Unit 5: Sorting and Searching.

1. **Searching**

1. What is Searching. Types of searching.
2. Explain linear search .
3. Write algorithm and program of Linear search.
4. Explain Binary search .
5. Write Algorithm and program of Binary search.

2. **Sorting**

6. What is Sorting. Types of sorting
7. Explain Bubble sort .
8. Write algorithm and program of bubble sort.
9. Explain Selection sort.
10. write algorithm and program of selection sort.
11. Explain insertion sort.
12. Write an algorithm and program for Insertion sort method.
13. Explain Quick sort.
14. write algorithm and program of quick sort.
15. Explain Merge sort.
16. write algorithm and program of Merge sort.

1. What is searching.

Ans:

- Searching is a process of finding element within list of elements that stored in any order or randomly.

- Searching is divided into two categories.
- 1.Linear or sequential search   2.Binary search

2. Explain linear search and Write algorithm and program of Linear search.

Ans:

- It simply sequentially compares the given search value with each value with list from start to end until value is found or list is ended.
- Sequential search works for both sorted and unsorted data.
- If value is found then it will return the position of that search value.

- Ex: 50 45 42 78 65 86 34 40

- Search 78 number and give its position.

First 78 will compare with 50 it does not match
next 78 will compare with 45 it does not match
next 78 will compare with 42 it does not match
next 78 will compare with 78 it matches.
So search value is found in 4th position

- Advantages:
- It is simple as it is easy to understand and implement.

- Disadvantages:
- Search time is not unique. if value is in start position it takes less time and if value is near to end it takes more time.
- Size of data increase search time also increase.

3. Algorithm of linear search.

Ans:

- Algorithm

```
    Algo. Linear search(L,N,X)
L→List of elements
N→number of elements in list
X→value to be searched in list

Step 1: [initialization]
        K←0
        Flag←1
```

```
Step2: Repeat step 3 for k=0,1,2,..N-1

Step 3: if(L(k) = = X)
        then
        Fiag←0
        Write “search successful”
        Write “ value found at location k+1”

Step 4: if(flag==1)
        Then
        Write “search is unsuccessful”

Step 5:[finished]
        Exit
```

- **Programme**

```
#include<stdio.h>
void main()
{
    int L[50],n,X,flag=0,i;
    printf("\n\n Enter Size of Array:->");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            scanf("%d",&L[i]);
    }
    printf("\n\nEnter Element To Search:->");
    scanf("%d",&X);
    for(i=0;i<n;i++)
    {
            if(  X == L[i])
            {
                    printf("data found in position %d",i+1);
                    flag=1;
                    break;
            }
    }
if(flag==0)
printf("data not found");
getch();
}
```

- Output 1: enter size of array / Output 2: enter size of array

```
Output 1:  enter size of array                    Output 2:  enter size of array
           5                                                  5
           12  10  23  45  5                                 12  10  23  45  5
           Enter element to search                           Enter element to search
           23                                                55
           Data found in position 2                          data not found
```

4. Explain Binary search.

Ans:

- Binary search requires sorted list.
- In binary search first it computes the middle ,which is index of the middle value in the list as
- Middle=(start +end)/2 ,Start=index of first element End=index of last element
- Then compare search value with middle value.
- If both matches then value is found and return middle position.
- If both does not match then list is divided into two parts called upper half and lower half.
- All values in upper half is greater than middle value and All values in lower half is less than middle value.
- If search value < middle value then

  Search will start in lower half

  start=start End=middle -1

  Again count middle value and compare until search found
- If search value > middle value then

  Search will start in upper half

  start=middle +1 End=end

  Again count middle value and compare until search found

- Ex. 9 , 12 , 24 , 30 , 36, 45 ,70 Search 45

9	12	24	30	36	45	70
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

Step 1 start=0 end=6 So middle=(0+6)/2 middle =3

0	1	2	3	4	5	6
9	12	24	30	36	45	70

Start middle end

Step 2 45 >30
So start =middle + 1 = 3+1 = 4 End = end =6

Step 3 middle= (4+6)/2 =10/2 =5

4	5	6
36	45	70

Start middle end

Now a[mid]=a[5]=45
So search value 45 =middle value 45
Search successful position is 5

5. Algorithm and programme of Binary search.

Ans:

- **Algorithm**

```
      Algo. Binary search(L,N,X)
L→List of elements
N→number of elements in list
X→value to be searched in list

Step 1: [initialization]
        start←0
        end←N-1
        Flag←1

Step2: while (start<=end)  Repeat step 3 to 4

Step 3: middle = (start+end) /2

Step 4: if ( X <L[middle])   Then
End=middle-1
Else  if (X > L[middle])   Then
```

```
Start =middle +1
Else if(X==L[middle])
Write " search successful ,middle+1"
Flag←0

Step 4:  if  (flag==1)
Then Write " search unsuccessful"

Step 5:[finished]
        Exit
```

- **Programme**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int L[50],n,middle,X,flag=0,i,start,end;
    clrscr();
    printf("\n\n Enter Size of Array:->");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
          scanf("%d",&L[i]);
    }
    printf("\n\nEnter Element To Search:->");
scanf("%d",&X);
start=0;
end=n-1;
while (start<=end)
{
middle = (start+end) /2;
if ( X <L[middle])
end=middle-1;
else if (X > L[middle])
start =middle +1;
elseif(X==L[middle])
{
printf( " search successful %d ", middle+1);
flag=0 ;
break;
}
}
     if  (flag==1)
     {
printf( " search unsuccessful");
}
getch();
}
```

```
Output 1:  enter size of array                              Output 2:  enter size of array
             5                                                           5
             12  20  23 24  25                                          12  20  23 24  25
             Enter element to search                                    Enter element to search
             23                                                         55
             Data found in position 2                                   data not found
```

6. What is sorting .Classify various sorting method.

Ans :

- Sorting is an arrangement of data in some given sequence i.e. increasing order or decreasing order.
- There are so many sorting tech. available.
- Sorting methods:

1) Bubble sort
2) Selection sort
3) Quick sort
4) Insertion sort
5) Merge sort
6) Radix sort

7. Explain Bubble sort.

Ans :

- It is simple and easy tech.
- In this tech., start with first element .
- each element is compared with its adjacent(next) element.
- If first element is larger than second element then position of elements are interchanged(swap that two numbers) otherwise it is not changed.

- Then next element is compared with its adjacent element and same process is repeated for all elements.
- Each pass places maximum value in last position.
- Ex: 1 10 5 2

Pass 1:	step 1	step 2	step 3
1	1	1	1
10	10	5	5
5	5	10	2
2	2	2	10
1< 10 no chang	10>5 swap	10>2 swap	

Pass 2:	step 1	step 2
1	1	1
5	5	2
2	2	5
10	10	10
1< 5 no change	5>2 swap	

Pass 3:	step 1
1	1
2	2
5	5
10	10
1< 2 no change	

8. Write Algo and Programme for Bubble sort.

Ans :

- Algorithm:

```
Step 1:  [Initialization ]
         i←0

Step 2:  repeat step 3 to 7  while  i < n-1

Step 3:  set j ← 0

Step 4:  repeat step 5 to 6  while  j < n - i- 1

Step 5:  if a[ j ] > a [ j+1] then
         Set temp = a[ j ]
         Set a[ j ]  = a[ j+1]
         Set a[ j+1] = temp
         End if

Step 6:  j=j+1

Step 7:  i=i+1

Step 8:  [finished]
         Exit
```

- **Programme:**

```
void main()
{
int i,j,a[10],temp,n;
clrscr();
printf("enter size of array");
scanf("%d",&n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n-1;i++)
{
        for(j=0;j<n-i-1;j++)
        {
                if(a[j]>a[j+1])
                {
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                   }
              }
}
for(i=0;i<n;i++)
printf("\n%d",a[i]);
getch();
}
```

- Output 1 enter size of array

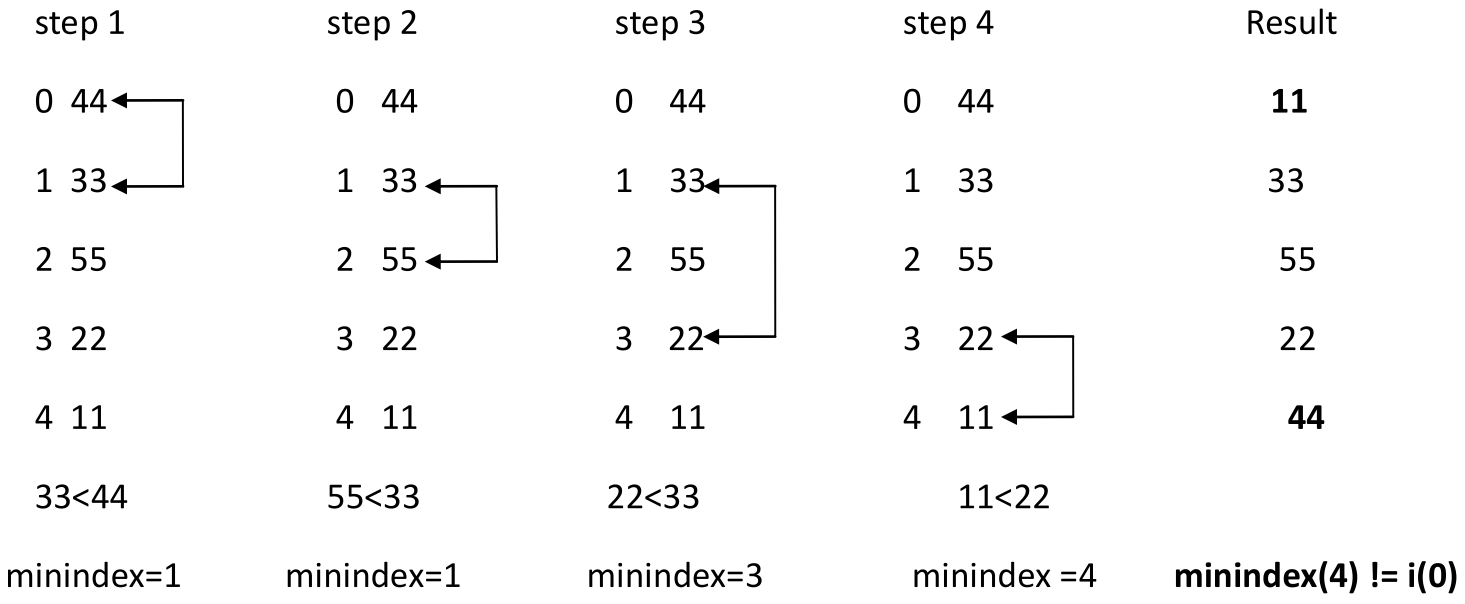
```
            5
             10  2  1  3  12
  Ans:       1  2 3  10  12
```

9. Explain Selection sort.

Ans :

- ➢ Initialize the minindex with i variable.
- ➢ Compare one by one element with minindex element.
- ➢ If any element is found which having value less than the minindex element then "change the minindex".
- ➢ At the end of each pass if minindex value is updated then only swap two elements.
- ➢ Ex: 44 33 55 22 11

Pass 1: i=0 minindex=0



step 1	step 2	step 3	step 4	Result
0 44	0 44	0 44	0 44	**11**
1 33	1 33	1 33	1 33	33
2 55	2 55	2 55	2 55	55
3 22	3 22	3 22	3 22	22
4 11	4 11	4 11	4 11	**44**
33<44	55<33	22<33	11<22	
minindex=1	minindex=1	minindex=3	minindex =4	**minindex(4) != i(0)**

- swap a[minindex] and a[i]   swap a[4] and a[0] swap 11 and 44

<u>Pass 2:</u> i=1 minindex=1

step 1	step 2	step 3	Result
0 11	0 11	0 11	11
1 33	1 33	1 33	**22**
2 55	2 55	2 55	55
3 22	3 22	3 22	**33**
4 44	4 44	4 44	44
55<33	22<33	44<22	
minindex=1	minindex=3	minindex=3	**minindex(3) != i(1)**

- swap a[minindex] and a[i]   swap a[3] and a[1] swap 22 and 33

<u>Pass 3:</u> i=2 minindex=2

step 1	step 2	Result
0 11	0 11	11
1 22	1 22	22
2 55	2 55	**33**
3 33	3 33	**55**
4 44	4 44	44
33<55	44<33	
minindex=3	minindex=3	**minindex(3) != i(2)**

- swap a[minindex] and a[i]   swap a[3] and a[2] swap 33 and 55

<u>Pass 4:</u> i=3 minindex=3

step 1		Result	
0	11	0	11
1	22	1	22
2	33	2	33
3	55	3	44
4	44	4	55

44<55

minindex=4 **minindex(4) != i(3)**

- swap a[minindex] and a[i] swap a[4] and a[3] swap 44 and 55

10. Write Algo and Programme for selection sort.
Ans :

- **<u>Algorithm:</u>**

```
Step 1:   [Initialization ]
          i←0

Step 2:   repeat step 3 to 7 while  i < n-1

Step 3:   set minindex ← i
          j ← i+1

Step 4:   repeat through step 5 while  j <= n -1

Step 5:   if a[j ] < a [ minindex ] then
          set  minindex = j
          End if

Step 6:   j=j+1
Step 7:    if  minindex != i then
           temp=a[i]
           a[i]=a[minindex]
           a[minindex]=temp

Step 8:    i=i+1

Step 9:    [finished]
           Exit
```

- **Programme:**

```
#include<stdio.h>
void main()
{
    int a[5],minindex,i,j,n,temp;
    printf("\n\n Enter Size of Array:->");
    scanf("%d",&n);
    printf("\n\n Enter List to Sort");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<n-1;i++)
    {
        minindex=i;
        for(j=i+1;j<=n-1;j++)
        {
            if(a[j]<a[minindex])
            {
                minindex=j;
            }
        }
        if(i!= minindex)
        {
            temp=a[minindex];
            a[minindex]=a[i];
            a[i]=temp;
```

```
                }
        }
        printf("\n\n Array in Ascending Order is :->");
        for(i=0;i<n;i++)
        {
                printf("\n\na[%d] ==> %d",i,a[i]);
        }
}
```

- Output : enter size of array

```
                5
                 10  2  1  3  12
  Ans:           1  2 3  10  12
```

11. Explain Insertion sort.

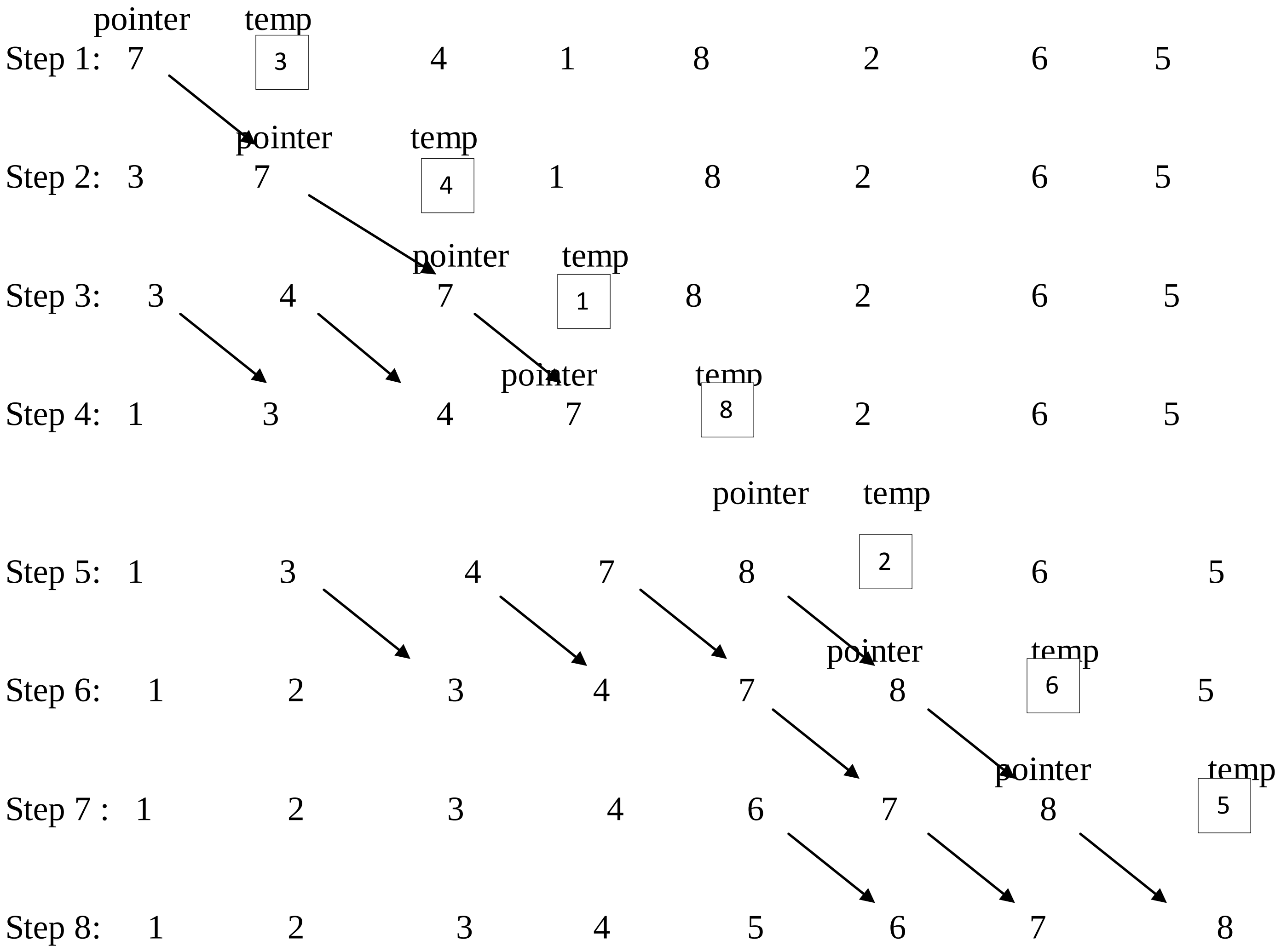
Ans :

- In insertion sort element is inserted at appropriate place.
- It is simple to implement and efficient for small data set.
- For ex:to sort cards in one's hand one extract card,shift remaining cards,then insert extracted card in the correct place.
- This process is repeated until all the cards are in the correct sequence.

- <u>Working</u>

- In each pass element is considered as input element and is compared with all the elements that appear before it.
- As there is no element before first element we can directly start from second element.
- All the element which are greater than input element are shifted by one position.
- At the end input element can be stored in the free slot.

- Ex: 7 3 4 1 8 2 6 5



Step 1: 7 3 4 1 8 2 6 5

Step 2: 3 7 4 1 8 2 6 5

Step 3: 3 4 7 1 8 2 6 5

Step 4: 1 3 4 7 8 2 6 5

Step 5: 1 3 4 7 8 2 6 5

Step 6: 1 2 3 4 7 8 6 5

Step 7 : 1 2 3 4 6 7 8 5

Step 8: 1 2 3 4 5 6 7 8

12. Write Algo and Programme for Insertion sort.
Ans :

a: list of elements
N:number of elements in list

- **Algorithm**:

```
Step 1:   repeat through step 2 to 4 For  i=1,2 ,3,4 … n-1

Step 2:   temp=a[i]
          Pointer= i- 1

Step 3:   repeat step 4 while (a[pointer]>temp)&&(pointer>=0))

Step 4:   a[pointer + 1]=a[pointer]
```

```
            Pointer = pointer - 1

Step 5:     a [pointer + 1] = temp

Step 6:     [finished]
            Exit
```

- **Programme:**

```
void main()
{
    int a[5],temp,n,i,pointer;
    clrscr();
    printf("Enter the Size of Array:->");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=1;i<=n-1;i++)
    {
        temp=a[i];
        pointer=i-1;
        while((a[pointer]>temp)&&(pointer>=0))
        {
            a[pointer+1] = a[pointer];
            pointer=pointer - 1;
        }
        a[pointer+1]=temp;
    }
    for(i=0;i<n;i++)
    {
        printf("\n\n %d",a[i]);
    }
    getch();
}
```

- Output : enter size of array

```
          5
           10  2  1  3  12
  Ans:     1  2 3  10  12
```

13. Explain Quick sort.

Ans :

- It is very efficient method for larger lists.
- It use divide and conquer technique.
- Step -1

An element is chosen as pivot element based on the value of pivot the array is splited into two subarray in such way that each element in left array is less than or equal to pivot element and each element in right array is greater than the pivot element.

- ------		- ----
elements<=pivot	pivot	elements>pivot

- Step -2

  Conquer:Recursively sort two subarray
- Step -3

  Combine :- Combine all sorted elements in a group to form list of sorted elements.

If a[I]<=a[pivot] → i++ ,If a[J] > a[pivot] →J- -

When I and J stop and If I < J → interchange a[I] and a[J]

If I > J → interchange a[pivot] and a[J]

Ex: **33,77,44,55,88,11,22,66**

0	1	2	3	4	5	6	7
33	77	44	55	88	11	22	66
i,pivot							j

33<=33 yes -> i++ 66>33 yes -> j - -

0	1	2	3	4	5	6	7
33	**77**	44	55	88	11	**22**	66
	i					j	

77<33 22> 33 no -> stop check i<j yes swap (a[i],a[j])

0	1	2	3	4	5	6	7
33	22	44	55	88	11	77	66
	i					j	

22<33 yes → i++ 77>33 yes → j- -

0	1	2	3	4	5	6	7
33	22	44	55	88	11	77	66
		i			j		

44<33 11>33 no -> stop check i<j yes swap (a[i],a[j])

0	1	2	3	4	5	6	7
33	22	11	55	88	44	77	66
		i			j		

11<33 yes → i++ 44>33 yes → j--

0	1	2	3	4	5	6	7
33	22	11	55	88	44	77	66
			i	j			

55<33 no -> stop 88>33 yes → j--

0	1	2	3	4	5	6	7
33	22	11	55	88	44	77	66
			i, j				

55<33 no -> stop 55>33 yes → j—

0	1	2	3	4	5	6	7
33	22	11	55	88	44	77	66
		j	i				

55<33 no ->stop 11>33 no ->stop now check i<j no → swap**(a[pivot],a[j])**

0	1	2	3	4	5	6	7
11	22	33	55	88	44	77	66
		j	i				

now call function recursively

- quick_sort(a,low,j-1)   quick_sort(a, 0 ,1)
- quick_sort(a,j+1,high)   quick_sort(a , 3 ,7)

- quick_sort(a, 0 ,1)

0	1
11	22
i	j

11<=11 yes i++   22>11 yes j- -

0	1
11	22
j	i

now check i<j   no → swap**(a[pivot],a[j])**

0	1
11	22

- quick_sort(a,3,7)

0	1	2	3	4	5	6	7
			55	88	44	77	66
			i				j

55<=55   yes → i++   66>55   yes → j--

0	1	2	3	4	5	6	7
			55	88	44	77	66
				i		j	

55<88 no 77>55 yes → j - -

0	1	2	3	4	5	6	7
			55	88	44	77	66
				i	j		

55<88 no 44>55 no check i<j yes swap (a[i],a[j])

0	1	2	3	4	5	6	7
			55	44	88	77	66
				i	j		

44<55 yes i++ 88>55 yes j—

0	1	2	3	4	5	6	7
			55	44	88	77	66
				j	i		

88<55 no 44>55 no check i<j no swap (a[pivot],a[j])

0	1	2	3	4	5	6	7
			44	55	88	77	66
				j	i		

- quick_sort(a, 3, 3) : no processing
- quick_sort(a , 5 ,7)

- **quick_sort(a , 5 ,7)**

0	1	2	3	4	5	6	7
					88	77	66
					i		j

88<=88 yes i++ 66>88 no

0	1	2	3	4	5	6	7
					88	77	66
							i j

77<88 yes i++ 66>88 no check i<j no swap (a[pivot],a[j])

0	1	2	3	4	5	6	7
					66	77	88
							i j

- quick_sort(a, 5, 6)
- quick_sort(a , 9 ,7) : no processing

- **<u>quick_sort(a , 5 ,6)</u>**

0	1	2	3	4	5	6	7
					66	77	
					i	j	

66<=66 yes i++ 77>66 yes j--

0	1	2	3	4	5	6	7
					66	77	
					j	i	

check i<j no swap (a[pivot],a[j])

0	1	2	3	4	5	6	7
					66	77	

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	**22**	**33**	**44**	**55**	**66**	**77**	**88**

14. Write Algo and Programme for Quick sort.

Ans :

a: list of array
low:leftmost element in array.
high:rightmost element in array.
pivot:As key value we chose first element.

- **Algorithm:**

```
Step 1:   if (low <high) then perform step 2- 11
Step 2:   pivot<-low
          i<-low
          j<-high

Step 3:   while (i<j) then perform step 4-9


Step 4:   repeat step 5 while(a[i]<=a[pivot]  && i<high)
Step 5:    i← i + 1

Step 6:    repeat step 7 while (a[j]>a[pivot])
Step 7:     J<-   J – 1

Step 8:    if (i<j) then perform  step 9
Step 9:   temp  ← a[i]
           a[i] ← a[j]
            a[j]<-temp

Step 10:  temp  ← a[pivot]
           a[pivot] ← a[j]
            a[j]<-temp
```

```
Step 11:    Call Quick sort(a,low,j – 1)
            Call Quick sort(a,j + 1 ,high)

Step 12:    [finished]
            Exit
```

- **Programme:**

```
#include<stdio.h>
#include<stdlib.h>
void quicksort(int ,int);
int a[20];
void main()
{
      int i,j,temp,n;
      printf("\n \tEnter size \n");
      scanf("%d",&n);
      printf("\n \tEnter the value of array \n");
      for(i=0;i<n;i++)
      {

              printf("\n \tEnter A[%d] ",i);
              scanf("%d",&a[i]);
      }
      printf("\n\n \tArray before sortting \n");
      for(i=0;i<n;i++)
      {
              printf("\n \tEnter A[%d]=%d \n",i,a[i]);
      }
      quicksort(0,n-1);
      printf("\n\n \tArray after quick short \n");
      for(i=0;i<n;i++)
      {
              printf("\n \tEnter A[%d]=%d \n",i,a[i]);
      }
}
void quicksort(int low,int high)
{
      int pivot,temp,i,j;
      if(low<high)
```

```
    {
        pivot=low;
        i=low;
        j=high;
        while(i<j)
        {
            while(a[i]<=a[pivot] && i<high)
            {
                i=i+1;
            }
            while(a[j]>a[pivot])
            {
                j=j-1;
            }
            if(i<j)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }

        }
        temp=a[pivot];
        a[pivot]=a[j];
        a[j]=temp;
        quicksort(low,j-1);
        quicksort(j+1,high);
    }
}
```

- Output : enter size of array

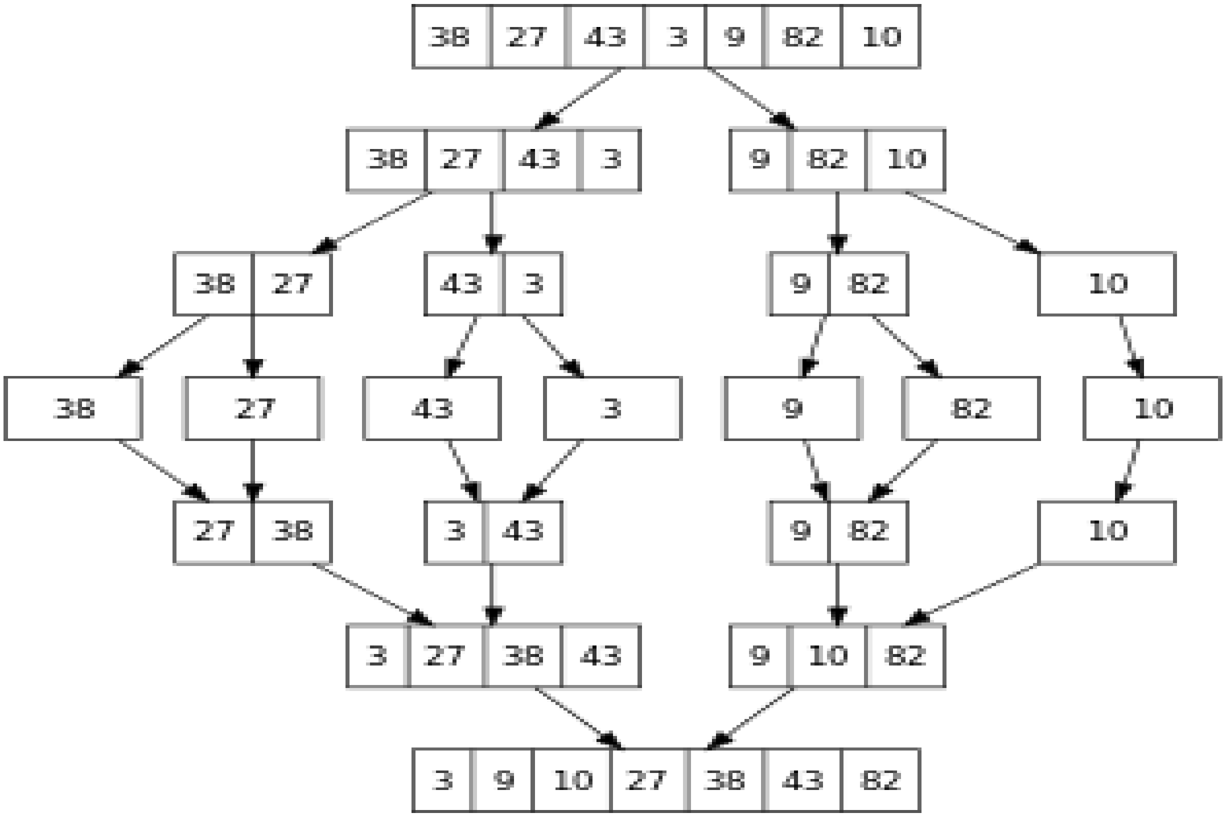
```
         5
          10  2  1  3  12
 Ans:     1  2 3  10  12
```

15. Explain Merge sort.

Ans :

- a merge sort works as follows:

1. Divide the unsorted list into *n* sublists, each containing 1 element (a list of 1 element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.



Ex Sort following elements using merge sort
80, 40, 50, 20, 70, 30, 10, 60

⇒ Mergesort(A, $\overset{l}{0}$, $\overset{h}{7}$)

0	1	2	3	4	5	6	7
80	40	50	20	70	30	10	60
↑ low			↑ mid				↑ high

✓ merge sort (A, 0,3)
✓ merge sort (A, 4,7)
✓ Combine (A, 0,3,7)

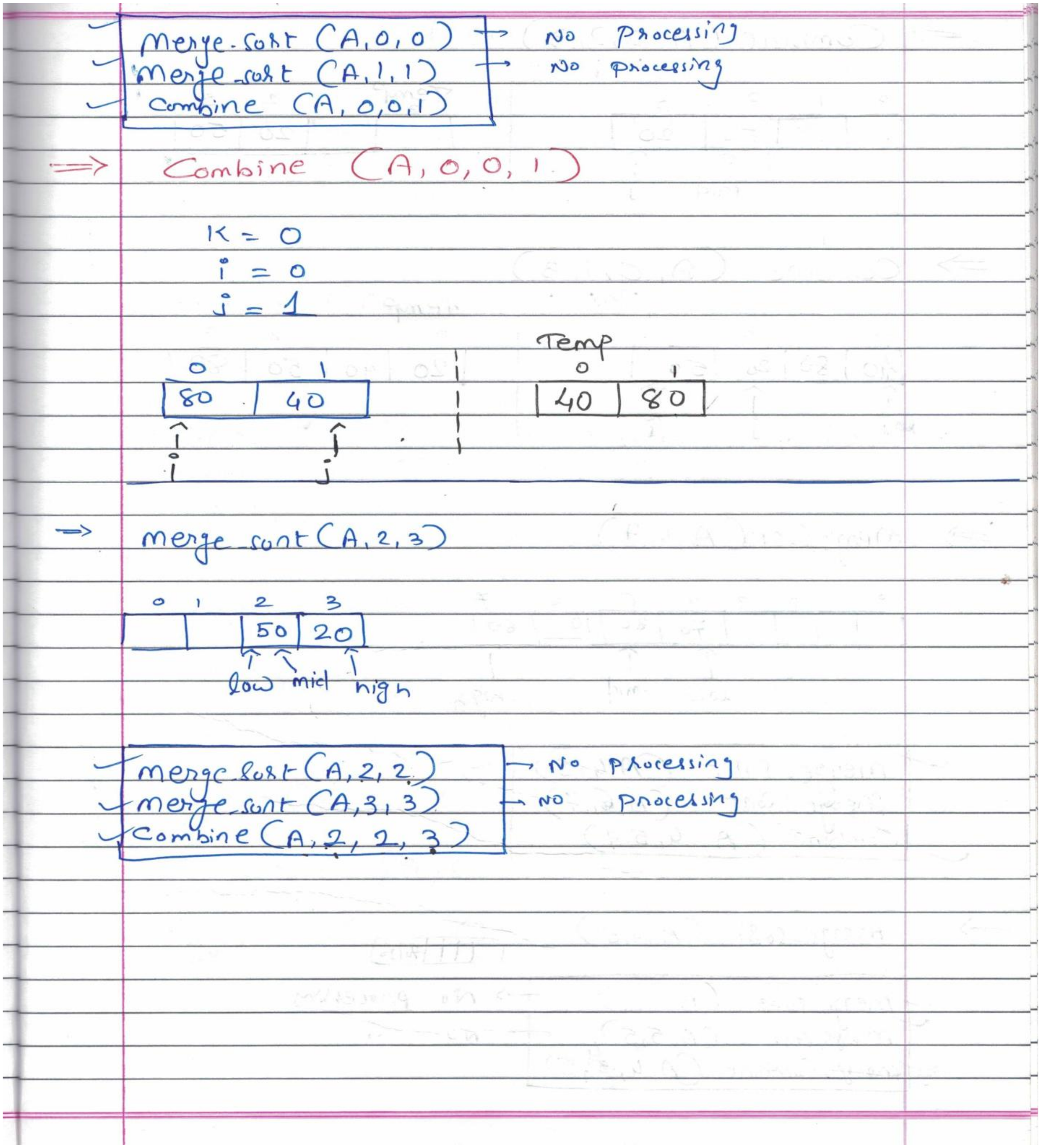
⇒ merge_sort (A, 0,3)

0	1	2	3
80	40	50	20
↑ low	↑ mid		↑ high

✓ merge sort (A, 0, 1)
✓ merge sort (A, 2, 3)
✓ combine (A, 0, 1, 3)

⇒ merge sort (A, 0, 1)

0	1
80	40
↑ low ↑ mid	↑ high



Merge-sort (A,0,0) → No processing
Merge-sort (A,1,1) → No processing
Combine (A,0,0,1)

⇒ Combine (A,0,0,1)

K = 0
i = 0
j = 1

0	1
80	40
↑ i	↑ j

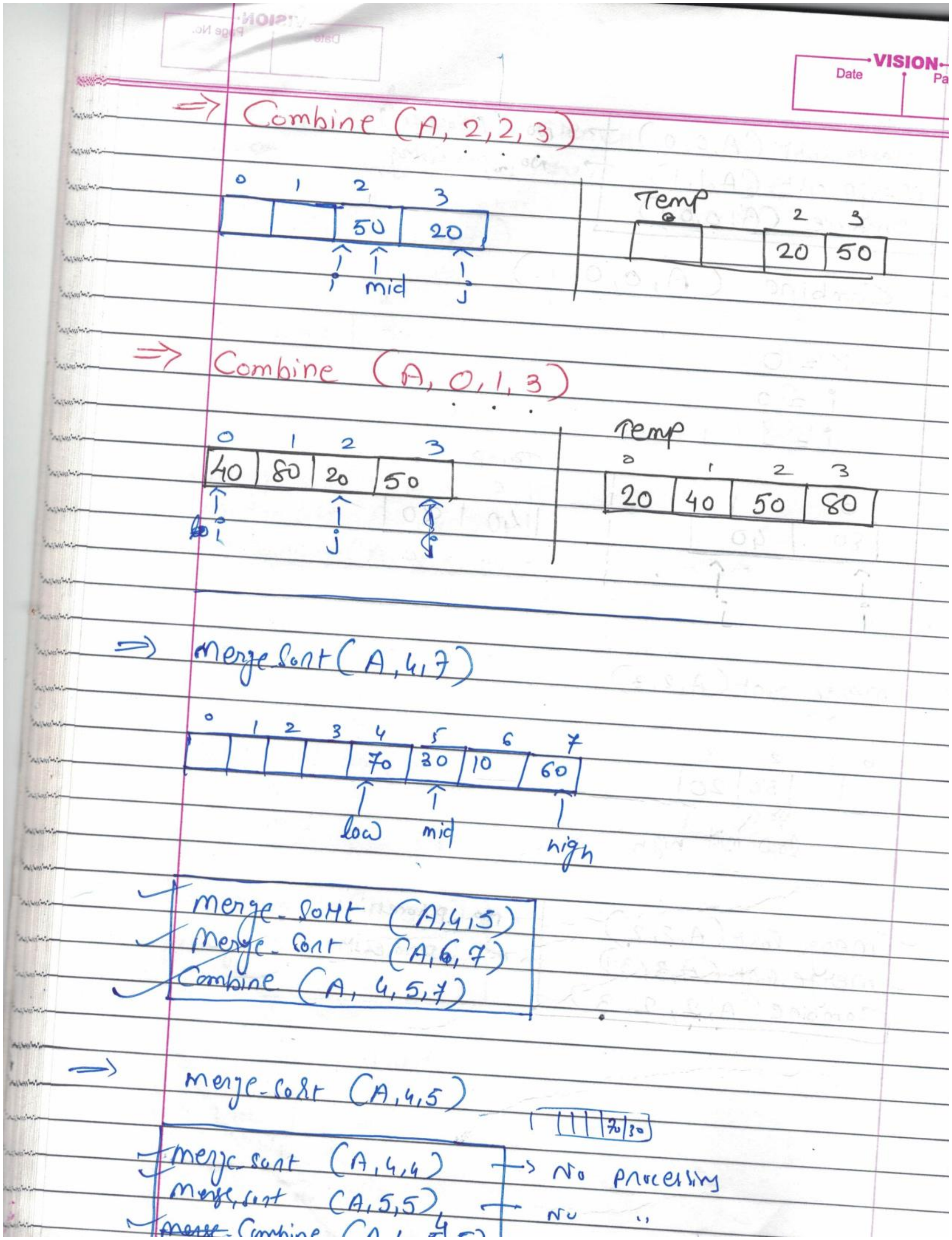
Temp

0	1
40	80

⇒ merge-sort (A,2,3)

0	1	2	3
		50	20
		↑ low ↑ mid	↑ high

merge-sort (A,2,2) → No processing
merge-sort (A,3,3) → No processing
Combine (A,2,2,3)



⇒ Combine (A, 2,2,3)

0	1	2	3
		50	20
		i mid	j

Temp

0	1	2	3
		20	50

⇒ Combine (A, 0,1,3)

0	1	2	3
40	80	20	50
i		j	

Temp

0	1	2	3
20	40	50	80

⇒ merge sort (A,4,7)

0	1	2	3	4	5	6	7
				70	30	10	60
				low	mid		high

merge-sort (A,4,5)
Merge-sort (A,6,7)
Combine (A, 4,5,7)

⇒ merge-sort (A,4,5)

			70	30

merge sort (A,4,4) → No processing
Merge sort (A,5,5) → No ''
~~merge~~ Combine (A, 4, 4, 5)

⇒ Combine (A, 4, 4, 5)

0	1	2	3	4	5
				70	30
				l mid	j

Temp

0	1	2	3	4	5
				30	70

⇒ merge sort (A, 6, 7)

✓ merge-sort (A, 6,6) → No processi

✓ mergesort (A, 7, 7) → No P

✓ Combine (A, 6, 6, 7)

⇒ Combine (A, 6, 6, 7~~8~~)

0	1	2	3	4	5	6	7
						10	60
						i	j

Temp.

0	1	2	3	4	5	6	7
						10	60

~~merge sort (A, 6, 6, 7)~~

⇒ ~~merge-sort (A, 6, 7)~~

~~mergesort (A, 6, 6)~~

⇒ Combine (A, 4, 5, 7)

0	1	2	3	4	5	6	7
				30	70	10	60
				↑ i		↑ j	

Temp

				4	5	6	7
				10	30	60	70

{ At this point A will be

20	40	50	80	10	30	60	70

}

⇒ Combine (A, 0, 3, 7)

0	1	2	3	4	5	6	7
20	40	50	80	10	30	60	70
↑ i			↑ mid	↑ j			↑ high

Temp

0	1	2	3	4	5	6	7
10	20	30	40	50	60	70	80

16. Write Algorithm and program of **Merge sort.**

Ans :

- **Algorithm:**

```
Step 1:   if (low <high) then
          mid =(low + high)/2
          mergesort(a ,low ,mid)
          mergesort(a, mid+1 ,high)
          combine (a, low ,mid ,high)

Step 2:   combine(a,low,mid,high)
          k<-low
          i<-low
          j=mid +1

Step 3:   while (i<=mid  &&  j<= high) do step 4

Step 4:   if(a[i] <= a[j])
          temp [k] =a[i]
          i ← i+ 1
          k← k+1
          else
          temp[k] =a[j]
          j← j+1
          k←k+1

Step 5:   while ( i<=mid) do step 6

Step 6:    temp [k] =a[i]
           i++
          k++

Step 7:   while ( j<=high) do step 8

Step 8:    temp [k] =a[j]
           j++
           k++

Step 9:   for ( x= low  to high) do
```

```
            a[x] = temp [x]
```

- **Program:**

```
        #include<stdio.h>
        #include<conio.h>
        #define n 8

int A[n];
int Temp[n];

void Combine(int Low, int Mid, int High)
{
        int i,j,k,x;

        k=Low;
        i=Low;
        j=Mid+1;

        while( i <= Mid   &&   j <= High)
        {
                if(A[i]  <= A[j])
                {
                        Temp [k] = A[i];
                        i++;
                        k++;
                }
                else
                {
                        Temp[k] = A[j];
                        j++;
                        k++;
                }
        }

        while( i <= Mid )
        {
                Temp[k] = A[i];
                i++;
                k++;
        }

        while(j <= High)
        {
                Temp[k] = A[j];
                j++;
                k++;
        }

        for(x = Low; x <= High; x++)
        {
                A[x] = Temp[x];
        }
}

void Merge_Sort(int Low,int High)
{
      int Mid;

      if(Low < High)
      {
                Mid = (Low + High) / 2;

                Merge_Sort(Low, Mid);
                Merge_Sort(Mid + 1, High);
                Combine(Low, Mid, High);
      }
}

void main()
{
      int i;
      int Low, High;

      clrscr();

      for(i=0 ; i <= n-1; i++)
      {
                printf("\n Enter A[%d] : ", i);
                scanf("%d",&A[i]);
      }

      Low = 0;
      High = n-1;

      Merge_Sort(Low,High) ;

      printf("\n\n ARRAY AFTER SORTING : \n\n");

      for(i=0 ; i <= n-1; i++)
      {
                printf(" A[%d] = %d \n\n", i, A[i]);
      }

      getch();
}
```

Gtu Question

1. Explain Sequential search method.
2. Explain Binary search method
3. Write an algorithm for Selection sort method. Explain each step with example.
4. Write an algorithm for Insertion sort method. Explain each step with example.
5. Write a ‘C’ program for insertion sort and discuss its efficiency.
6. Apply quicksort algorithm to sort the following data. Justify the steps.

   42, 29, 74, 11, 65, 58