Problem Solving by Computer – Logical Thinking

Tech Capsule 5 – Sorting and Searching based problems

Sorting of data provide us with a means of organizing information to facilitate easy retrieval of specific data. **Searching** methods are designed to take advantage of the organization of information and reduce the amount of effort to either locate a particular item or to establish that it is not present in the set of data.

Sorting algorithms arrange data in an order according to a predefined ordering relation. The 2 most common types of information are String Information and Numeric Information. The ordering relation for numeric data mostly involves arranging items in sequence from smallest to largest or vice versa (ascending or descending orders). Sorting string information is generally in standard lexicographical (alphabetical + dictionary) order.

Examples of Sorted arrays:

1) Ascending order in the case of Numeric information

   15, 18, 42, 51

2) Descending order in the case of Numeric information

   71, 64, 39, 11, 5

3) Lexicographical order in the case of String information

   b, back, bad, box, cat, zen

Problem 1- [Merging 2 sorted arrays]

Design an algorithm to accept 2 arrays of elements (which are given below. The elements of these 2 arrays are already sorted in ascending order) and merge them into one. All elements of the merged array must be in ascending order.

Input Arrays

array 1 -> 15, 18, 42, 51 (4 elements)
array 2 -> 8, 11, 16, 17, 44, 58, 71, 74 (8 elements)

Solution-

The output array must be –

8, 11, 15, 16, 17, 18, 42, 44, 51, 58, 71, 74 (4+8 = 12 elements)

Here in this problem we already know that both the input arrays are already sorted in ascending orders, which makes the algorithm to merge them simple and easy to write.

Step 1 –

/* declare and initialize input arrays and declare output array */

int array1[4]={15,18,42,51}, array2[8]={ 8, 11, 16, 17, 44, 58, 71, 74}, array3[12]

int i = 0, j = 0, k = 0 / indexes for arrays */*

Step 2 –

The merged array must be in ascending order, so we must compare elements of both arrays before we construct the elements of the output array.
Let us compare each element of the smaller array (array1) with each element of array2.

0^{th} Element -

if (array1[0] < array2[0])
array3[0] = array1[0]

- ✓ If the 0^{th} element of array1 is less than the 0^{th} element of array2, we must copy the element array1[0] to array3[0]
- ✓ Now we have the 0^{th} element of the output array
- ✓ In the next iteration we must compare 1^{st} element of array1 with the 0^{th} element of array2

else if (array1[0] > array2[0])
array3[0] = array2[0]

- ✓ If the 0^{th} element of array1 is greater than the 0^{th} element of array2, we must copy the element array2[0] to array3[0]
- ✓ Now we have the 0^{th} element of the output array
- ✓ In the next iteration we must compare 0^{th} element of array1 with the 1^{st} element of array2

else if (array1[0] == array2[0])
array3[0] = array1[0]
array3[1] = array2[0]

- ✓ If the 0^{th} element of array1 is equal to the 0^{th} element of array2, we must copy the element array1[0] to array3[0] and then copy array2[0] to array3[1]
- ✓ Now we have the 0^{th} and he 1^{st} element of the output array
- ✓ In the next iteration we must compare 1^{st} element of array1 with the 1^{st} element of array2

Note –

1) If all elements of array1 are copied to array3 then copy remaining elements of array2 to array3.
2) If all elements of array2 are copied to array3 then copy remaining elements of array1 to array3

The above comparison which we did for 2 elements of array1 and array2 must be done for all elements until we reach the end of any of the arrays.

Now let us write the algorithm with using a iterative loop and indexing to access array elements.

Assumption -

m = Number of elements in array1
n = Number of elements in array2
i = Index for array1 and initialized to 0 in the start.
j = Index for array 2 and initialized to 0 in the start.

while (i <= m) and (j<=n) do

- *Compare array1[i] and array2[j] and then copy the smaller member to array3*
- *Increment appropriate pointers*

done

If i < m then

- *Copy rest of array1 to array3*

else

- *Copy rest of array2 to array3*

Problem 2- [Sorting a given array]

Design an algorithm to accept 5 integers for an array and sort them into ascending order.

Solution-

Let us consider the below set of 5 integers to be sorted –
array -> 25, 10, 35, 5, 2

Step 1 –

if (array[0] > array[1])
We must swap the element

Now here in our case 25 is greater than 10 so we swap the elements to get the below array
array = 10, 25, 35, 5, 2

Step 2 –

if (array[1] > array[2])
We must swap the element

25 is not greater than 35 so we do nothing
The array remains unchanged as below –
array = 10, 25, 35, 5, 2

Step 3 –

if (array[2] > array[3])
We must swap the element

35 is greater than 5 so we swap. The array now becomes –
array = 10, 25, 5, 35, 2

Step 4 –
if (array[3] > array[4])
We must swap the element

35 is greater than 8 so we swap. The array now is –

array = 10, 25, 5, 2, 35

Here after Step 4, we have compared each element of the array with its next element, and swapped wherever required. This array is still not the final list which we require and so we need to continue comparing the elements from the start again. We can observe from this list that the greatest element of the list has reached its correct position so the number of comparisons to be done now must be 1 less than earlier.

Now if we repeat Step 1 to Step 3 (Step 4 is not required), the array will be like below –
array = 10, 5, 2, 25, 35

Here after step3, we have the 2nd greatest element also in the correct place.

Now repeat Step 1 and Step 2 (Step 3 and Step 4 not required in this case as the elements have been placed in their correct positions), the array will look like below –
array =5, 2, 10, 25, 35
Here we have 3 elements of the array in its correct position.

Now for the last time let us repeat Step 1, the array will look like below -
array =2, 5, 10, 25, 35

Here all the elements are sorted in ascending order and is the final array required.

Now let us write the above steps using iterative loops and proper indexing mechanism to implement our algorithm.

```
int array[5]={10, 25, 5, 2, 35}, i = 0, j = 0

while i is less than 4 ; do

        j = 0
        while j is less than (4-i); do

                if array[j] > array[j+1]
                        swap elements

                j = j+1
        done

i = i + 1
done
```

Try by yourself

Activity 1:
Design an algorithm to accept 10 integer elements for an array and rearrange the elements in descending order. The new array must be printed to the user.

Activity 2:
Design an algorithm to remove all duplicate elements from an ordered array (may be ascending or descending order) and contract the array accordingly.
For Example –

Input Array is -
2 4 2 5 3 4 6 5 1 7

After ordering –
1 2 2 3 4 4 5 5 6 7
Output array must be –
1 2 3 4 5 6 7

Activity 3:
Design an algorithm to accept 25 integer elements for an array. Then find the number with the maximum value in the set and the position

- where it occurs first
- where it occurs last

Activity 4:

Design an algorithm to find the position of a number X (entered by user and if it occurs) in an array of 15 elements. Array elements are also entered by the user.

Activity 5:

Design an algorithm to accept elements for 2 random arrays. Rearrange the array elements in descending order then merge them into form one array.