0; i < L.length(); i++)
    {
        QA[GetDigit(L[i], k)].enqueue(L[i]);
    }
    return QA;
}
```

(b) Write function `QueuesToArray`, as started below. `QueuesToArray` corresponds to step 2 of each pass of the Radix sort algorithm, creating a new list from the values in the intermediate array of queues.

Complete function `QueuesToArray` below.

```
apvector<int> QueuesToArray(apvector<apqueue<int> > & QA, int numVals)
// precondition: QA.length() == 10; numVals is the total number of
//               integers in all the queues in QA
// postcondition: returns an apvector of length numVals that contains
//               the integers from QA[0] through QA[9] in the order
//               in which they were stored in the queues;
//               the queues in QA are empty
```

```
{
```

```
    apvector<int> partialSortList(numVals);
```

```
    int i;
```

```
    int count = 0;
```

```
    for (i = 0; i < 10; i++)
```

```
    {
```

```
        while (!QA[i].isEmpty())
```

```
        {
```

```
            partialSortList[count] = QA[i].front();
```

```
            QA[i].dequeue();
```

```
            count++;
```

```
        }
```

```
    }
```

```
    return partialSortList;
```

```
}
```

(c) Write function RadixSort, as started below.

In writing RadixSort, you may call functions GetDigit, ItemsToQueues, and QueuesToArray specified in parts (a) and (b). Assume that ItemsToQueues and QueuesToArray work as specified, regardless of what you wrote in parts (a) and (b).

Complete function RadixSort below.

```
void RadixSort(apvector<int> & L, int numDigits)
// precondition: L.length() > 0; all values in L are positive?
//               the largest value in L has numDigits digits;
// postcondition: L contains the same list of values, sorted in
//               nondecreasing order
```

```
{
```

```
    apvector<apqueue<int>> QA(10);
```

```
    int i;
```

```
    int count = L.length();
```

```
    for (i=0; i < numDigits; i++)
```

```
{
```

```
    QA = ItemsToQueues(L, i);
```

```
    L = QueuesToArray(QA, count);
```

```
}
```

```
}
```

Complete function ItemsToQueues below.

```
apvector<apqueue<int> > ItemsToQueues(const apvector<int> & L, int k)
// precondition: all values in L are positive;
//              k ≥ 0
// postcondition: Step 1 of the Radix sort algorithm for the digit
//              at position k has been completed
{
    apvector<apqueue<int> > temp,
    int j, k;
    for (int i = 0; i < L.length(); i++)
    {
        j = GetDigit(L[i], k);
        temp[j].enqueue(L[i]);
    }
    return temp;
}
```

(b) Write function `QueuesToArray`, as started below. `QueuesToArray` corresponds to step 2 of each pass of the Radix sort algorithm, creating a new list from the values in the intermediate array of queues.

Complete function `QueuesToArray` below.

```
apvector<int> QueuesToArray(apvector<apqueue<int> > & QA, int numVals)
// precondition: QA.length() == 10; numVals is the total number of
//               integers in all the queues in QA
// postcondition: returns an apvector of length numVals that contains
//               the integers from QA[0] through QA[9] in the order
//               in which they were stored in the queues;
//               the queues in QA are empty
```

```
{
    apvector<int> L;
    int i = 0;
    for (int j = 0; j < 9; j++)
    {
        if (!QA[j].isempty())
        {
            L[i] = QA[j].dequeue();
            i++;
            if (i == numVals - 1)
                return L;
        }
    }
    return L;
}
```

(c) Write function RadixSort, as started below.

In writing RadixSort, you may call functions GetDigit, ItemsToQueues, and QueuesToArray specified in parts (a) and (b). Assume that ItemsToQueues and QueuesToArray work as specified, regardless of what you wrote in parts (a) and (b).

Complete function RadixSort below.

```
void RadixSort(apvector<int> & L, int numDigits)
// precondition: L.length() > 0; all values in L are positive;
//               the largest value in L has numDigits digits;
// postcondition: L contains the same list of values, sorted in
//               nondecreasing order
```

```
for (int i = 0; i < numDigits; i++)
{
    apvector<apqueue<int>> QA;
    QA = ItemsToQueue(L, i);
    L = QueuesToArray(QA, L.length);
}
```

Complete function ItemsToQueues below.

```
apvector<apqueue<int> > ItemsToQueues(const apvector<int> & L, int k)
// precondition: all values in L are positive;
//               k ≥ 0
// postcondition: Step 1 of the Radix sort algorithm for the digit
//               at position k has been completed
{
```

```
    int c;
    for (int a=0; a < L.length(); a++)
```

```
    {
```

```
        for (int b=0; b < L.length; b++)
```

```
        {
```

```
            c = GetDigit(L[b], b);
```

```
            if (c == 0);
```

```
                L[b] = QA[0];
```

```
            else if (c == 1);
```

```
                L[b] = QA[1];
```

```
            else if (c == 2);
```

```
                L[b] = QA[2];
```

```
            else if (c == 3);
```

```
                L[b] = QA[3];
```

```
            else if (c == 4);
```

```
                L[b] = QA[4];
```

```
            else if (c == 5);
```

```
                L[b] = QA[5];
```

```
            else if (c == 6);
```

```
                L[b] = QA[6];
```

```
            else if (c == 7);
```

```
                L[b] = QA[7];
```

```
            else if (c == 8);
```

```
                L[b] = QA[8];
```

```
            else
```

```
                L[b] = QA[9];
```

```
        }
```

```
    }
```

```
}
```

(b) Write function `QueuesToArray`, as started below. `QueuesToArray` corresponds to step 2 of each pass of the Radix sort algorithm, creating a new list from the values in the intermediate array of queues.

Complete function `QueuesToArray` below.

```
apvector<int> QueuesToArray(apvector<apqueue<int> > & QA, int numVals)
// precondition: QA.length() == 10; numVals is the total number of
// integers in all the queues in QA
// postcondition: returns an apvector of length numVals that contains
// the integers from QA[0] through QA[9] in the order
// in which they were stored in the queues;
// the queues in QA are empty
```

```
{
    apvector<int> b;
    for(int a=0; a<QA; a++)
    {
        while(QA.dequeue())
        {
            QA[a] = b[a];
        }
    }
    return b;
}
```

(c) Write function RadixSort, as started below.

In writing RadixSort, you may call functions GetDigit, ItemsToQueues, and QueuesToArray specified in parts (a) and (b). Assume that ItemsToQueues and QueuesToArray work as specified, regardless of what you wrote in parts (a) and (b).

Complete function RadixSort below.

```
void RadixSort(apvector<int> & L, int numDigits)
// precondition: L.length() > 0; all values in L are positive;
//               the largest value in L has numDigits digits;
// postcondition: L contains the same list of values, sorted in
//               nondecreasing order
```

```
{
    for (int a=0; a < numDigits; a++)
    {
        ItemsToQueues(L, a);
        QueuesToArray(L, a);
    }
}
```