SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IoT and CS Including BCT

COURSE NAME :23ITB201-DATA STRUCTURES & ALGORITHMS

II YEAR / III SEMESTER

Unit III- **SORTING, SEARCHING & HASHING**

Topic :Insertion Sort – Divide & Conquer

Insertion sort *is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.*
It is a ***stable sorting*** *algorithm, meaning that elements with equal values maintain their relative order in the sorted output.*

OR

Insertion sort is a simple sorting algorithm that works by building a sorted array one element at a time.
It is considered an ” **in-place** ” sorting algorithm, meaning it doesn’t require any additional memory space beyond the original array.

Algorithm

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a **key.**

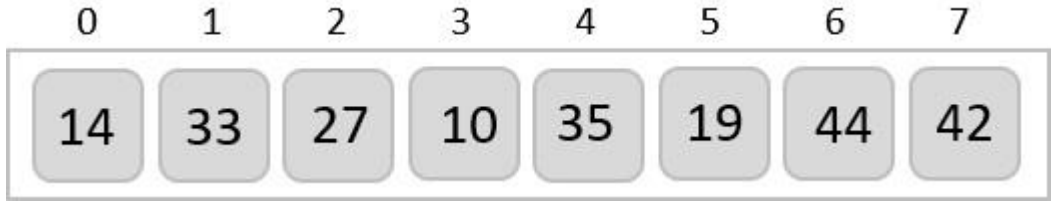
Step3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

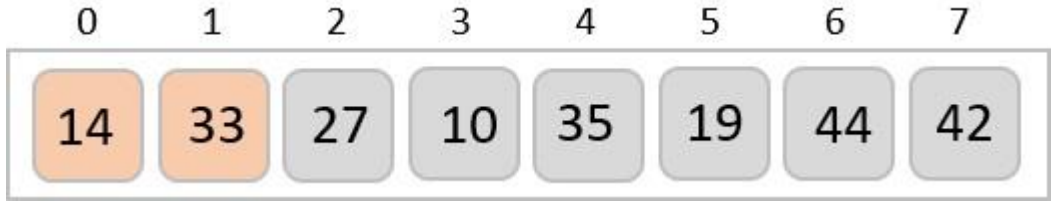
Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

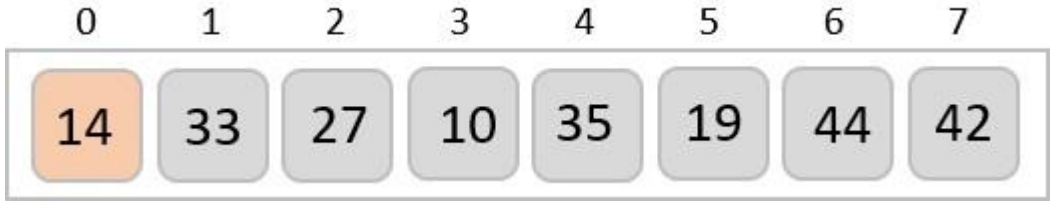
We take an unsorted array for our example.



Insertion sort compares the first two elements.



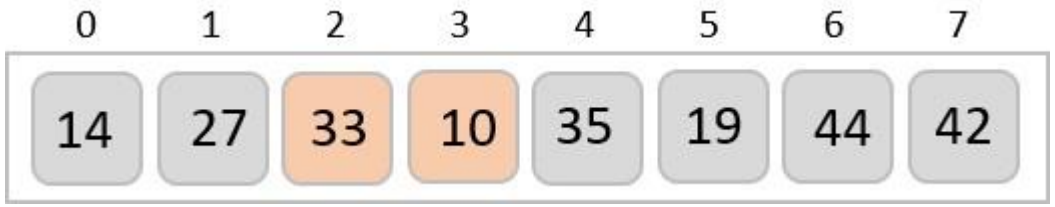
It finds that both 14 and 33 are already in ascending order.
For now, 14 is in sorted sub-list.



Insertion sort moves ahead and compares 33 with 27.

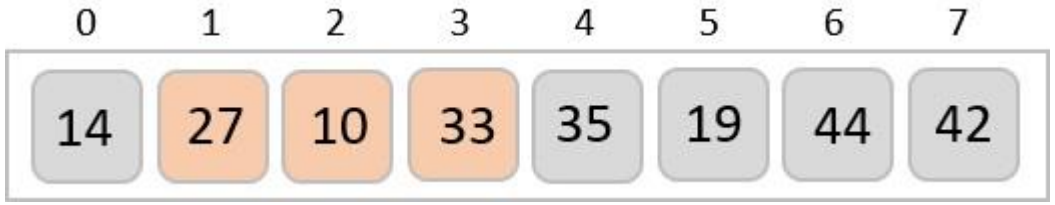
And finds that 33 is not in the correct position.
It swaps 33 with 27.
It also checks with all the elements of sorted sub-list.
Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14.
Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list.
Next, it compares 33 with 10.
These values are not in a sorted order.

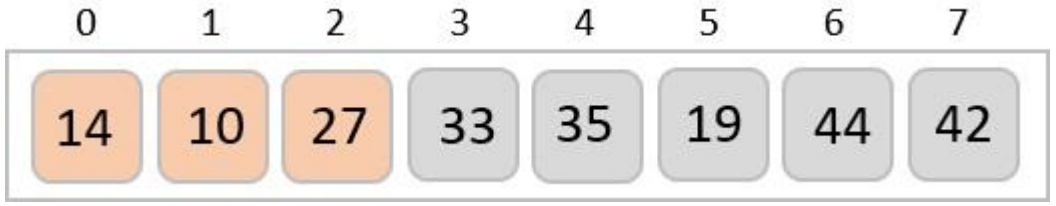


So they are swapped.

However, swapping makes 27 and 10 unsorted.

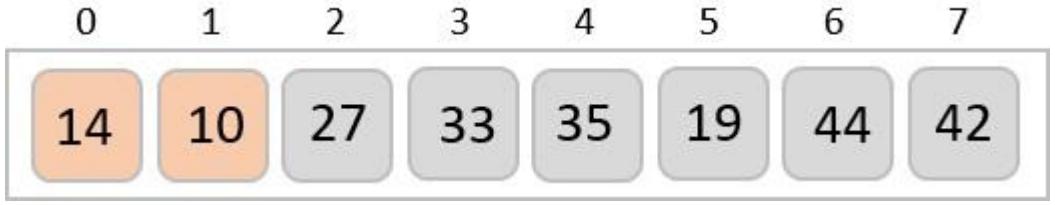


Hence, we swap them too.

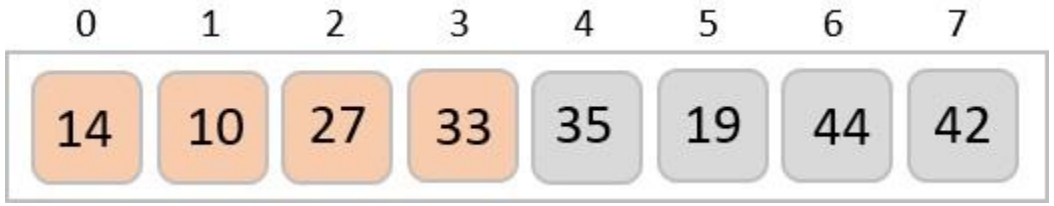


Again we find 14 and 10 in an unsorted order.

We swap them again.



By the end of third iteration, we have a sorted sub-list of 4 items.



This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

```
void insertionSort(int array[], int size){
  int key, j;
  for(int i = 1; i<size; i++) {
    key = array[i];//take value
    j = i;
    while(j > 0 && array[j-1]>key) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = key; //insert in right place
  }
}
```

```
int main(){
  int n;
  n = 5;
  int arr[5] = {67, 44, 82, 17, 20}; // initialize the array
  printf("Array before Sorting: ");
  for(int i = 0; i<n; i++)
    printf("%d ",arr[i]);
  printf("\n");
  insertionSort(arr, n);
  printf("Array after Sorting: ");
  for(int i = 0; i<n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
```

Divide and Conquer Introduction

Divide and Conquer is an algorithmic pattern.

In algorithmic methods, the design is to take a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces, and then merge the piecewise solutions into a global solution.

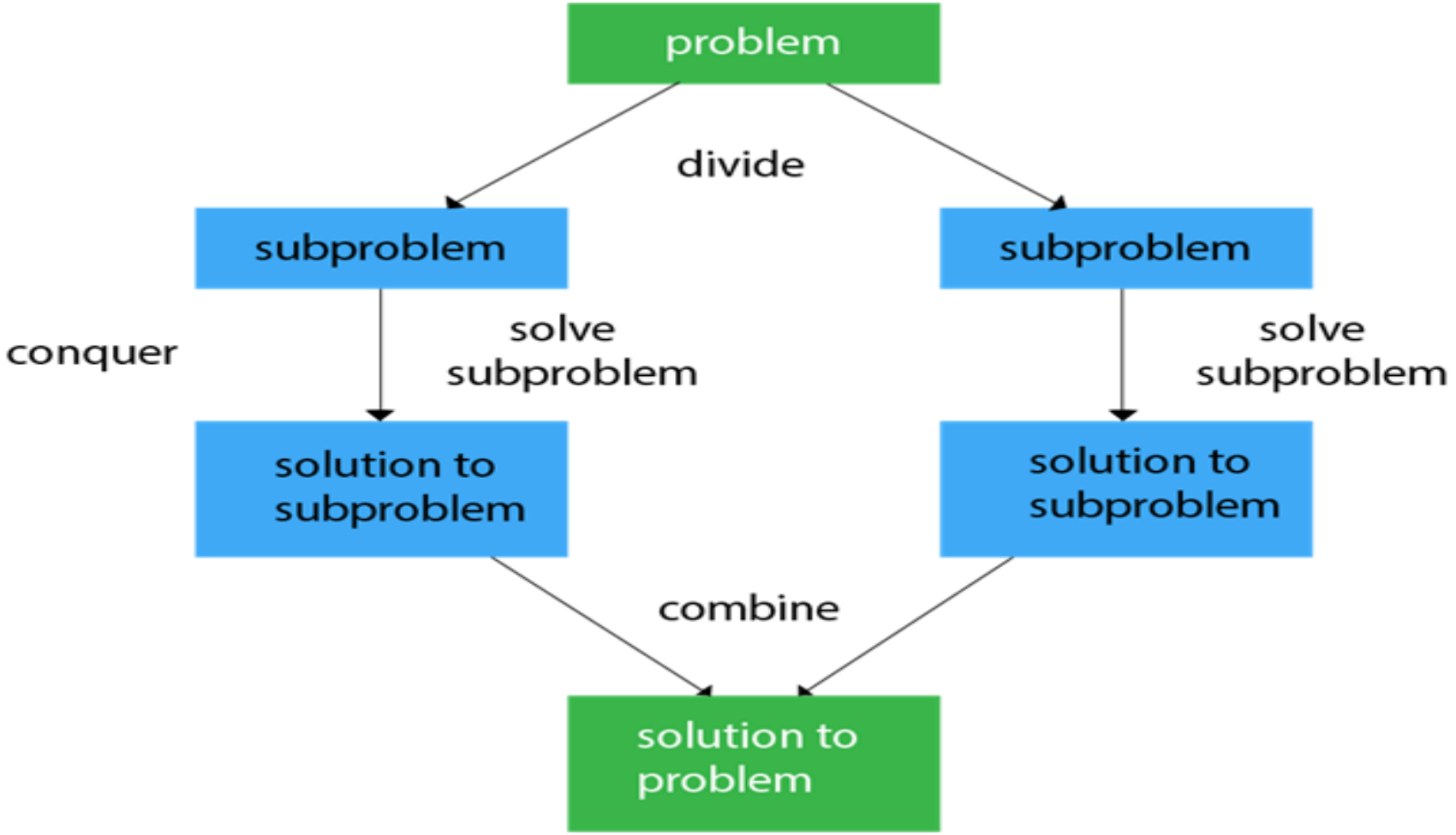
This mechanism of solving the problem is called the Divide & Conquer Strategy.

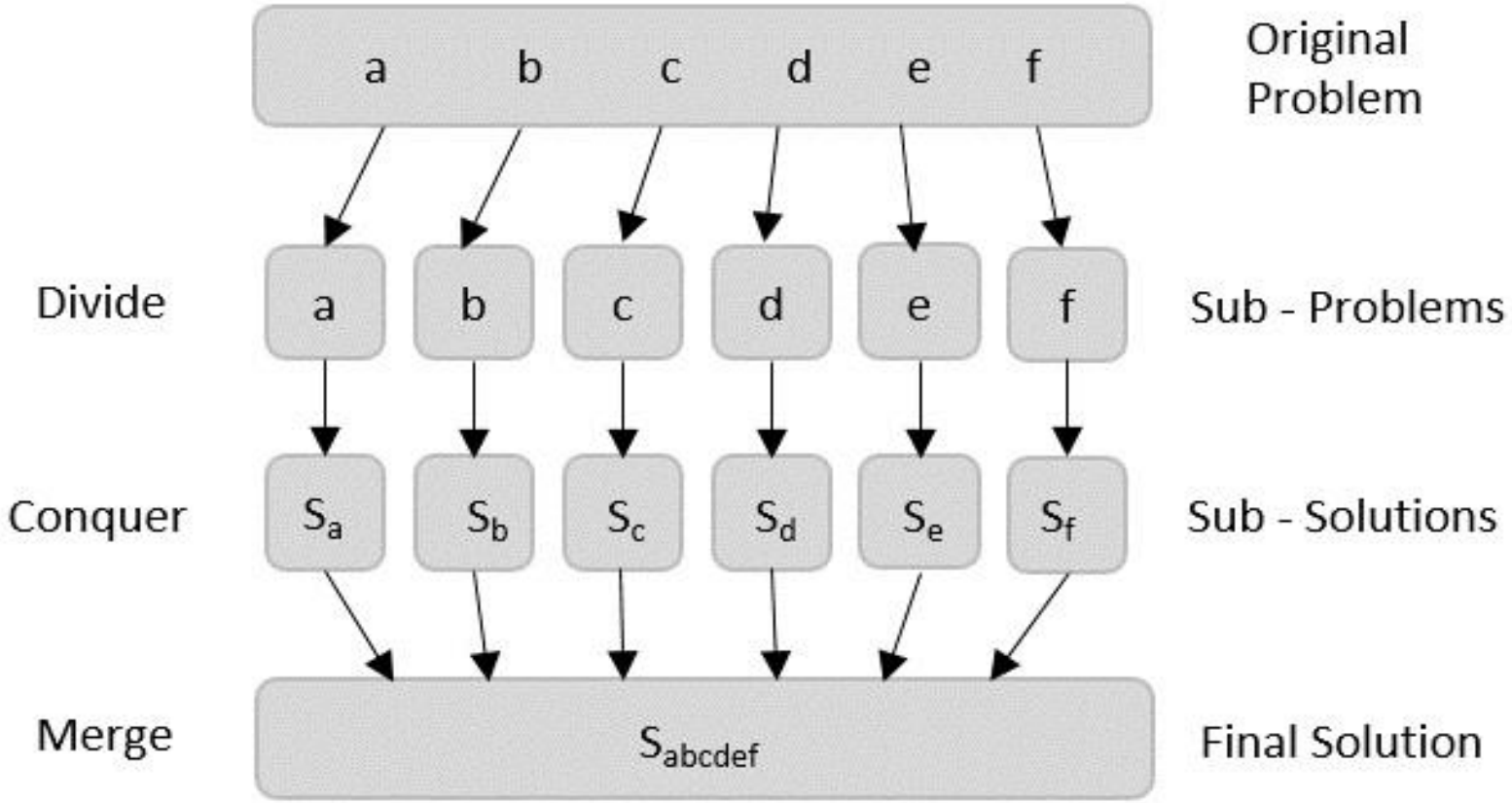
Divide and Conquer algorithm consists of a dispute using the following three steps.

Divide the original problem into a set of subproblems.

Conquer: Solve every subproblem individually, recursively.

Combine: Put together the solutions of the subproblems to get the solution to the whole problem.


problem
divide
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
combine
solution to problem


a b c d e f
Original Problem
Divide
a b c d e f
Sub - Problems
Conquer
S_a S_b S_c S_d S_e S_f
Sub - Solutions
Merge
S_{abcdef}
Final Solution

Divide/Break

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem.

This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible.

At this stage, sub-problems become atomic in size but still represent some part of the actual problem.

Conquer/Solve

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

Merge/Combine

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem.

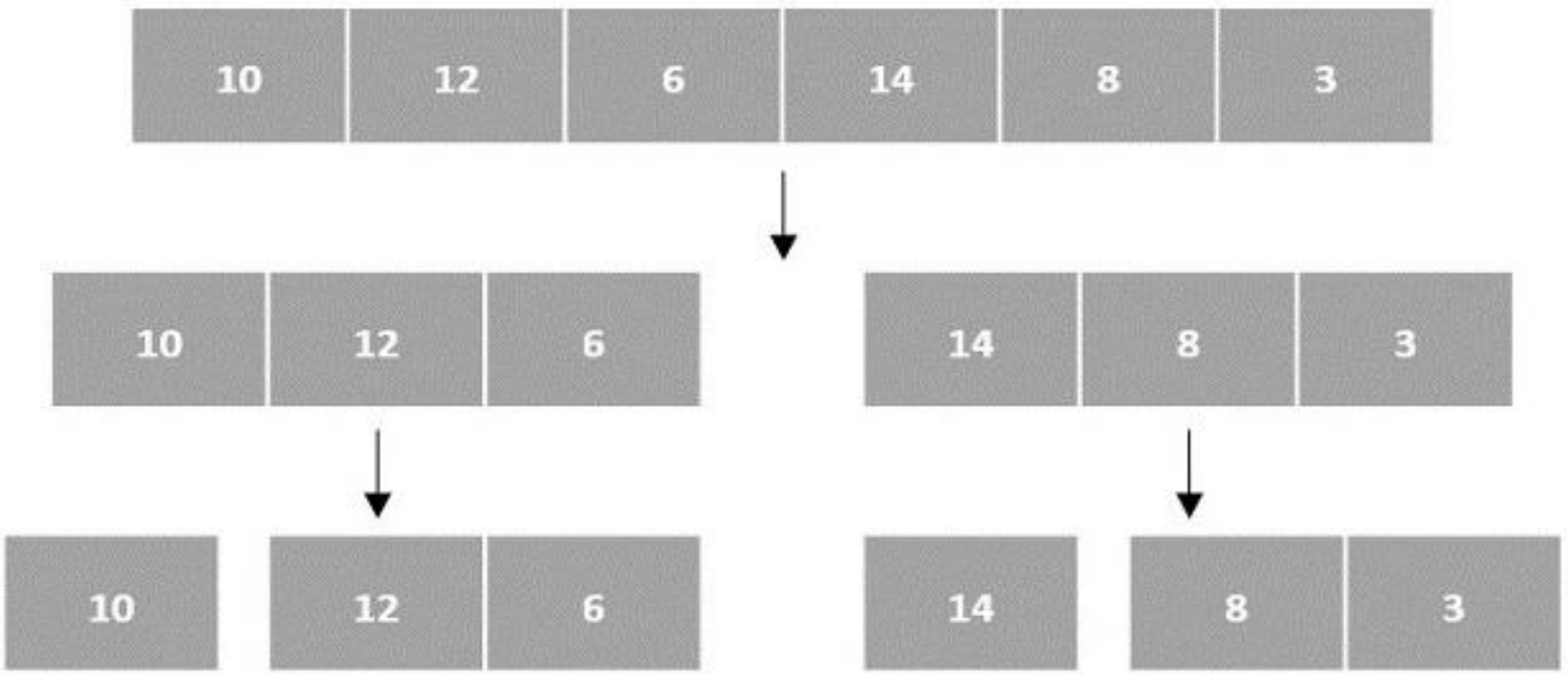
This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

Arrays as Input

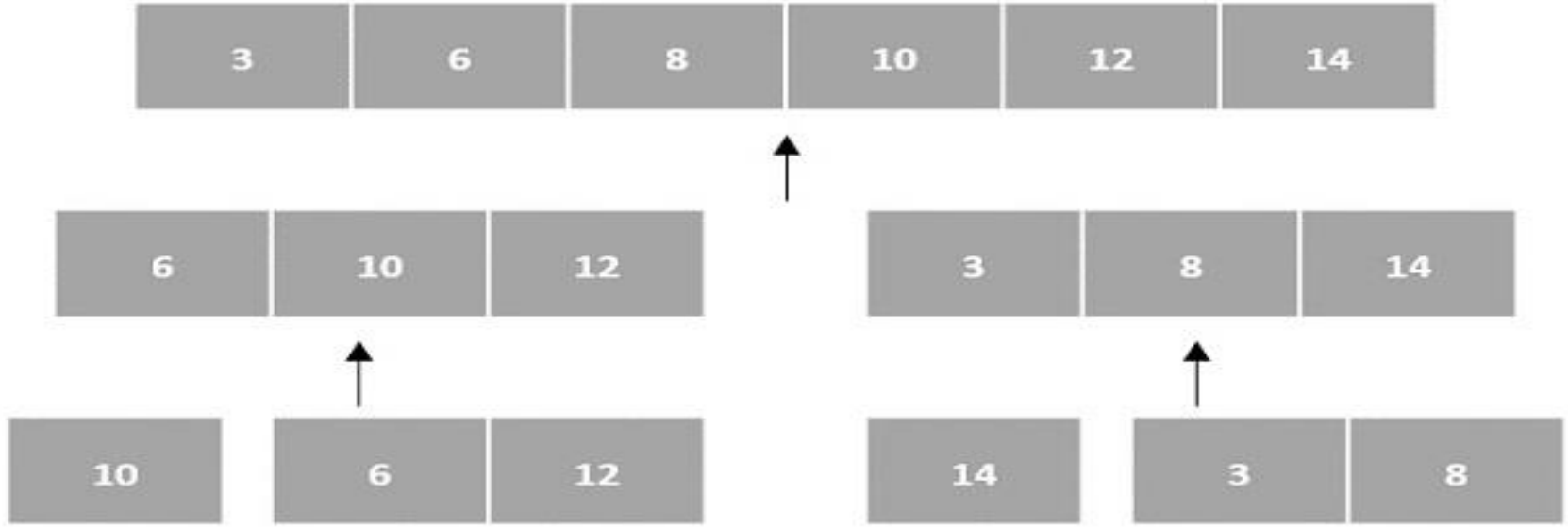
There are various ways in which various algorithms can take input such that they can be solved using the divide and conquer technique.

Arrays are one of them. In algorithms that require input to be in the form of a list, like various sorting algorithms, array data structures are most commonly used.

In the input for a sorting algorithm below, the array input is divided into subproblems until they cannot be divided further.


10
12
6
14
8
3
10
12
6
14
8
3
10
12
6
14
8
3

Then, the subproblems are sorted (the conquer step) and are merged to form the solution of the original array back (the combine step).



Any Query????

Thank you……