AP® COMPUTER SCIENCE A
2015 GENERAL SCORING GUIDELINES

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times, or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

(v) Array/collection access confusion (`[] get`)

(w) Extraneous code that causes side effect (*e.g., writing to output, failure to compile*)

(x) Local variables used but none declared

(y) Destruction of persistent data (*e.g., changing value referenced by parameter*)

(z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side effect (*e.g., precondition check, no-op*)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`, with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration (*e.g.,* `int[size] nums = new int[size];`)
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context; for example, "`ArayList`" instead of "`ArrayList`". As a counterexample, note that if the code declares "`Bug bug;`", then uses "`Bug.move()`" instead of "`bug.move()`", the context does **not** allow for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE A
2015 SCORING GUIDELINES

Question 1: Diverse Array

Part (a)	`arraySum`	**2 points**

Intent: *Compute and return sum of elements in 1D array* `arr`, *passed as parameter*

+1 Accesses all elements of `arr` (*no bounds errors on* `arr`)

+1 Initializes, computes, and returns sum of elements

Part (b)	`rowSums`	**4 points**

Intent: *Compute and return 1D array containing sums of each row in the 2D array* `arr2D`, *passed as parameter*

+1 Constructs correctly-sized 1D array of ints

+1 Accesses all rows in `arr2D` (*no bounds errors on* `arr2D`)

+1 Computes sum of row in `arr2D` using `arraySum` and assigns to element in 1D array

+1 Returns 1D array where kth element is computed sum of corresponding row in 2D array for all rows

Part (c)	`isDiverse`	**3 points**

Intent: *Determine whether* `arr2D`, *passed as parameter, is diverse*

+1 Computes and uses array of row sums from `arr2D` using `rowSums`

+1 Compare all and only pairs of row sums for equality (*No bounds errors on row sums array; point not awarded if no adjustment when compares any row sum with itself*)

+1 Returns `true` if all compared row sums are different and `false` otherwise (*point not awarded for immediate return*)

Question-Specific Penalties

-1 (g) Uses `getLength/getSize` for array size

-1 (y) Destruction of persistent data (`arr` *or* `arr2D`)

AP® COMPUTER SCIENCE A
2015 CANONICAL SOLUTIONS

Question 1: Diverse Array

Part (a):

```
public static int arraySum(int[] arr){
       int sum=0;
       for (int elem : arr){
            sum += elem;
       }
       return sum;
}
```

Part (b):

```
public static int[] rowSums(int[][] arr2D){
        int [] sums=new int[arr2D.length];
        int rowNum=0;
        for(int[] row : arr2D){
            sums[rowNum]=arraySum(row);
            rowNum++;
        }
        return sums;
    }
```

Part (c):

```
public static boolean isDiverse(int[][] arr2D){
        int [] sums=rowSums(arr2D);
        for (int i=0; i < sums.length; i++){
            for (int j=i+1; j < sums.length; j++){
                if (sums[i]==sums[j]){
                    return false;
                }
            }
        }
        return true;
    }
```

1 Aa
1 of 3

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr) {
    int sum = 0;
    for (int i : arr) {
        sum += i;
    }
    return sum;
}
```

Part (b) begins on page 6.

1 AB

2 of 3

Assume that `arraySum` works as specified, regardless of what you wrote in part (a). You must use `arraySum` appropriately to receive full credit.

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 *  the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D) {
    int[] ans = new int[arr2D.length];

    for (int r=0; r < arr2D.length; r++) {
        ans[r] = arraySum(arr2D[r]);
    }

    return ans;
}
```

Part (c) begins on page 8.

1 AC

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

3 of 3

```
/** Returns true if all rows in arr2D have different row sums;
 *          false otherwise.
 */
public static boolean isDiverse(int[][] arr2D) {
    int[] sums = rowSums(arr2D);
    ~~boolean ans = false;~~
    for (int i : sums) {
        for (int j : sums) {
            if (i == j) {
                return false;
            }
        }
    }
    return true;
}
```


GO ON TO THE NEXT PAGE.

Complete method `arraySum` below.

1 Ba

1 of 3

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
{
 int sum=0;
 for(int i=0; i<arr.length; i++){
   sum+= arr[i];
   }
 return sum;
}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

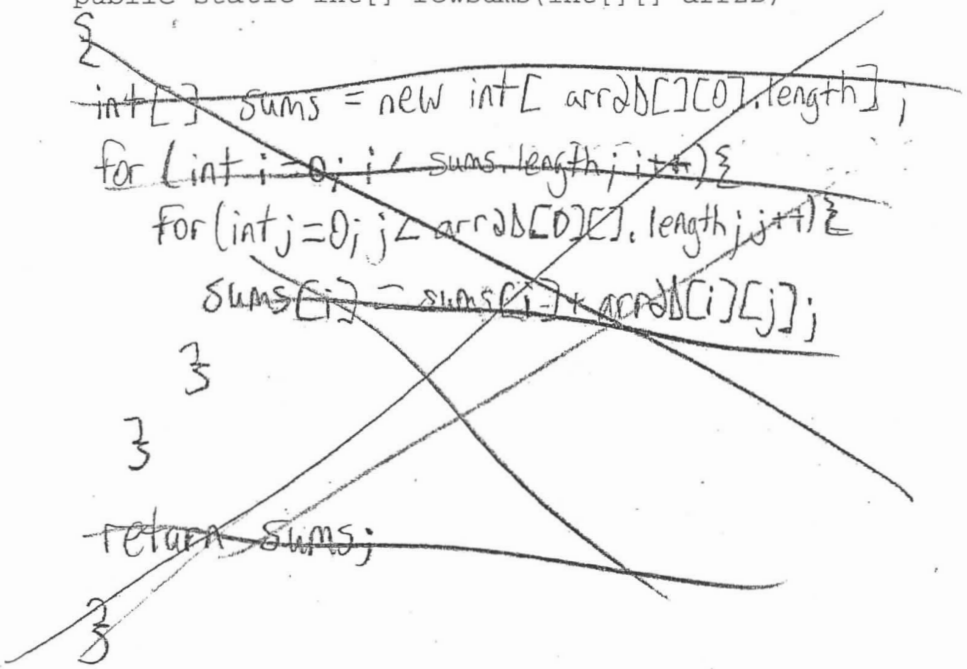
1 Bb

Assume that `arraySum` works as specified, regardless of what you wrote in part (a). You must use `arraySum` appropriately to receive full credit.

2 of 3

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 *  the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
```



```
{
int[] sums = new int[ arr2D[][0].length];
for (int i=0; i< sums.length; i++){
   for(int j=0; j< arr2D[0][].length; j++){
     sums[i] = sums[i-1]+ arr2D[i][j];
   }
}
return sums;
}
```

```
{
int[] sums= new int[arr2D[][0].length];
for(int i=0; i< sums.length; i++){
  sums[i]=      this.arraySum( arr2D[i]);
 }
return sums;
}
```

Part (c) begins on page 8.

1Bc

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

3 of 3

Complete method `isDiverse` below.

```
/** Returns true   if all rows in arr2D have different row sums;
 *          false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
{
  int count=0;
  int[] sums= this.rowSums(arr2D);
  for(int i=0; i<sums.length-1; i++){
    for(int j=i; j<sums.length-1; j++){
      if(sums[i]==sums[j+1])
        count++;
    }
  }
  if (count>0)
    return false;
  else
    return true;
}
```


GO ON TO THE NEXT PAGE.

1Ca

1of3

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
{ int arraySum= 0;
  for(int i = 0, i<= arr.size; i ++)
   {
     arraySum+= arr[i];
   }
   return  arraySum;
 }
```

Part (b) begins on page 6.

1Cb

2of3

Assume that `arraySum` works as specified, regardless of what you wrote in part (a). You must use `arraySum` appropriately to receive full credit.

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 *  the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
{
  int counter = 0;
  for( int r=0, r< = arr[][].size/2 ; r++)
  {
    for (int c=0, c< = arr[][].size/2; c++)
    {
      int k=0;
      If (k< = arr[].size/2)
      {
rraySum ( arr2D[k][c]);
      k++;
      }
    }
  }
}
```

Part (c) begins on page 8.

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

```
/** Returns true  if all rows in arr2D have different row sums;
 *         false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
{
  boolean isDiverse = true;
  If (rowSums(arr2D) == arraySums(arr2D)
   {
    isDiverse = false;
   }
  else
   {
     isDiverse = true;
   }
  return  isDiverse;
}
```

AP® COMPUTER SCIENCE A
2015 SCORING COMMENTARY

Question 1

Overview

This question focused on accessing and processing data from one and two-dimensional arrays. It involved abstraction through the use of methods. Part (a) asked students to sum the integers stored in a one-dimensional array. Part (b) asked students to sum the rows of a two-dimensional array by explicitly calling the method from part (a). Part (c) asked students to call the method from part (b) to obtain the sums of the rows of a two-dimensional array and analyze those sums. More specifically, it asked students to look for duplicates within the one-dimensional array of sums and to return the answer `false` if the one-dimensional array had any duplicate sums and `true` if it did not have any duplicate sums.

Sample: 1A
Score: 8

The solution accesses all elements of array parameter `arr` and correctly initializes, computes, and returns the sum of all elements of `arr`. The solution earned 2 points for part (a).

The solution constructs a correctly sized one-dimensional array of `ints`, accesses all rows of the `arr2D` parameter to compute the sum of each row using method `arraySum`, and assigns the computed sum to an element of the one-dimensional array. It returns a correctly computed array of row sums. The *kth* element of the returned array contains the computed sum of the corresponding row in the two-dimensional array for all rows. The solution earned 4 points for part (b).

The solution computes an array of row sums from `arr2D` using method `rowSums`. While doing the comparison, the student did not adjust the loops for the case where the row sum is being compared to itself. The value `true` is returned when all compared sums are different and `false` is returned otherwise. The solution earned 2 points for part (c).

Sample: 1B
Score: 6

The solution accesses all elements of array parameter `arr`, and correctly initializes, computes, and returns the sum of all elements of `arr`. The solution earned 2 points for part (a).

The solution constructs a one-dimensional array of `ints`, but the size is incorrect. All rows of the `arr2D` parameter are accessed with no bounds errors. The call to method `arraySum` is incorrect due to the use of "`this.`" in front of the method name. The *kth* element of the returned array contains the computed sum of the corresponding row in the two-dimensional array for all rows. The solution earned 2 points for part (b).

The call to method `rowSums` is incorrect due to the use of "`this.`" in front of the method name. The solution checks all and only pairs of row sums for equality and correctly returns `true` or `false` appropriately. The solution earned 2 points for part (c).

Sample: 1C
Score: 2

The solution does not access all elements of `arr` because of an incorrect bounds test. It does earn the point for initializing, computing, and returning the sum of elements. The solution earned 1 point for part (a).

Question 1 (continued)

The solution does not declare an array for the row sums; it fails to access all rows of `arr2D` because of incorrect bounds test in both loops; it does not correctly call method `arraySum` to compute row sums; there is no array to be returned. The solution earned no points for part (b).

The solution computes an array of row sums using method `rowSums`. The solution does not compare all necessary pairs of row sums. The value `true` or `false` is returned after only one comparison, an immediate return. The solution earned 1 point for part (c).