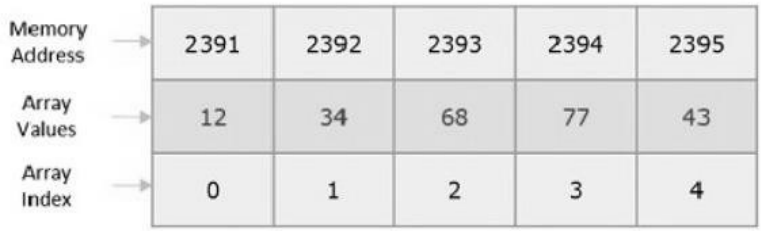
St. Peter's Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. QUESTION BANK						Dept.	:	CSE, CSM, CSD, CSC
						Academic Year 2023-24		
Subject Code	:	AS22-05ES07	Subject	:	Data Structures			
Class/Section	:	B.Tech.	Year	:	I	Semester	:	II

BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

*****

Q. No	Question (s)	Marks	BL	CO
	UNIT - II			
1	a) Explain the representation of a linear array in memory with an example.	1M	L1	C123.2
	The elements of array are stored in successive memory cells. The computer does not keep track of the address of every element of Array, but needs to keep track only the address of the first element of Array and is called the base address of array.  			
	b) Write the formula to find the next element's address in an array .	1M	L3	C123.2
	Using the base address of Array, the computer calculates the address of any element of Array by the formula **LOC (A[K]) = Base Address+ W (K – Lower Bound)** LOC (A [K]) = Address of the element A [K] of the array A Where, w is the number of words/bytes per memory cell for the array, K is the position **Example:** BA=2000 Int a[10]; LOC(a[5])=BA+w(k-LB)=2000+2(5-0) = 2000+10 = 2010			
	c) Why are insertion and deletion considered complicated operations in arrays?	1M	L4	C123.2
	In arrays, elements are stored in contiguous memory locations. When you insert or delete an element, it may require shifting all subsequent elements to make room for the new element or to fill the gap left by the deleted element and leading to a time-consuming operation, especially for large arrays.			

	d) What is the partition algorithm, and how is it used?	1M	L2	C123.3
	The partition algorithm is a fundamental technique in quick sort. The primary purpose of the partition algorithm is to rearrange the elements of an array or list in a way that separates the elements into two groups based on a chosen pivot element. Elements smaller than the pivot are placed to its left, and elements larger than the pivot are placed to its right.			
	e) What is the key requirement for applying binary search to a dataset?	1M	L1	C123.4
	Key requirement for applying binary search to a dataset must be in sorted order (ascending or descending order). binary search depended on the ability to compare elements and eliminate half of the remaining elements in each iteration.			
2	a) How many types of array initialization exist, and explain each with examples	3M	L3	C123.2

	In the C programming language, there are a few ways to initialize arrays. Here are common types of array initialization with examples:
	1. Static Initialization:
	Explicitly specify the elements of the array at the time of declaration.
	Example: -
	• Array initialization with declaration
	`int a[5] = {2,4,6,1,3};`
	• Array initialization with declaration without size
	`int a[] = {2,4,6,1,3};`
	2. Explicit Initialization (Using Loops):
	Initialize array elements explicitly using loops.
	Example: -
	`int a[5];`
	`for (int i = 0; i < 5; ++i)`
	`{`
	`a[i] = i + 1;`
	`}`
	3. Indexed array initialization:
	Initialize the array elements with the help of indexes
	Example: -
	`int a[3];`
	`a[0]=2;`
	`a[1]=4;`
	`a[2]=6;`			

	b) Define an array and explain the process of traversing a linear array with an example	3M	L2	C123.2
	An array is a linear data structure that collects elements of the same data type and stores them in contiguous memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array. Traversing an array means visiting each element of the array in order to perform a specific operation, such as printing, counting, modifying, or searching for values. The process typically involves using a loop to iterate through the elements from the beginning to the end of the array.			

Program: - Printing the elements using traversal operation.

```
Void traversal (int a[],int n)
{
   printf("Elements of the array: ");
   for (int i = 0; i < n; i++)
   printf("%d ", a[i]);
 }
```

	c) List the advantages and disadvantages of using arrays	3M	L4	C123.2

Advantages:

- Random & fast access of elements using array indexes.
- Arrays are simple and easy to understand.
- Using single line code, we can store multiple elements in an array.

Disadvantages: -

- Arrays have a fixed size determined at the time of declaration.
- Inserting or deleting elements in the middle of an array requires shifting the remaining elements, resulting in inefficient time complexity (O(n)).
- It cannot store multiple data type elements.

	d) Explain the bubble sort algorithm and write about its advantages	3M	L2	C123.3

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order until the array is sorted. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Bubble Sort algorithm: -

Step 1: Begin with the first element in the list.
Step 2: Compare the first element with the second element. If they are in the wrong order (e.g., the first element is greater than the second), swap them.
Step 3: Move on to the next pair of elements (the second with the third), and repeat the comparison and swap if necessary.
Step 4: Continue comparing and swapping adjacent elements until you reach the end of the list. After the first pass, the largest element is moved to the end of the list.
Step 5: Repeat the process for the remaining unsorted elements, excluding the last one (since it is already in its correct position after the first pass).
Step 6: Repeat steps 2-5 until the entire list is sorted. Each pass through the list ensures that the largest unsorted element is moved to its correct position.

Advantages:

- Bubble sort is easy to understand and implement.
- It requires minimal additional memory space to perform the sorting.
- Bubble sort is a stable sorting algorithm.

	e) What is the basic idea behind the linear search algorithm and how it works?	3M	L2	C123.4

The linear search algorithm, also known as a sequential search, is a simple method for finding a specific value within a list or array. The basic idea behind linear search is

	straightforward: it checks each element in the list one by one until the target value is found or the entire list has been traversed. **Linear Search Explanation: -** The basic idea behind linear search is to iterate through the elements in a sequential manner, starting from the beginning, and comparing each element with the search element that you are searching in the array. If the current element matches the search element, then it gives "Number is Found", and the index of the matching element is returned. If the entire list is traversed without finding a match, the algorithm concludes that the "Number is not Found" i.e search element is not present in the list. Linear search is straightforward to implement and is effective for small datasets or unordered lists. However, its time complexity is O(n), where n is the number of elements in the list, making it less efficient than some other search algorithms for larger datasets.			
3	a) What are the different methods for inserting elements into an array? Write a C Program for inserting an element in the specified position?	5M	L3	C123.2

	Inserting refers to the operation of adding another element to the existing array. There are various methods for inserting elements into an array in C. We can insert either at starting position or at the specified position or at end in the array.
	Inserting an element at the "end" of the linear array can be easily done provided the memory space allocated for the array is large enough to accommodate the additional element
	Inserting an element in the starting or at a position of the array is difficult because half of the elements must be moved downwards to new locations to accommodate the new element and keep the order of the other elements.
	
	Here's a simple C program that demonstrates this approach:
	
	`#include<stdio.h>`
	`int main()`
	`{`
	`int a[50],pos,i,size,value;`
	`printf("enter no of elements in array:");`
	`scanf("%d",&size);`
	`printf("enter %d elements are:",size);`
	`for(i=0;i<size;i++)`
	`scanf("%d",&a[i]);`
	`printf("enter the position where you want to insert the element:");`
	`scanf("%d",&pos);`
	`printf("enter the value:");`
	`scanf("%d",&value);`			

```
    for(i=size-1;i>=pos-1;i--)
          a[i+1]=a[i];
    a[pos-1]= value;
    printf("final array after inserting the value is");
    for(i=0;i<size+1;i++)
        printf("%d",a[i]);
    return 0;
}
```

Output: -

enter no of elements in array: 5
enter 5 elements are:
10 20 30 40 50
enter the position where you want to insert the element:3
enter the value: 25
final array after inserting the value is
10 20 25 30 40 50

	b) What are the different methods for deleting elements from an array? Discuss the steps involved in deleting an element from a linear array	5M	L3	C123.2

Deleting refers to the operation of removing one element from the existing array. There are various methods for deleting elements from an array in C. We can delete either at starting position or at the specified position or at end in the array.

Deleting an element at the "end" of the linear array can be easily done with difficulties.

Deleting an element in the starting or at a position of the array is difficult because each subsequent elements be moved one location upward to fill up the array.

Deleting at starting Algorithm:

Step 1: Declare an array(a[50]) and "size" variable to store the size of the array.
Step 2: Read the size of the array ('size') from the user.
Step 3: Read the elements in an array by using a for loop to iterate from 0 to n-1.
Step 4: Use a for loop to iterate from 0 to n-1, assign the value of a[i+1] to a[i]. This effectively shifts all elements one position to the left, overwriting the first element.
Step 5: Display the message "After deletion."
Step 6: Decrement the value of 'size' by 1 to represent the new size of the array after deletion
Step 7: Use a for loop to iterate from 0 to n-1, to print each element of the modified array on a new position.

	c) Explain the working mechanism of the selection sort algorithm	5M	L2	C123.3

Selection sort is a simple comparison-based sorting algorithm that divides the input array into two parts: a sorted and an unsorted region. Selection sort is an efficient sorting algorithm that works by repeatedly selecting the smallest element from the unsorted portion of the list and

swapping it to the sorted portion of the list. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted.

Example:-

Let us Consider, an array consisting of 9 elements.

Before First Iteration a[9]={29,72,98,13,87,66,52,51,36};

After the first iteration: {13,72,98,29,87,66,52,51,36} (13 is the minimum and swapped with the first element)

After the second iteration: {13,29,98,72,87,66,52,51,36} (29 is the minimum and swapped with the second element)

After the third iteration: {13,29,36,72,87,66,52,51,98} (36 is the minimum and swapped with the third element)

After the fourth iteration: {13,29,36,51,87,66,52,72,98} (51 is the minimum and swapped with the fourth element)

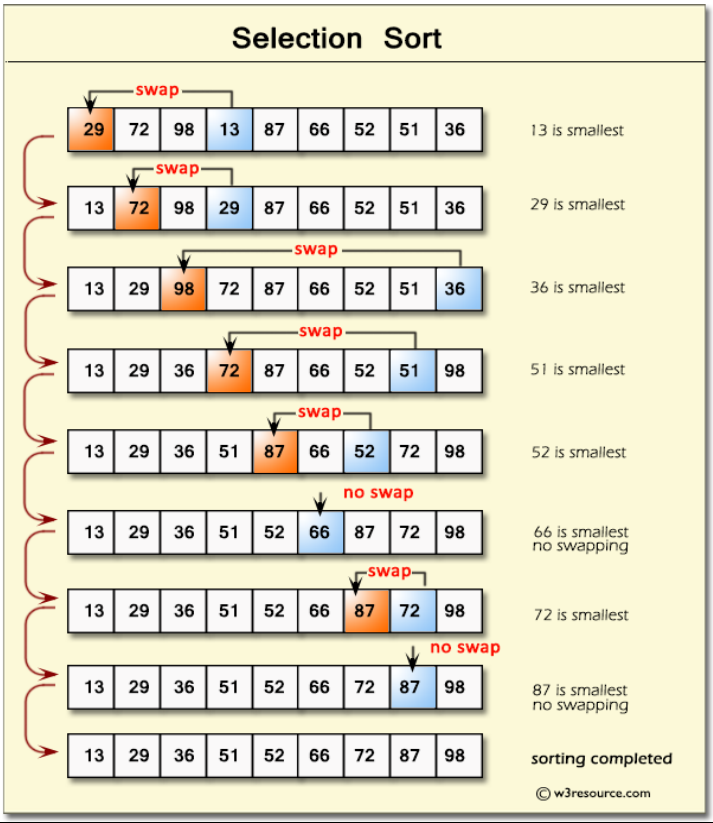
After the fifth iteration: {13,29,36,51,52,66,87,72,98} (52 is the minimum and swapped with the fifth element)

After the sixth iteration: {13,29,36,51,52,66,87,72,98} (66 is the minimum and swapping are not required)

After the seventh iteration: {13,29,36,51,52,66,72,87,98} (72 is the minimum and swapped with the seventh element)

After the eighth iteration: {13,29,36,51,52,66,72,87,98} (87 is the minimum and swapping are not required)

Finally the array is sorted: - a[9]= {13,29,36,51,52,66,72,87,98}



	d) Write a C program to implement the insertion sort method for sorting a given list of integers in ascending order	5M	L3	C123.3

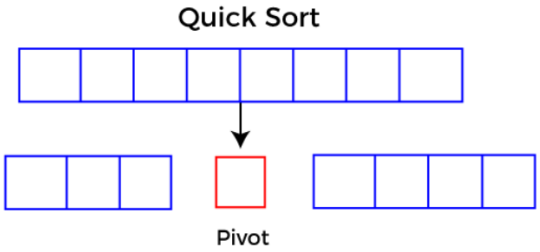
```
#include<stdio.h>
//Insertion Sort
void insertion(int a[],int num)
{
    int i,j,key;
    for(i=1;i<num;i++)
    {
        key=a[i];
        j=i-1;
        while(j>=0 && a[j]>key)
        {
            a[j+1]=a[j];
            j=j-1;
        }
        a[j+1]=key;
    }
}
int main()
{
     int a[50],num,i;
     printf("Enter the number of elements :");
     scanf("%d",&num);
     printf("\nEnter the elements:\n");
     for(i=0; i<num; i++)
          scanf("%d",&a[i]);
     printf("\nElements present in the list are:\n\n");
     for(i=0; i<num; i++)
          printf("%d\t",a[i]);
     printf("\n**Insertion Sort**\n");
     insertion(a,num);
     for(i=0; i<num; i++)
          printf("%d\t",a[i]);
     return 0;
}//end main

Output:
Enter the number of elements :5
Enter the elements:
5       2       4       9       1
Elements present in the list are:
5       2       4       9       1
**Insertion Sort**
1       2       4       5       9
```

	e) Explain the process of Quick Sort with an example	5M	L2	C123.3

Quick Sort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

Pivot is any chosen element from the given array. It serves as a point that is used for partitioning the array into two subarrays. Partitioning places all the elements less than the pivot in the left part of the array, and all elements greater than the pivot in the right part of the array. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.
Conquer: Recursively, sort two subarrays with Quicksort until the one element present in the sub arrays.
Combine: Combine the already sorted array.

Example: -
Let us consider an array having 6 elements.

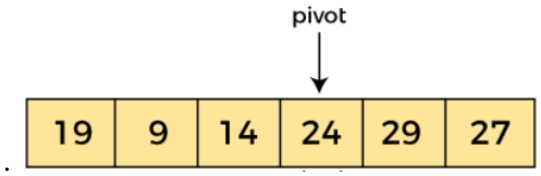
A[6]={24,9,29,14,19,27};

24	9	29	14	19	27

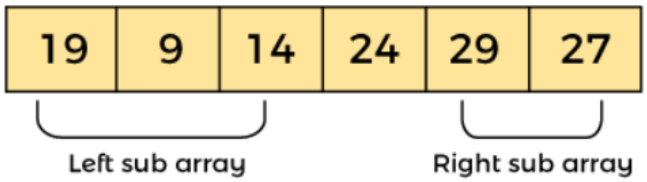
Unsorted Array

Step 1: In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.
Step 2: Since, pivot is at left, so algorithm starts from left to right
Now, a[left]<pivot, so the algorithm moves from forward one position towards right until a value greater than pivot is found and a[right]>pivot, so the algorithm moves from forward one position towards left until a value less than pivot is found.
Step 3: Once the above conditions are failed then swap the a[left] and a[right].

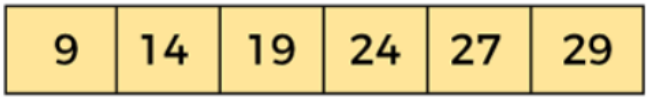
Step 4: repeat step 2 and 3 until left>right. Once the condition true, we need to swap the pivot and a[right] elements. Then element 24, which is the pivot element is placed at its exact position.



Step 5: Divide the array in to two sub arrays, elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.



Step 6: Reat the above steps 1,2,3,4 & 5 that means as the partition process is done recursively, it keeps on putting the pivot in its actual position in the sorted array. Repeatedly putting pivots in their actual position makes the array sorted.



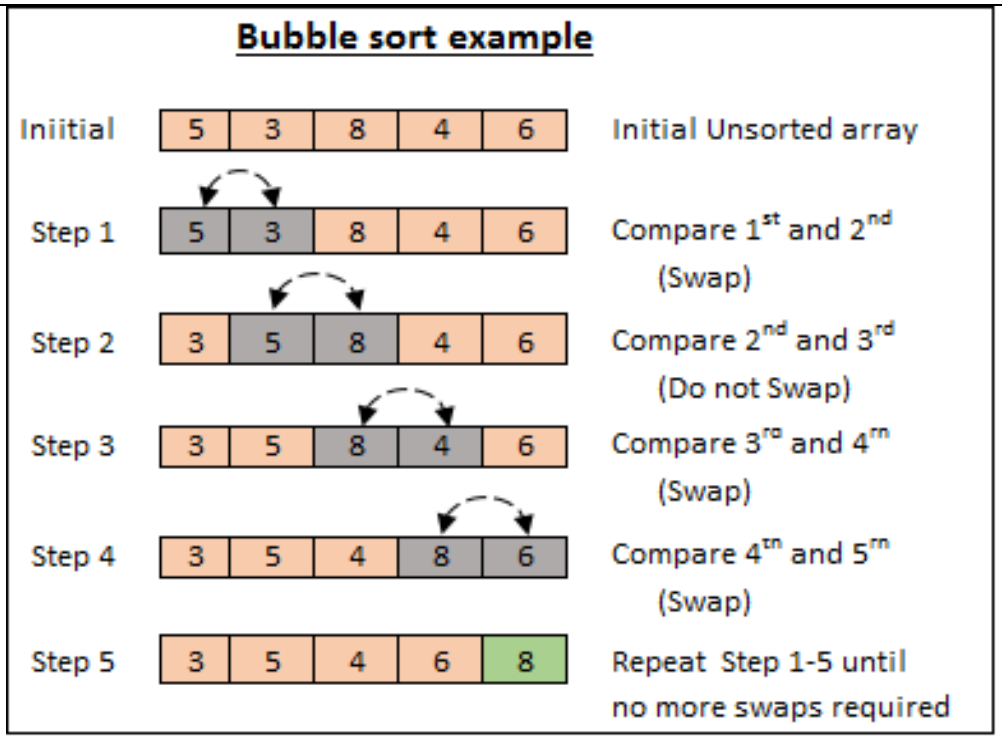
Sorted Array

4	a) What is Bubble Sort? Explain with an example and write a program for Recursive Bubble Sort.	10M	L2	C123.3

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order until the array is sorted. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

In Bubble Sort algorithm,

- Traverse from left and compare adjacent elements and the higher one is placed at right side.
- At the end of first pass, the largest element is moved to the rightmost in the array just like a bubble rising up in water.
- This process is then continued to find the second largest and place it and so on until the data is sorted.



In Each iteration we need follow the same procedure until all elements get in to the proper position.

Program:

```
#include<stdio.h>
//Recursive Bubble Sort
void recursivebubble(int a[],int num)
    {
        int j,temp;
        if(num==0)
          return;
        for(j=0;j<num-1;j++)
            {
                if(a[j]>a[j+1])
                {
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }
            }
        recursivebubble(a,num-1);
     }
```

```
int main()
{
     int a[50],num,i;
     printf("Enter the number of elements :");
     scanf("%d",&num);
     printf("\nEnter the elements:\n");
     for(i=0; i<num; i++)
          scanf("%d",&a[i]);
     printf("\nElements present in the list are:\n\n");
     for(i=0; i<num; i++)
          printf("%d\t",a[i]);
     printf("\n**Recursive Bubble Sort**\n");
     recursivebubble(a,num);
     for(i=0; i<num; i++)
          printf("%d\t",a[i]);
     return 0;
}//end main

Output:
Enter the number of elements :5

Enter the elements:
15   10   20   5    25

Elements present in the list are:

15       10       20       5       25

**Recursive Bubble Sort**
5        10       15       20      25
```

	b) Explain the process of Merge Sort with an example and mention its advantages?	10M	L4	C123.3

The process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted. One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly.

The Working Process of Merge Sort:

If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

Example: -

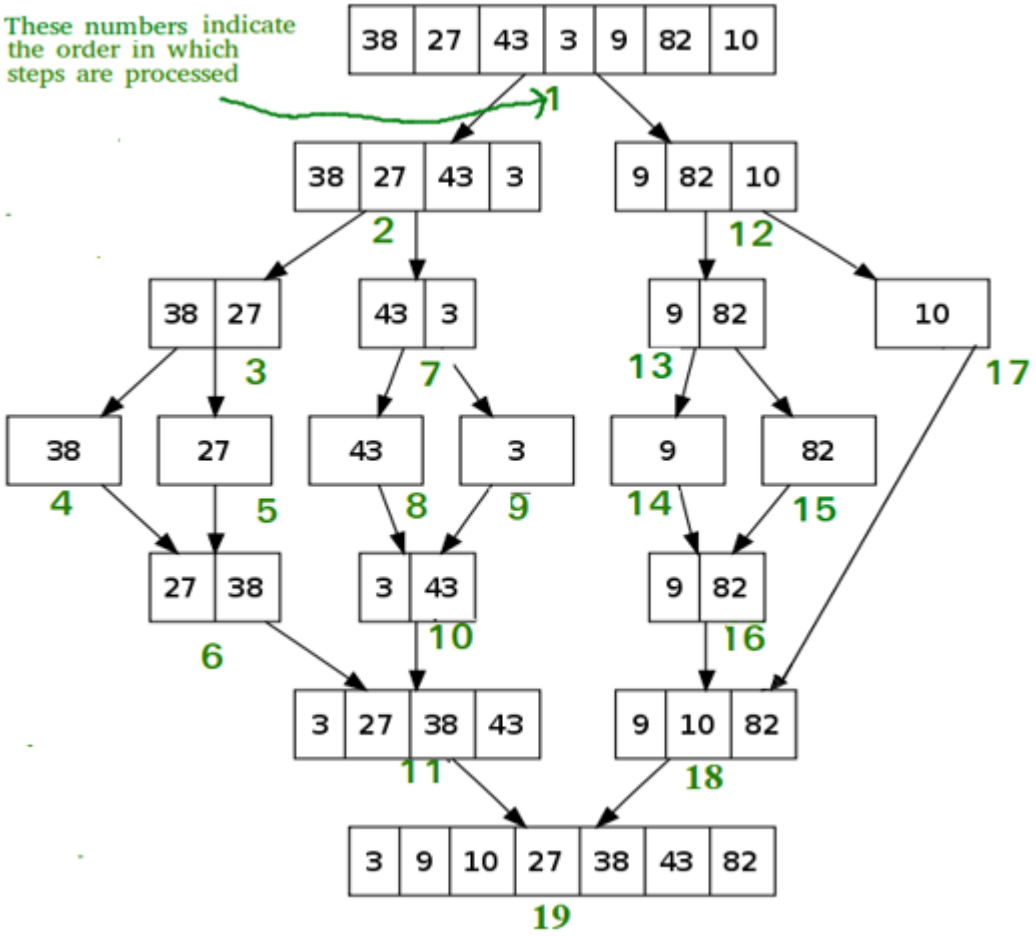
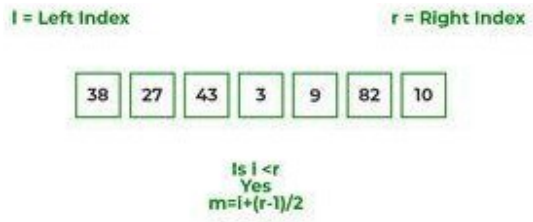


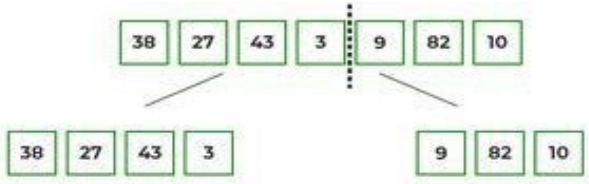
Illustration:

Let us consider an array a[7] = {38, 27, 43, 3, 9, 82, 10}

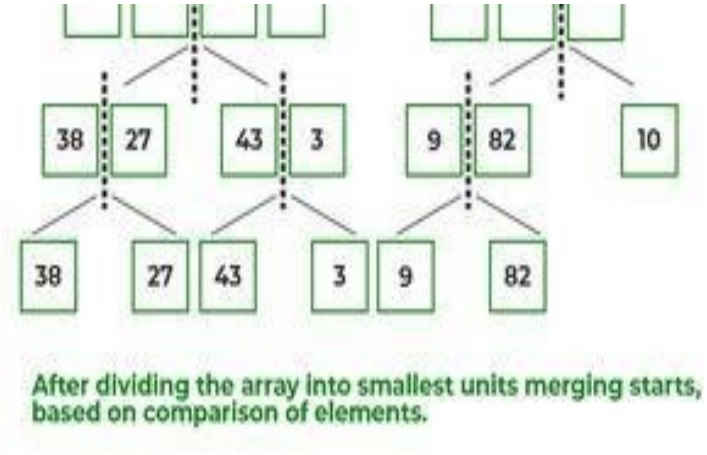
At first, check if the left index of array is less than the right index, if yes then calculate its mid-point.



Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.



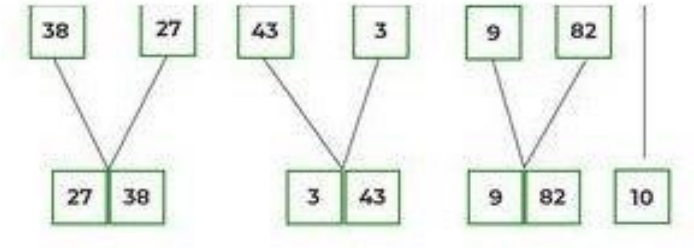
Now, again find that is left index is less than the right index for both arrays, if found yes, then again calculate mid points for both the arrays. Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.



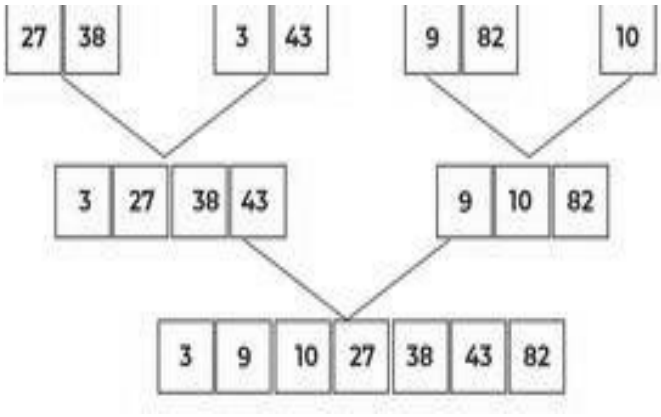
After dividing the array into smallest units merging starts, based on comparison of elements.

After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

Firstly, compare the element for each list and then combine them into another list in a sorted manner.



After the final merging, the list looks like this:



If we take a closer look at the diagram, we can see that the array is recursively divided into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.

	c) Explain the step-by-step process of the binary search algorithm with a suitable example, discussing its efficiency and limitations	10M	L3	C123.4

The binary search algorithm can be used with only a sorted list of elements. That means the binary search is used only with a list of elements that are already arranged in an order This search process starts comparing the search element with the middle element in the list.

Binary search is implemented using following steps...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Find the middle element in the sorted list.
- **Step 3 –** Compare the search element with the middle element in thesorted list.
- **Step 4 -** If both are matched, then display "Given element is found!!!"and terminate the function.
- **Step 5 –** If both are not matched, then check whether the search element is smaller or larger than the middle element.
- **Step 6 –** If the search element is smaller than middle element, repeatsteps 2, 3, 4 and 5 for the left sublist of the middle element.
- **Step 7 –** If the search element is larger than middle element, repeatsteps 2, 3, 4 and 5 for the right sublist of the middle element.
- **Step 8 –** Repeat the same process until we find the search element in the list or until sublist contains only one element.
- **Step 9 –** If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function

Example: -
Consider the following list of elements and the element to be searched

Step 1:
search element (12) is compared with middle element (50)

Both are not matching. And 12 is smaller than 50. So we search only in the left sublist (i.e. 10, 12, 20 & 32).

Step 2:
search element (12) is compared with middle element (12)

Both are matching. So the result is "Element found at index 1"

search element 80

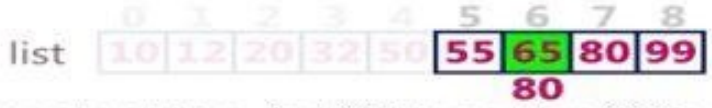
Step 1:

search element (80) is compared with middle element (50)

Both are not matching. And 80 is larger than 50. So we search only in the right sublist (i.e. 55, 65, 80 & 99).

Step 2:

search element (80) is compared with middle element (65)



Both are not matching. And 80 is larger than 65. So we search only in the right sublist (i.e. 80 & 99).

Step 3:

search element (80) is compared with middle element (80)

Both are not matching. So the result is "Element found at index 7"