AP® Computer Science A
Free-Response Scoring Guidelines

Question 1: Word Scrambler

Part A:	`recombine`	**4 points**

+1 correctly finds middle index of each word

+1 correctly forms first half of `word1` and second half of `word2` using identified index (can be off by one)

+1 first half and second half of each word is correct length (second half of an odd length word is 1 character longer than first half)

+1 concatenates and returns "combined" string

Part B:	`mixedWords`	**5 points**

+1 creates and returns a result array of correct length

+2 attempts to loop over all pairs of strings in `words`

- **+1** attempt
- **+1** correct (loses for off-by-one pairing)

+1 correctly combines pairs of strings (must use `recombine`)

+1 correctly stores all new string pairs in result array

AP® Computer Science A
Free-Response Scoring Guidelines

Question 2: Mountain

Part A:	`getPeakIndex`	**5 points**

+2 loop over array

- **+1** Accesses a consecutive triplet of `array` values (*must be in context of loop*).
- **+1** Accesses all and only triplets of `array` values (*no boundary errors*).

+2 compare for peak

- **+1** Compares middle element of `array` triplet to both previous and subsequent `array` values.
- **+1** Determines if array element is a peak.

+1 Returns peak index or -1 if there is no peak.

Part B:	`isMountain`	**4 points**

+1 Calls `getPeakIndex` method.

+1 Calls `isIncreasing` and `isDecreasing` methods.

+1 Returns `false` when `getPeakIndex` returns -1
(must not call `isIncreasing` and `isDecreasing`).

+1 Returns `true` if `array` has the mountain property.

AP® Computer Science A
Free-Response Scoring Guidelines

Question 3: Compute Temperatures

Part A:	`computeTemp`	**4 points**

+2 Returns border element

- **+1** Checks border conditions
- **+1** Returns element when "on border"

+1 Computes sum of the four adjacent elements *(must not assign to an* `int`*)*

+1 Returns the average when not "on border"

Part B:	`updateAllTemps`	**5 points**

+1 Instantiates a new `double` array with same number of rows and columns as `temps`

+1 Accesses all and only values in `temps` or new array *(no bounds errors)*

+1 Calls `computeTemp` and stores results in the corresponding elements of the new array

+1 At exit: `temps` contains all of the updated temperatures

+1 Returns whether all temp changes are in tolerance

AP® Computer Science A
Free-Response Scoring Guidelines

Question 4: Score Statistics

Part A:	`record`	**7 points**

+1 Compares `score` value with value retrieved from object in list (must use `getScore`)

+1 Compares `score` with all appropriate entries in `scoreList`
(*no bounds error, early exit, or infinite loop*)

+1 Creates a new `ScoreInfo` object containing `score`

+1 Inserts object into list based on a comparison (other than equality) with object in list
(*point not awarded if inserted more than once*)

+1 Inserts new `ScoreInfo` object into `scoreList` once and only once in maintaining numerical order and numerical uniqueness (*no destruction of existing data*)

+1 All `ScoreInfo` objects in `scoreList` have correct frequencies after updates if any.

+1 correctly return `boolean` value based on whether a new `ScoreInfo` object was added

-2 Add to `scoreList` inside for-each loop of `scoreList`
(gets `ConcurrentModificationException`).

Part B:	`recordScores`	**2 points**

+1 correctly loop over all scores in `stuScores`

+1 call `record(someInt)` (*in context of loop*)

AP® Computer Science A
Free-Response Canonical Solutions

Question 1: Word Scrambler

PART A:

```
private String recombine(String word1, String word2)
{
  return word1.substring(0, word1.length() / 2) +
         word2.substring(word2.length() / 2);
}
```

PART B:

```
private String[] mixedWords(String[] words)
{
  String[] result = new String[words.length];

  for (int k = 0; k < result.length; k = k + 2)
  {
    result[k] = recombine(words[k], words[k + 1]);
    result[k + 1] = recombine (words[k + 1], words[k]);
  }

  return result;
}
```

AP® Computer Science A
Free-Response Canonical Solutions

Question 2: Mountain

PART A:

```
public static int getPeakIndex(int[] array)
{
  for (int k = 1; k < array.length - 1; k++)
  {
    if (array[k - 1] < array[k] && array[k] > array[k + 1])
      return k;
  }

  return -1;
}
```

PART B:

```
public static boolean isMountain(int[] array)
{
  int peak = getPeakIndex(array);
  return (peak != -1) && isIncreasing(array, peak) &&
                         isDecreasing(array, peak);
}
```

AP® Computer Science A
Free-Response Canonical Solutions

Question 3: Compute Temperatures

PART A:

```
private double computeTemp(int row, int col)
{
  if (row == 0 || row == temps.length - 1
      || col == 0 || col == temps[0].length - 1)
  {
    return temps[row][col];
  }

  double sum = temps[row - 1][col] + temps[row + 1][col]
               + temps[row][col - 1] + temps[row][col + 1];

  return sum / 4.0;
}
```

PART B:

```
public boolean updateAllTemps(double tolerance)
{
  double[][]newTemps = new double[temps.length][temps[0].length];

  boolean within = true;

  for (int r = 0; r < temps.length; r++)
  {
    for (int c = 0; c < temps[0].length; c++)
    {
      newTemps[r][c] = computeTemp(r,c);
      if (Math.abs(newTemps[r][c] - temps[r][c]) > tolerance)
      {
        within = false;
      }
    }
  }

  temps = newTemps;
  return within;
}
```

AP® Computer Science A
Free-Response Canonical Solutions

Question 4: Score Statistics

PART A:

```
public boolean record(int score)
{
  int k = 0;
  while (k < scoreList.size() && score > scoreList.get(k).getScore())
  {
    k++;
  }

  boolean found = k < scoreList.size() &&
                  score == scoreList.get(k).getScore();
  if (found)
    scoreList.get(k).increment();
  else
    scoreList.add(k, new ScoreInfo(score));

  return !found;
}
```

Alternate solution

```
public boolean record(int score)
{
  for (int k = 0; k < scoreList.size(); k++)
  {
    if (score < scoreList.get(k).getScore())
    {
      scoreList.add(k, new ScoreInfo(score));
      return true;
    }
    else if (score == scoreList.get(k).getScore())
    {
      scoreList.get(k).increment();
      return false;
    }
  }

  scoreList.add(new ScoreInfo(score));
  return true;
}
```

PART B:

```
public void recordScores(int[] stuScores)
{
  for (int score : stuScores)
    record(score);
}
```