CS101 – Computer Programming
Quiz for Thursday Batch – 9 October 2014

Q1. Consider the recursive and iterative implementations of Merge Sort algorithm to sort the given array 'A' of size 'n' in descending order. The recursive function named as '**mergeSort**' and the iterative function named as '**mergeIterativeSort**', are both shown below separately. Both functions use a common function for merging i.e. '**mergeSortedSubArrays**'.
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 already 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].

```
void mergeSort(int A[], int start, int end) {
      if(end == start)
            return;
      int mid1 = (start + end)/2;
      mergeSort(A, start, mid);
      mergeSort(A, mid, end);
      mergeSortedSubArrays(A, start, mid, end);
      return;
}

void mergeIterativeSort(int A[], int n) {
      int subArrayLength = 1;
      int mid, end;
      while(subArrayLength <= n/2) {
            for(int start = 0; start < n; start += 2*subArrayLength) {
                  mid = start + subArrayLength;
                  end = start + 2*subArrayLength;
                  mergeSortedSubArrays(A, start, mid, end);
            }
            subArrayLength *= 2;
      }
}
```

Select the correct choice(s) from the following:

- **A.** For any array of size 2^k, where k is a positive integer, the **number of function calls** made to '**mergeSortedSubArrays',** by calling '**mergeSort**(A,0,n);' and '**mergeIterativeSort**(A,n);' from the '**main**' program, are both equal.
- **B.** After the final iteration of the while loop in the function '**mergeIterativeSort**' , A[i] >= A[i+1], for all even integers < n
- **C.** The last function call to '**mergeSortedSubArrays**', from both the recursive and iterative version of the above Merge Sort is '**mergeSortedSubArrays(A, 0, n/2, n)**'
- **D.** None of these

Q2. Consider an array of points in the XY plane represented as (x,y) ordered pairs.

- (x1,y1) > (x2, y2) if the euclidean distance of (x1, y1) from the origin (0,0) is more than that of (x2, y2) i.e. $\text{sqrt}(x1^2 + y1^2) > \text{sqrt}(x2^2 + y2^2)$.
- If the points (x1, y1) and (x2, y2) lie on the same circle, then they are ordered based on their x-coordinates. i.e. (x1, y1) > (x2, y2) if x1 > x2 and $\text{sqrt}(x1^2 + y1^2) = \text{sqrt}(x2^2 + y2^2)$.
- Similarly, two points are equal only if the corresponding x and y coordinates are equal.

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.
Given below is the function '**binarySearch**' for searching pairs (x1,y1) using Binary Search method:

```
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 i, int x, int y){
	int x_pow, y_pow;
	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 && y_temp < y) {
		return -1;
	} else if(y_temp < y) {
		x_pow = x_temp * x_temp; y_pow = y_temp * y_temp;
		return (x_pow + y_pow - (x*x) - (y*y));
	}
}
```

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 i, int x, int y) {
   int x_temp = A[i][0], y_temp = A[i][1];
   return ((x*x) + (y*y) - (x_temp*x_temp) – (y_temp * y_temp));
}
```

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 i, int x, int y) {
   int x_temp = A[i][0], y_temp = A[i][1];
   int val = (x * x) + (y * y) – (x_temp * x_temp) – (y_temp * y_temp);
   return val ? val : x - x_temp ;
}
```

D. If the array A is sorted in descending order, then using the following definition of the function '**compareElements**', the search will be successful. **Note**: the function 'sqrt' returns the square-root of a number.

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

E. None of these