Bubble Sort-

If N elements are given in memory then for sorting we do following steps-

1. First compare the 1st and 2nd element of array if 1st < 2nd then compare the 2nd with 3 rd.
2. If 2nd > 3 rd

then interchange the value of 2nd and 3rd.

3. Now compare the value of 3rd (which has the value of 2nd) with 4th.
4. Similarly compare until the N-1th element is compared with Nth element.
5. Now the highest value element is reached at the Nth place.
6. Now elements will be compared until N-1 elements.

Ex-

Let us take the elements are-

13	32	20	62	68	52	38	46

Pass 1-

(1) Compare 1st and 2nd element, 13 < 32 No change

(2) Compare 2nd and 3 rd element, 32 > 20 Interchange

13	20	32	62	68	52	38	46

(3) Compare 3rd and 4th element, 32 < 62 No change
(4) Compare 4th and 5th element, 62 < 68 No change
(5) Compare 5th and 6th element, 68 > 52 Interchange

13 20 32 62 52 68 38 46

(6) Compare 6th and 7th element, 68 > 38 Interchange

13 20 32 62 52 38 68 46

(7) Compare 7th and 8th element, 68 > 46 Interchange

13 20 32 62 52 38 46 68

Pass 2-
Now we will show only interchange-

13 20 32 52 62 38 46 68
13 20 32 52 38 62 46 68
13 20 32 52 38 46 62 68

Pass 3-

13 20 32 38 52 46 62 68
13 20 32 38 46 52 62 68

Pass 4-

13 20 32 38 46 52 62 68

```
/* Program of sorting using bubble sort */
#include <stdio.h>
#define MAX 20
main( )
{
        int arr[MAX],i,j,k,temp,n,xchanges;
        printf("Enter the number of elements : ");
        scanf("%d",&n);
        for (i = 0; i < n; i++)
        {
                printf("Enter element %d : ",i+1);
                scanf("%d",&arr[i]);
        }
        printf("Unsorted list is :\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
```

```
        printf("\n");

        /* Bubble sort*/
        for (i = 0; i < n-1 ; i++)
        {
                xchanges=0;
                for (j = 0; j <n-1-i; j++)
                {
                        if (arr[j] > arr[j+1])
                        {
                                temp = arr[j];
                                arr[j] = arr[j+1];
                                arr[j+1] = temp;
                                xchanges++;
                        }/*End of if*/
                }/*End of inner for loop*/
                if(xchanges==0) /*If list is sorted*/
                        break;
                printf("After Pass %d elements are :  ",i+1);
                for (k = 0; k < n; k++)
                        printf("%d ", arr[k]);
                printf("\n");
        }/*End of outer for loop*/

        printf("Sorted list is :\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}/*End of main( )*/
```

Analysis-

In bubble sort, n-1 comparisons will be in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So it's very simple to calculate the number of comparisons. Total number of comparisons will be-

$(n-1) + (n-2) + (n-3) + \ldots\ldots\ldots\ldots\ldots + 3 + 2 + 1$

It's a form of arithmetic progression series. So we can apply formula-

$\text{Sum} = n/2 [2a + (n-1)d]$
$= (n-1)/2 [2 \times 1 (n-1-1) \times 1]$
$= (n-1)/2 [2 + n - 2]$
$= n(n-1)/2$

which is of $O(n^2)$.

The main advantage is the simplicity of algorithm, additional space requirement is only one temporary variable and it behaves as $O(n)$ for sorted array of element.

Selection Sort-

As the name suggests selection sort is the selection of an element and keeping it in sorted order. If you have a list of elements in unsorted order and you want to make a list of elements in sorted order then first you will take the smallest element and keep in the new list, after that second smallest element and so on until the largest element of list.

Let us take an array arr[0], arr[1].....arr[N-1] of elements. First you will search the position of smallest element from arr[0].....arr[N-1]. Then you will interchange that smallest element with arr[0]. Now you will search position of smallest element (second smallest element because arr[0] is the first smallest element) from arr[1].....arr[N-1], then interchange that smallest element with arr[1]. Similarly the process will be for arr[2].....arr[N-1]. The whole process will be as-

Pass 1 :
1. Search the smallest element from arr[0].....arr[N-1].
2. Interchange arr[0] with smallest element.
Result : arr[0] is sorted.

Pass 2 :
1. Search the smallest element from arr[1].....arr[N-1].
2. Interchange arr[1] with smallest element.
Result : arr[0], arr[1] is sorted.

.....
.....
.....

Pass N-1 :
1. Search the smallest element from arr[N-2] and arr[N-1].
2. Interchange arr[N-2] with smallest element.
Result : arr[0].....arr[N-1] is sorted.

Let us take list of elements in unsorted order and sort them by applying selection sort.

Pass								
1.	**75**	35	42	**13**	87	24	64	57
2.	13	**35**	42	75	87	**24**	64	57
3.	13	24	**42**	75	87	**35**	64	57
4.	13	24	35	**75**	87	**42**	64	57
5.	13	24	35	42	**87**	75	64	**57**
6.	13	24	35	42	57	**75**	**64**	87
7.	13	24	35	42	57	64	**75**	87

Now the elements are in sorted order-

13 24 35 42 57 64 75 87

```
/*Program of sorting using selection sort*/
#include <stdio.h>
#define MAX 20

main( )
{
        int arr[MAX], i,j,k,n,temp,smallest;
        printf("Enter the number of elements : ");
        scanf("%d",&n);
        for (i = 0; i < n; i++)
        {
                printf("Enter element %d : ",i+1);
                scanf("%d", &arr[i]);
        }
        printf("Unsorted list is : \n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
        /*Selection sort*/
        for(i = 0; i< n - 1 ; i++)
        {
                /*Find the smallest element*/
                smallest = i;
                for(k = i + 1; k < n ; k++)
                {
                        if(arr[smallest] > arr[k])
                                smallest = k ;
                }
                if( i != smallest )
                {
                        temp = arr [i];
                        arr[i] = arr[smallest];
                        arr[smallest] = temp ;
                }
                printf("After Pass %d elements are :  ",i+1);
                for (j = 0; j < n; j++)
                        printf("%d ", arr[j]);
                printf("\n");
        }/*End of for*/
        printf("Sorted list is : \n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}/*End of main( )*/
```

Analysis-

As we have seen selection sort algorithm will search the smallest element in the array and then that element will be at proper position. So in Pass 1 it will compare n-1 elements Same process will be for Pass 2 but this time comparison will be n-2 because first element is already at proper position. The elements, which are already at correct position, will not be disturbed. Same thing we will do for other passes. We can easily write function for comparisons as-

$F(n) = (n-1) + (n-2) + (n-3) + \ldots\ldots\ldots\ldots + 3 + 2 + 1$

This is arithmetic series in decreasing order, so applying the formula-

$Sum = n/2 [2a + (n-1) d]$

Where n = Number of elements in series, a = First element in series and
d = difference between second element and first element
or 2nd element – 1st element

Hence,

$F(n) = (n-1)/2 [2 (n-1) + \{ (n-1) - 1\} \{ (n-2) - (n-1) \}]$

$= (n-1)/2 [2n - 2 + (n-1-1) (n-2-n+1)]$
$= (n-1)/2 [2n - 2 + (n-2) (-1)]$
$= (n-1)/2 [2n - 2 - n + 2]$
$= n(n-1)/2$
$= O(n^2)$

Since, selection sort doesn't see the order of elements, so it's behaviour is near about same for worst and best case. The best thing with selection sort is that in every pass one element will be at correct position, very less temporary variables will be required for interchanging the elements and it is simple to implement.

Insertion Sort-

The insertion sort inserts each element in proper place. This is same as playing cards, in which we place the cards in proper order. There are n elements in the array and we place each element of array at proper place in the previously sorted element list.

Let us take there are N elements in the array arr. Then process of inserting each element in proper place is as-

Pass 1 – arr[0] is already sorted because of only one element.

Pass 2 – arr[1] is inserted before or after arr[0].
So arr[0] and arr[1] are sorted.
Pass 3 – arr[2] is inserted before arr[0], in between arr[0] and arr[1] or after arr[1].
So arr[0], arr[1] and arr[2] are sorted.
Pass4- arr[3] is inserted into its proper place in array arr[0], arr[1], arr[2]
So arr[0], arr[1], arr[2], arr[3] are sorted.
.....
.....
.....
Pass N-1 arr[N-1] is inserted into its proper place in array
arr[0], arr[1],.....arr[N-2]
So arr[0], arr[1].....arr[N-1] are sorted.

The element inserted in the proper place is compared with the previous elements and placed in between the ith element and i+1th element if

- element >= i^{th} element
- element <= $(i+1)^{th}$ element

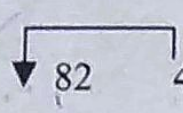
Ex-
Let us take the elements are-

82 42 49 8 92 25 59 52

Pass1-

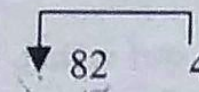
82 42 49 8 92 25 59 52

Pass 2-



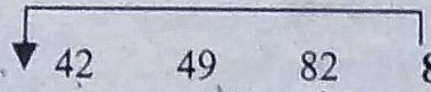
82 **42** 49 8 92 25 59 52

Pass 3-



42 82 **49** 8 92 25 59 52

Pass 4-

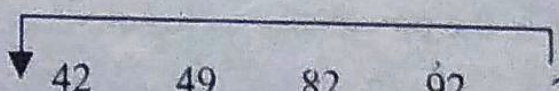


42 49 82 **8** 92 25 59 52

Pass 5-

8 42 49 82 **92** 25 59 52

Pass 6-

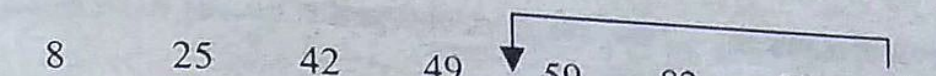


8 42 49 82 92 **25** 59 52

Pass 7-

8 25 42 49 82 92 **59** 52

Pass 8-



Sorted elements-

8 25 42 49 52 59 82 92

```
/* Program of sorting using insertion sort */
#include <stdio.h>
#define MAX 20

main( )
{
        int arr[MAX],i,j,k,n;
        printf("Enter the number of elements : ");
        scanf("%d",&n);
        for (i = 0; i < n; i++)
        {
                printf("Enter element %d : ",i+1);
                scanf("%d", &arr[i]);
        }
        printf("Unsorted list is :\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
        /*Insertion sort*/
        for(j=1;j<n;j++)
        {
                k=arr[j]; /*k is to be inserted at proper place*/
                for(i=j-1;i>=0 && k<arr[i];i--)
                        arr[i+1]=arr[i];
                arr[i+1]=k;
                printf("Pass %d, Element inserted in proper place: %d\n",j,k);
                for (i = 0; i < n; i++)
                        printf("%d ", arr[i]);
                printf("\n");
        }
        printf("Sorted list is :\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}/*End of main( )*/
```

Analysis-

In insertion sort we insert the element before or after and we start comparison from the first element. Since first element has no other elements before it, so it does not require any comparison. Second element requires 1 comparison, third requires 2 comparisons.

fourth requires 3 comparisons and so on. The last element requires n-1 comparisons. So the total number of comparisons will be-

$$1 + 2 + 3 + \ldots\ldots\ldots\ldots\ldots + (n-2) + (n-1)$$

It's a form of arithmetic progression series, so we can apply formula-

Sum = n/2 [2 x 1 + (n-1) x 1]
=(n-1)/2 [2 x 1 + (n-1-1) x 1]
= (n-1)/2 [2 + n – 2]
= n(n-1)/2
which is of $O(n^2)$.

$$\frac{(n-1)}{2}[n]$$
$$\frac{n^2}{2} - \frac{n}{2}$$

It's a worst case behaviour of insertion sort where all elements are in reverse order.
The advantage of insertion sort is it's simplicity and it is very efficient when number of elements to be sorted are very less. Because for smaller file size n the difference between $O(n^2)$ and $O(n \log n)$ is very less and $O(n \log n)$ has complex sorting technique.
Insertion sort behaves as of O(n) when elements are in sorted order and also has worth when list of elements are near about sorted.

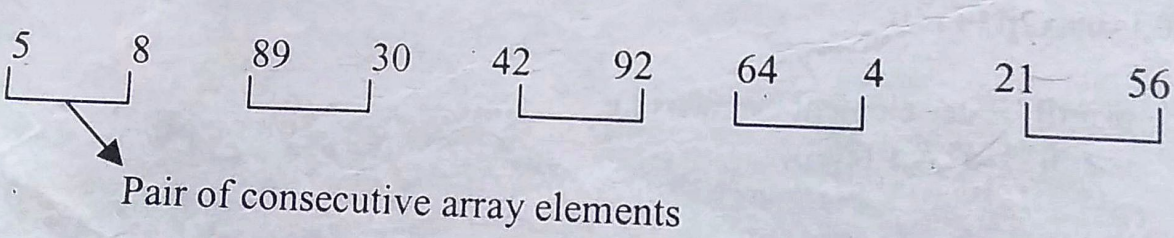
Merge Sort-

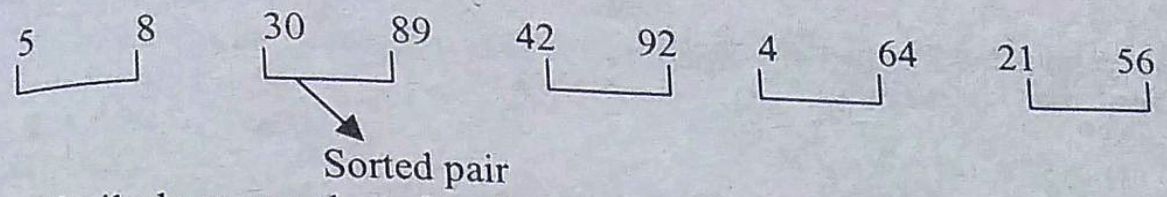
Similarly, in second approach of merging, we can take the pair of consecutive array elements, merge them in sorted array and then take adjacent pair of array elements and so on until the all elements of array are in single list.

.

Let us take an array of elements and process the merge sort as –

Pass 1- Merge the two sorted pairs in a sorted pair. Initially we have pair size 1. Merge these pair of size 1 into pairs of size 2.

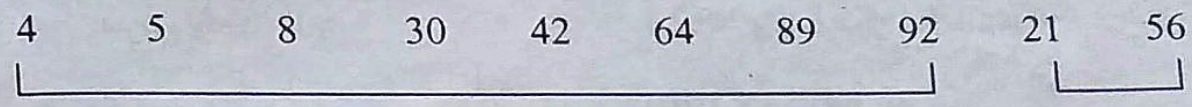




Pass 2-. Similarly merge the pairs of size 2 into pair of size 4.

Here 5 , 8 and 30 , 89 are in one pair and 21 , 56 is not merged because it is single pair and there is no other pair to merge.

Pass 3- Merge the two sorted pairs in a sorted pair.



Pass 4- Merge the two sorted pairs in a sorted pair.

4 5 8 21 30 42 56 64 89 92

Now the sorted list is-

4 5 8 21 30 42 56 64 89 92

```
/* Program of sorting using merge sort without recursion*/
#include<stdio.h>
#define MAX 30

main( )
{
        int arr[MAX],temp[MAX],i,j,k,n,size,l1,h1,l2,h2;

        printf("Enter the number of elements : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter element %d : ",i+1);
                scanf("%d",&arr[i]);
        }
        printf("Unsorted list is : ");
        for( i = 0 ; i<n ; i++)
                printf("%d ", arr[i]);
```

```
        /*ll lower bound of first pair and so on*/
        for(size=1; size < n; size=size*2 )
        {
                ll=0;
                k=0;  /*Index for temp array*/
                while( ll+size < n)
                {
                        h1=ll+size-1;
                        l2=h1+1;
                        h2=l2+size-1;
                        if( h2>=n ) /* h2 exceeds the limlt of arr */
                                h2=n-1;
                        /*Merge the two pairs with lower limits ll and l2*/
                        i=ll;
                        j=l2;
                        while(i<=h1 && j<=h2 )
                        {
                                if( arr[i] <= arr[j] )
                                        temp[k++]=arr[i++];
                                else
                                        temp[k++]=arr[j++];
                        }
                        while(i<=h1)
                                temp[k++]=arr[i++];
                        while(j<=h2)
                                temp[k++]=arr[j++];
                        /**Merging completed**/
                        ll=h2+1; /*Take the next two pairs for merging */
                }/*End of while*/

                for(i=ll; k<n; i++) /*any pair left */
                        temp[k++]=arr[i];

                for(i=0;i<n;i++)
                    arr[i]=temp[i];

                printf("\nSize=%d \nElements are : ",size);
                for( i = 0 ; i<n ; i++)
                        printf("%d ", arr[i]);
        }/*End of for loop */
        printf("Sorted list is :\n");
        for( i = 0 ; i<n ; i++)
                printf("%d ", arr[i]);
        printf("\n");
}/*End of main( )*/
```

We can implement merge sort with recursion also.

Quick Sort (Partition Exchange Sort) –

The idea behind this sorting is that sorting is much easier in two short lists rather than one long list. C. A. R. Hoare implements the divide and conquer means divide the big

problem into two small problems and then those two small problems into two small ones and so on. As example you have a list of 100 names and you want to list them alphabetically then you will make two lists for names A-L and M-Z from original list. Then you will divide list A-L into A-F and G-L and so on until the list could be easily sorted. Similar policy you will adopt for the list M-Z.

In Quick Sort we divide the original list into two sublists. We choose the item from list called key or pivot from which all the left side of elements are smaller and all the right side of elements are greater than that element. So we can create two lists, one list is on the left side of pivot and second list is on right side of pivot, means pivot will be in its actual position. Hence there are two conditions for choosing pivot-

1. All the elements on the left side of pivot should be smaller or equal to the pivot.
2. All the elements on the right side of pivot should be greater than or equal to pivot.

Similarly we choose the pivot for dividing the sublists until there are 2 or more elements in the sublist.

The process for sorting the elements through quick sort is as-
1. Take the first element of list as pivot.
2. Place pivot at the proper place in list. So one element of the list i.e. pivot will be at it's proper place.
3. Create two sublists left and right side of pivot.
4. Repeat the same process until all elements of list are at proper position in list.

For placing the pivot at proper place we have a need to do the following process-
1. Compare the pivot element one by one from right to left for getting the element which has value less than pivot element.
2. Interchange the element with pivot element.
3. Now the comparison will start from the interchanged element position from left to right for getting the element which has higher value than pivot.
4. Repeat the same process until pivot is at it's proper position.

Let us take a list of element and process through quick sorting-

48	29	8	59	72	88	42	65	95	19	82	**68**

Here we are taking 48 as pivot and we have to start comparison from right to left. Now the first element less than 48 is 19. So interchange it with pivot i.e. 48.

19	29	8	59	72	88	42	65	95	**48**	82	68

Now the comparison will start from 19 and will be from left to right. The first element greater than 48 is 59. So interchange it with pivot.

19 29 8 **48** 72 88 42 65 95 **59** 82 68

Now the comparison will start from 59 and will be from right to left. The first element less than 48 is 42. So interchange it with pivot.

19 29 8 **42** 72 88 **48** 65 95 59 82 68

Now the comparison will start from 42 and will be from left to right. The first element greater than 48 is 72. So interchange it with pivot.

19 29 8 42 **48** 88 **72** 65 95 59 88 68

Now the comparison will start from 72 and will be from right to left. There is no element less than 48. So now 48 is at it's proper position in the list. So we can divide the list into tow sublist, left and right side of pivot.

19 29 8 42 **48** 88 72 65 95 59 88 68

Sublist 1 Sublist 2

Now we have a need to do the same process for sublists and at the end all the elements of list will be at its proper position.

/*Program of sorting using quick sort through recursion*/

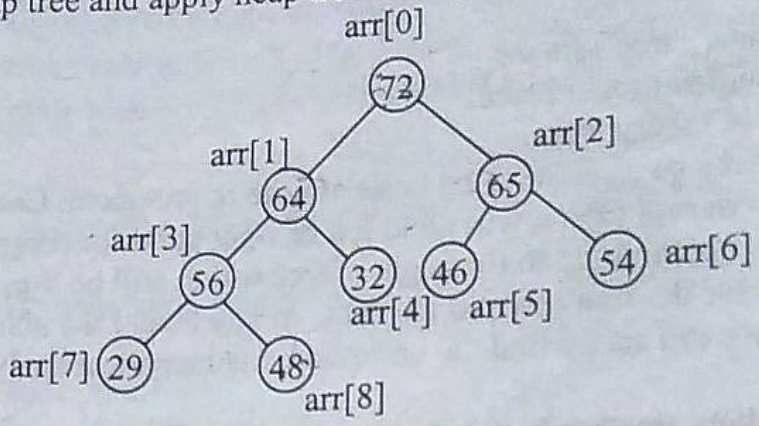
sorted order or in reverse order.

Heap Sort-

The elements of the heap tree are represented by an array. The root will be largest element of heap tree. Since it is maintained in the array, so the largest value should be the last element of array.
For heap sorting we will keep on deleting the root till there is only one element in the tree.
Then the array which represented the heap tree will now contain sorted elements.

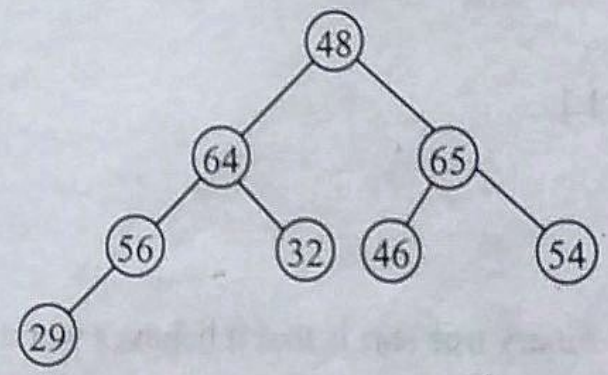
So we do following steps for heap sorting -
1. Replace the root with last node of heap tree.
2. Keep the last node (now root) at the proper position, means do the delete operation in heap tree but here deleted node is root

Let us take a heap tree and apply heap tree sorting-



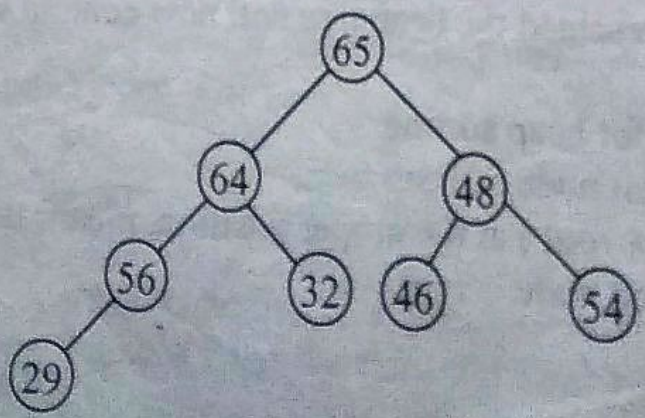
arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	72	64	65	56	32	46	54	29	48

Step 1-

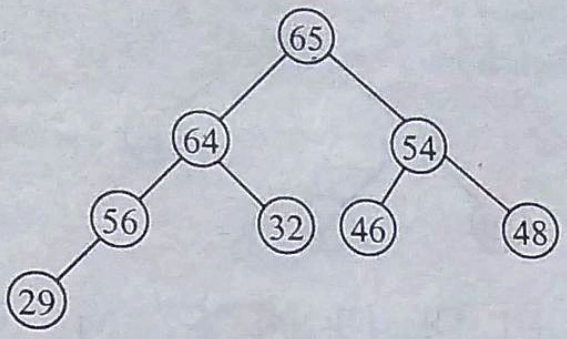


arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	48	64	65	56	32	46	54	29	72

Now the root is at the position of last node and last node at the position of root.
Here left and right child of 48 is 64 and 65. Both are greater than 48, but right child 65 is greater than left child 64, hence replace it with right child 65.



Here right child of 48 is 54, which is greater than 48, hence replace it with 54.

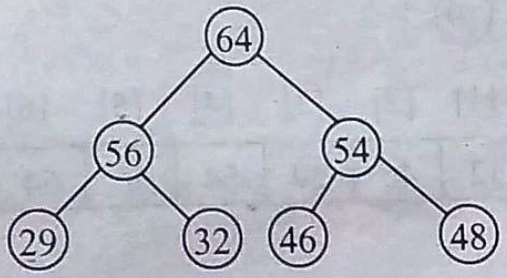


Now the elements of heap tree in array are as-

arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	65	64	54	56	32	46	48	29	72

Now 29 is the last node. So replace it with root 65 and do the same operation.

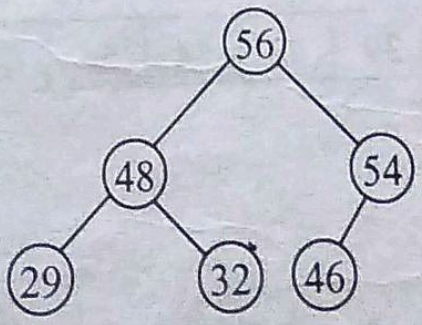
Step 2-



arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	64	56	54	29	32	46	48	65	72

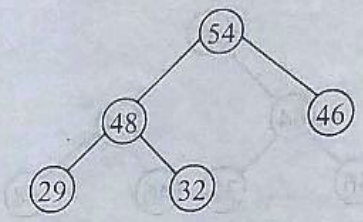
Similarly in next iteration

Step 3-



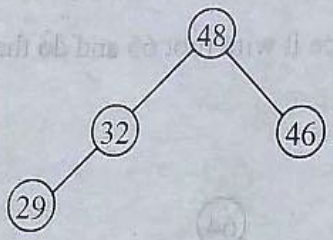
arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	56	48	54	29	32	46	64	65	72

Step 4-



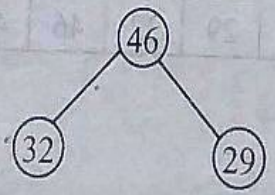
arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	54	48	46	29	32	56	64	65	72

Step 5-



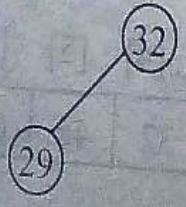
arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	48	32	46	29	54	56	64	65	72

Step 6-



arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	46	32	29	48	54	56	64	65	72

Step 7-



arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	32	29	46	48	54	56	64	65	72

Step 8

(29)

arr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	29	32	46	48	54	56	64	65	72

ow all the numbers are in sorted order.