CS101 – Computer Programming
Quiz for Friday Batch – 10 October 2014

Q1. We want to “sort” an array of positive integers in increasing order by looking only at the least significant two digits of each element (in decimal representation). Thus, if the unsorted array (of 4 elements) is A[0] = 109, A[1] = 307, A[2] = 215, A[3] = 500, then the sorted (in increasing order) array should be A[0] = 500, A[1] = 307, A[2] = 109, A[3] = 215.
A sorting function that accomplishes the above task is said to be "stable" if the relative order of every pair of elements in A whose last two digits (in decimal representation) are the same is preserved by the sorting function. Thus, if the unsorted array (of 4 elements) is A[0] = 201, A[1] = 101, A[2] = 300, A[4] = 400, and if the sorting function returns the sorted array A[0] = 400, A[1] = 300, A[2] = 101, A[4] = 201, then the function is not stable. This is because both 201 and 300 appeared before 101 and 400, respectively, in the unsorted array, and their last two digits were the same. Yet, the sorting function reversed their relative order (101 before 201 and 400 before 300) in the sorted array. A stable sorting program would return the following sorted array for the same input: A[0] = 300, A[1] = 400, A[2] = 201, A[1] = 101.
A student has written the following code fragment to sort an array A of integers as described above. Assume that when the function "mergeSort" is called from "main", its parameters are as follows: "A" is an array of "n" positive integers (where "n" is a positive integer <= 100), the value of "start" is 0, and the value of "end" is "n". The elements in the array A are indexed as A[0] through A[n-1].

```
void mergeSort(int A[], int start, int end)
{
  int mid;

  if (end == start + 1) { return; }
  else {
    mid = (start + end)/2;
    mergeSort(A, start, mid);
    mergeSort(A, mid, end);
    mergeSortedSubarrays(A, start, mid, end);
    return;
  }
}

void mergeSortedSubarrays(int A[], int start, int mid, int end)
{
  int i, j, index = start;
  int B[100];

  for (i = start, j = mid; ((i < mid) || (j < end)); ) {
      if ((i < mid) && (j < end)) {
        if ((A[j] % 100) <= (A[i] % 100)) { B[index++] = A[j++]; }
        else {B[index++] = A[i++];}
      }
      else if (i < mid) {B[index++] = A[i++]; }
      else {B[index++] = A[j++];}
  }

  for (i = start; i < end; i++) { A[i] = B[i]; }
  return;
}
```

Which of the following are true of the above function "mergeSort"?
A. It is stable
B. It is not stable
C. It sorts the array A in increasing order considering the last two digits of elements
D. It sorts the array A in decreasing order considering the last two digits of elements

Q2. Consider the following recursive binary search function discussed in the lecture slides.

```
 // PRECONDITION: A[start] … A[end – 1] sorted in increasing order
int binarySearch(int A[], int start, int end, int srchElement) {
  if (end == start + 1) { // Array A has exactly 1 element
    if (A[start] == srchElement) {
      return start;
    }
    else { return -1; }
  }
  int mid = (start + end)/2;
  if (A[mid] == srchElement) {
    return mid;
  }
  else {
    if (A[mid] < srchElement) {
      return binarySearch(A, mid, end, srchElement);
    }
    else {
      return binarySearch(A, start, mid, srchElement);
    }
  }
}
// POSTCONDITION: If srchElement in A[start] … A[end-1], return its index, else -1
```

The function 'binarySearch' depicted above expects the array 'A' to be sorted in increasing order, and guarantees that if 'srchElement' is present in 'A', the index of 'srchElement' in 'A' is returned; otherwise -1 is returned. In this question, we want to investigate how the same 'binarySearch' function would behave if the array 'A' is not sorted in increasing order(but in any arbitrary order).

Suppose that 'binarySearch' (as depicted above) is called from 'main' 10 times. Suppose further that in each of these 10 calls, the same parameters 'A', 'start' and 'end' are passed from 'main'. Specifically, A = {15, 22, 11, 13, 12, 18, 16, 6, 14, 5} (i.e. A[0] = 15 and A[9] = 5), start = 0, and end = 10. Only the parameter 'srchElement' is varied in the different calls of 'binarySearch' from 'main'. Specifically, each call of 'binarySearch' uses a distinct value of 'srchElement' from the set {15, 22, 11, 13, 12, 18, 16, 6, 14, 5}. Note that all values of 'srchElement' used are such that 'srchElement' is present in the array 'A'.

For which values of 'srchElement' will 'binarySearch' successfully find the index of 'srchElement' in 'A'?

A. 15,22,13,16,14
B. 11,13,18
C. 13,12,16, 6
D. 12,16,14, 5