CS101 – Computer Programming
Quiz for Wednesday Batch – 8 October 2014

Q1. Consider the following variation of Merge Sort to sort the array in descending order. In the version of Merge Sort discussed in class, the array is divided into two sub arrays of (almost) equal size. In the following, the array is divided into **three** sub arrays of almost equal size. Assume the maximum size of array A is 100.

The function '**mergeSortedThreeSubArrays(A, start, mid1, mid2, end)**' takes in input array A, where the three sub-arrays from A[start] to A[mid1-1], A[mid1] to A[mid2-1], **and** A[mid2] to A[end-1] are each sorted in descending order. When this function returns, it merges the elements of all three sub-arrays in descending order and stores them in A[start] to A[end-1].

The function '**mergeSortedSubArrays(A, start, mid, end)**' takes in input array A, where the two sub-arrays from A[start] to A[mid-1] **and** A[mid] to A[end-1] are each sorted in descending order. When this function returns, it merges the elements of both sub-arrays in descending order and stores them in A[start] to A[end-1].

The function **selectionSort(A, start, end)** sorts the elements of array A from index 'start' to index 'end-1' in descending order.

Given below is the function '**mergeSort**' for sorting array 'A' using Merge Sort algorithm. It calls the function '**mergeSortedThreeSubArrays**'**:**

```
void mergeSort(int A[], int start, int end) {
      if(end == start)
            return;
      if(end – start < 3) {
            selectionSort(A, start, end);
            return;
      }
      int mid1 = start + (end – start)/3;
      int mid2 = start + 2*(end - start)/3;
      mergeSort(A, start, mid1);
      mergeSort(A, mid1, mid2);
      mergeSort(A, mid2, end);

      mergeSortedThreeSubArrays(A, start, mid1, mid2, end);
      return;
}
```

Out of the following options, choose the **correct** function definition(s) for the function '**mergeSortedThreeSubArrays**' as specified in the description for this question:

A.
```
void mergeSortedThreeSubArrays(int A[],int start,int mid1,int mid2,int end){
      mergeSortedSubArrays(A, start, mid1, end);
      mergeSortedSubArrays(A, mid1, mid2, end);
}
```

B.
```
void mergeSortedThreeSubArrays(int A[],int start,int mid1,int mid2,int end){
      int i, j, k;
      int tempA[100]; int index = start;
      for(i =  start, j = mid1, k = mid2;
                  (i < mid1 || j < mid2 || k < end); index++) {
            if(i < mid1 && j < mid2 && k < end) {
                  if(A[i] > A[j] && A[i] > A[k]) {
                        tempA[index] = A[i];
                        i++;
                  } else if(A[j] > A[i] && A[j] > A[k]) {
                        tempA[index] = A[j];
```

```
                        j++;
                    } else {
                        tempA[index] = A[k];
                        k++;
                    }
                } else if(i < mid1 && j < mid2) {
                    if(A[i] > A[j]) {
                        tempA[index] = A[i];
                        i++;
                    } else {
                        tempA[index] = A[j];
                        j++;
                    }
                } else if(i < mid1 && k < end) {
                    if(A[i] > A[k]) {
                        tempA[index] = A[i];
                        i++;
                    } else {
                        tempA[index] = A[k];
                        k++;
                    }
                } else if(j < mid2 && k < end) {
                    if(A[j] > A[k]) {
                        tempA[index] = A[j];
                        j++;
                    } else {
                        tempA[index] = A[k];
                        k++;
                    }
                } else if(i < mid1) {
                    tempA[index] = A[i];
                    i++;
                } else if(j < mid2) {
                    tempA[index] = A[j];
                    j++;
                } else {
                    tempA[index] = A[k];
                    k++;
                }
            }

            for(int x = start; x < end; x++) {
                A[x] = tempA[x];
            }
    }
```

C.

```
void mergeSortedThreeSubArrays(int A[],int start,int mid1,int mid2,int end){
        int i, j, index = start; int tempA[100];
        for(i = start, j = mid1; (i < mid1 || j < mid2); index++) {
                if(i < mid1 && j < mid2) {
                    if(A[i] > A[j]) {
                        tempA[index] = A[i];
                        i++;
                    } else {
                        tempA[index] = A[j];
                        j++;
                    }
                } else if (i < mid1) {
                    tempA[index] = A[i];
                    i++;
                } else {
                    tempA[index] = A[j];
```

```
                        j++;
                }
        }
        for(; (i < mid2 || j < end); index++) {
                if(i < mid2 && j < end) {
                        if(A[i] > A[j]) {
                                tempA[index] = A[i];
                                i++;
                        } else {
                                tempA[index] = A[j];
                                j++;
                        }
                } else if (i < mid2) {
                        tempA[index] = A[i];
                        i++;
                } else {
                        tempA[index] = A[j];
                        j++;
                }
        }

        for(int x = start; x < end; x++) {
                A[x] = tempA[x];
        }
    }
```

D.
```
    void mergeSortedThreeSubArrays(int A[],int start,int mid1,int mid2,int end) {
            mergeSortedSubArrays(A, start, mid1, mid2);
            mergeSortedSubArrays(A, start, mid2, end);
    }
```

E. None of these

Q2. Let us consider an ordering on pairs. An ordered pair (x1,y1) > (x2, y2)
if x1 > x2 or if ((x1 = x2) and (y1 > y2))
Similarly, equality of two pairs (x1, y1) and (x2, y2) is defined as (x1 = x2) and (y1 = y2).
We want to search for a pair from a given array of pairs, which is stored in a 2D array A[100][2]. A[i][0] denotes the x-value and A[i][1] denotes the y-value of the i-th pair. Assume that the given array A is sorted in ascending order according to the above description.
Given below is the function '**binarySearch**' for searching pairs (x1,y1) using Binary Search:

```
int binarySearch(int A[][2], int start, int end, int x_ele, int y_ele) {
       int mid = (start + end)/2;
       if(start == end-1) {
              if(!compareElements(A, mid, x_ele, y_ele)) {
                     return mid;
              } else {
                     return -1;
              }
       }
       if(!compareElements(A, mid, x_ele, y_ele)) {
              return mid;
       } else if (compareElements(A, mid, x_ele, y_ele) < 0) {
              return binarySearch(A, start, mid, x_ele, y_ele);
       } else {
              return binarySearch(A, mid, end, x_ele, y_ele);
       }
}
```

The search is said to be **successful** if the pair to be searched exists in the array and returns the index of the searched pair. It returns -1 if it doesn't exist in the array.

The function '**compareElements**' is called by the function '**binarySearch**'. Choose the correct option(s) from the following.

A. If the array A is sorted in descending order, then using the following definition of the function '**compareElements**', the search will be successful.

```
int compareElements(int A[][2],  int index, int x, int y) {
       int x_temp = A[i][0], y_temp = A[i][1];
       if(x_temp == x) {
              if(y_temp == y) {
                     return 0;
              } else if(y_temp > y) {
                     return 1;
              } else {
                     return -1;
              }
       } else if(x_temp > x) {
              return 1;
       } else {
              return -1;
       }
}
```

B. If the array A is sorted in ascending order, then using the following definition of the function '**compareElements**', the search will be successful.

```
int compareElements(int A[][2],  int index, int x, int y) {
 int x_temp = A[i][0], y_temp = A[i][1];
 if(x_temp == x) {
 if(y_temp == y) {
 return 0;
 } else if(y_temp > y) {
 return -1;
 } else {
 return 1;
 }
 } else if(x_temp > x) {
 return -1;
 } else {
 return 1;
 }
}
```

C. If the array A is sorted in ascending order then using the following definition of the function '**compareElements**', the search will be successful.

```
int compareElements(int A[][2],  int index, int x, int y) {
       int x_temp = A[i][0], y_temp = A[i][1];
```

```
        return (x_temp - x) + (y_temp - y);
}
```

D. If the array A is sorted in descending order then using the following definition of the function '**compareElements**', the search will be successful.

```
int compareElements(int A[][2],  int index, int x, int y) {
 int x_temp = A[i][0], y_temp = A[i][1];
 return ((x_temp - x) ? (x_temp - x): (y_temp - y));
}
```

E. None of these