

**ICS 410 Programming Languages**

*Assignment 01*

Due Date: Mar 28

**Topics: Array, Pointers, and Functions**

**Problem: 01(30 points)**

Write a C program that will combine elements of two same size arrays and display those elements in a “Zigcending” order. The term “zigcending” here would mean remaining-highest-lowest order. For example, zigcending of 3, 5, 9, 2, 8 & 1 is: 9, 1, 8, 2, 5, 3.

First, the program should ask the user to enter array size, then prompt the user for inserting element values for both arrays. Finally the program should display all the elments in a ‘zigcending’ order.

Test Data :
Enter array size : 7
Input 7 elements in the 1st array :
element - 0 : 11
element - 1 : 5
…. …. …

element - 6 : 17

Input 7 elements in the 2nd array :
element - 0 : 9
element - 1 : 2

… … …
element - 6 : 11
Expected Output :
The combined array in zigcending order is :
17, 2, 11, 5, …..

[Hints: You may use any sorting algorithm]

**Problem 02 (30 points)**

Passing parameters into functions can be achieved through call-by-reference. In this method, the address of a parameter is passed to a certain function. To pass an argument by reference, a parameter pointer is supplied to the function. Write a C program that has a function called RoundedSwap() that takes in addresses of three interger type arguments as call-by-reference format, and swaps the three arguments value in a rounded way, i.e. value of second argument is replaced with the first one, the third value is replaced by the second, and the first value is replaced by the first.

**Problem 03 (40 points)**

We assume you have studied Insertion Sort in ICS353. In this problem you need to implement again the Insertion Sort algorithm by removing the array and using pointers and pointer arithmetic in functions.

**Part- A:**

The function mov\_element() takes the index as input of an array element that has been considered to be not in order. Therefore, the function moves the element towards at the front of the array, by repeatedly swapping the preceding element until the element, which is not in order, should be in the right position. The implementation of mov\_element() using array indexing is given below as reference for you.

void mov\_element(unsigned int i){

int i\_val;

for ( i\_val= arr[i]; i && arr[i−1]>i\_val ; i −−)

arr[i]= arr[i−1];

arr[i] = i\_val;

}

Now, you need to re-write this function by using pointers and pointer arithmetic, instead of using array indexing.

void mov\_element(int ∗ptrElement) {

/∗ your codes here ∗/

}

**Part-B**

The function i\_sort() has the main loop of the algorithm. It loops through the elements of the array, from the starting, until it reaches an element that is not in order. It calls the function mov\_element() to move the marked element to its proper position earlier in the array and resumes looping through until the end is reached. The code is given below:

void i\_sort(void) {

unsigned int i , length = arraylength(arr);

for (i=1 ;i<length ; i++)

if(arr[i]< arr[i−1])

mov\_element(i);

}

Re-write this function using pointers and pointer arithmetic by discarding the array. Use the mov\_element() function that you coded in part (A).

**Submission Guidelines**

You need to submit the following files:

* 1. yourID\_problem1.c
	2. yourID\_problem2.c
	3. yourID\_problem3\_Arr.c (using array)
	4. yourID\_problem3\_Ptr.c (using pointers)
	5. yourID\_simulation.pdf (showing your work simulation snapshots for each problem)

(All the best :)