**Sorting Techniques**

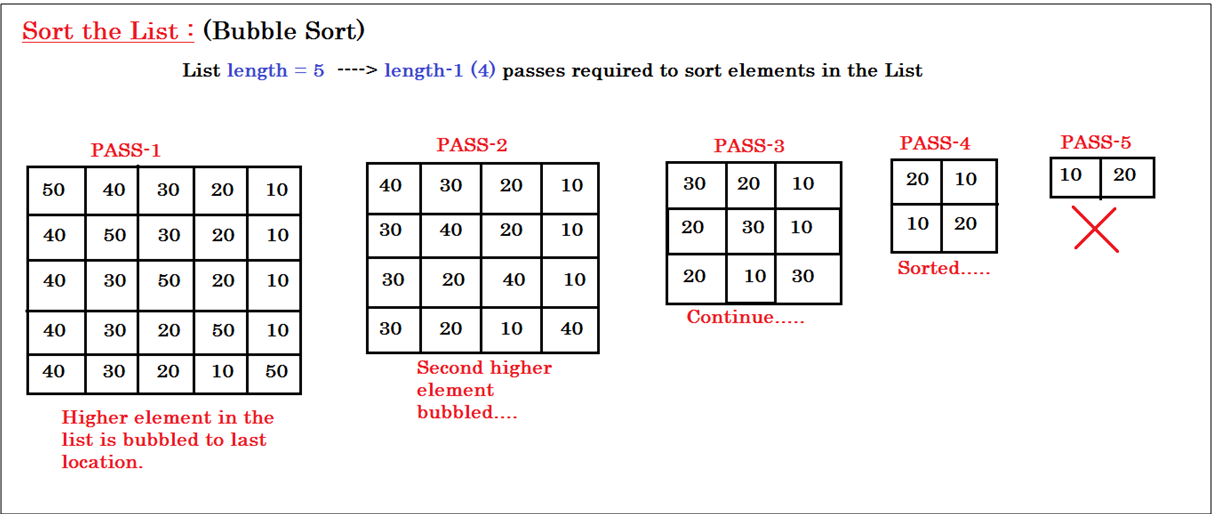
* Number of algorithms was proposed by different mathematicians used to arrange the elements of array in specified order.
* All algorithms mostly use numerical order.
* To optimize algorithms, efficiency of sorting is important.
* Following terms are familiar with before you go ahead and learn the sorting algorithms.

**Time Complexity:** It is the amount of time taken by a given code/algorithm to process or run as a function of the amount of input. In other words, time complexity is essentially efficiency, or how long a program function takes to process a given input.

**Space complexity:** Is the amount of memory used by the algorithm (including the input values to the algorithm) to execute and produce the result.

**Bubble sort**

* One of the easiest sorting algorithms to arrange the elements of array either in ascending or descending order.
* The concept of comparing index element with the next element and swapping them if required.
* For each pass, the highest element in the array is bubbled to the last location.
* This process will continue until all the elements get sorted.



#include<stdio.h>

#include<conio.h>

void bubble\_sort(int[] , int);

int main()

{

int arr[50] , n , i ;

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%100 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

bubble\_sort(arr , n) ;

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

return 0;

}

void bubble\_sort(int a[ ], int n)

{

int i, j, temp;

for (i=0 ; i<n-1 ; ++i)

{

for(j=0 ; j<n-1-i ; ++j)

{

if (a[j]>a[j+1])

{

temp = a[j+1];

a[j+1] = a[j];

a[j] = temp;

}

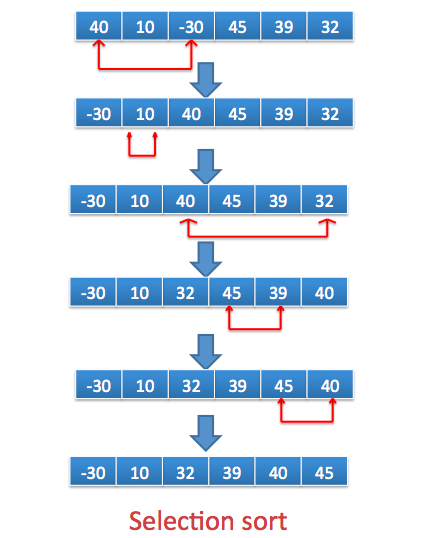
}

}

}

**Selection sort**

* Another sorting technique to performing sort operations on given array elements.
* In Selection sort, Index element always replace with the least element in the list.
* Finding the least element in the list using Iterators (loop) and replacing with index element is the process of Selection Sort.



#include<stdio.h>

#include<conio.h>

void selection\_sort(int[] , int);

int main()

{

int arr[50] , n , i ;

clrscr();

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%32767 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

selection\_sort(arr , n) ;

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

return 0;

}

void selection\_sort(int sort[ ], int n)

{

int i;

for (i=0 ; i<n-1; i++)

{

int j, min, temp;

min = i;

for (j=i+1; j<n ; j++)

{

if (sort[j] < sort[min])

{

min = j;

}

}

temp = sort[i];

sort[i] = sort[min];

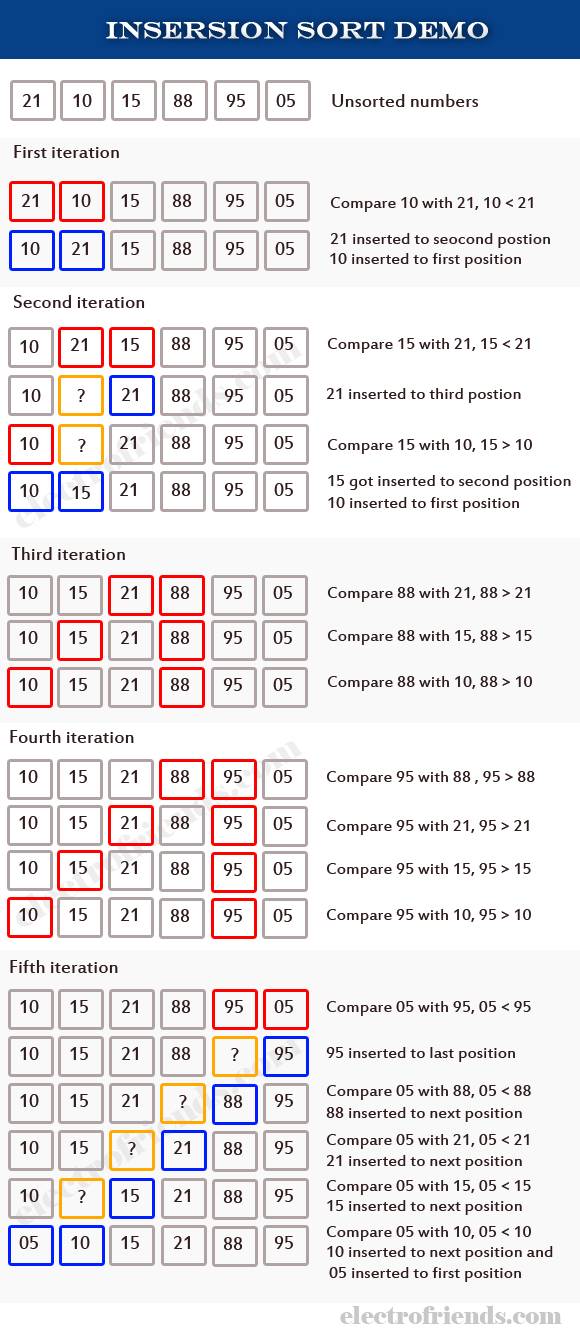
sort[min] = temp;

}

}

**Insertion sort**

* The concept of inserting suitable element into its position by shifting the index element from that position.
* In the first iteration, second element compare with first element.
* In the second iteration, third element compare with first two elements.
* In the nth iteration, first n-1 elements will be comparing for arrangement.
* The following diagram shows the process of elements re-arrangement.



#include<stdio.h>

#include<conio.h>

void insertion\_sort(int[] , int);

int main()

{

int arr[50] , n , i ;

clrscr();

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%32767 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

insertion\_sort(arr , n) ;

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

return 0;

}

void insertion\_sort(int sort[ ], int n)

{

int i , j, temp ;

for(i=1 ; i<n ; i++)

{

temp = sort[i];

j=i-1;

while(temp<sort[j] && j>=0)

{

sort[j+1] = sort[j];

j = j-1;

}

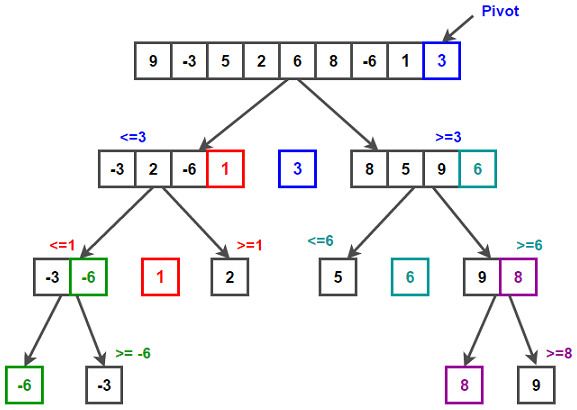
sort[j+1] = temp;

}

}

**Quick Sort**

* The idea of quick sort is dividing the given list into smaller lists then sorting those lists.
* Quick sort generally done through recursion easily.
* Quick sort is a divide-and-conquer algorithm like merge sort.
* The first step of the algorithm requires choosing a "pivot" value that will be used to divide least and highest numbers.
* Each implementation of quick sort has its own method of choosing the pivot value.
* Once the pivot value is ready, all values are smaller than the pivot will be placed towards the beginning of array and all the higher values move to right.
* This process essentially sets the pivot value in the correct place each time.
* Each side of the pivot is then sorted called Quick Sort.



#include<stdio.h>

#include<conio.h>

void quick\_sort(int[] , int, int);

int main()

{

int arr[50] , n , i ;

clrscr();

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%32767 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

quick\_sort(arr, 0, n-1) ;

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t", arr[i]);

}

printf("\n\n");

return 0;

}

void quick\_sort(int data[], int left, int right)

{

int mid, tmp, i, j ;

i = left;

j = right;

mid = data[(left+right)/2];

do

{

while(data[i]<mid)

i++;

while(mid<data[j])

j--;

if (i <= j)

{

tmp = data[i];

data[i] = data[j];

data[j] = tmp;

i++;

j--;

}

}while(i<=j);

if (left < j)

quick\_sort(data, left, j);

if (i < right)

quick\_sort(data, i, right);

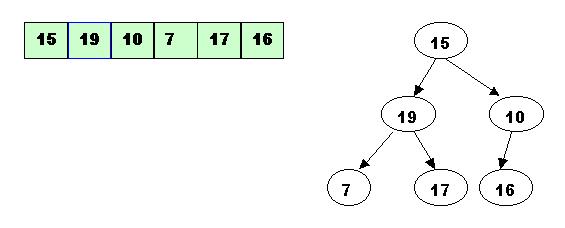
}

**Heap Sort**

* Heap sort algorithm is a comparison algorithm.
* Heap sort begins with construction of Heap tree with specified elements and then removing the largest item and placing it at the end of the sorted array.
* After removing the largest item we need to reconstruct the heap tree by following rules of tree.
* This process continuous until all elements gets arranged.
* Heap sort uses two heap operations insertion and root deletion.

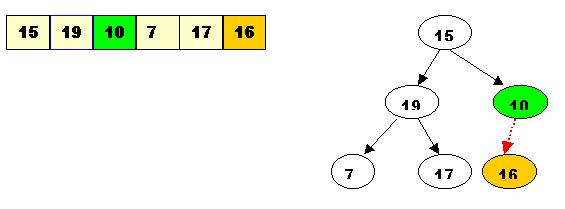
**Building the heap tree:**

* We need to construct the heap tree with given elements.
* Heap tree rule is “Parent node should contain greater value than both of its Children”.
* The tree representation is used to understand the algorithm more easily while sorting the element of this array.

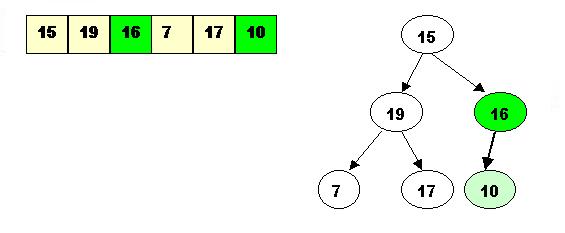


**Note:** We need to start check and compare the element from the rightmost node at height 1 - the node at position 3 = Size/2.

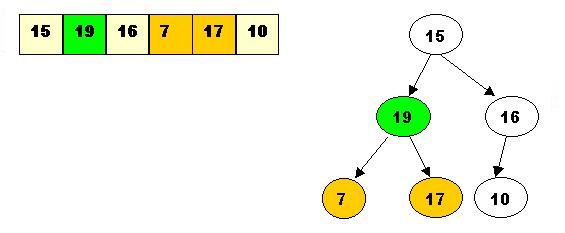
* It has one greater child and has to be percolated down:



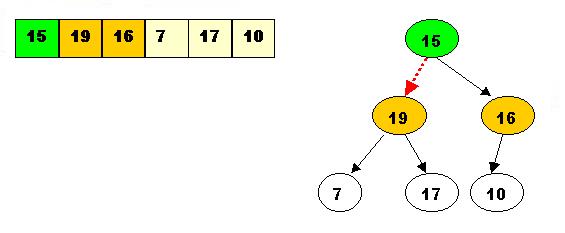
* After processing array[3] the situation is:



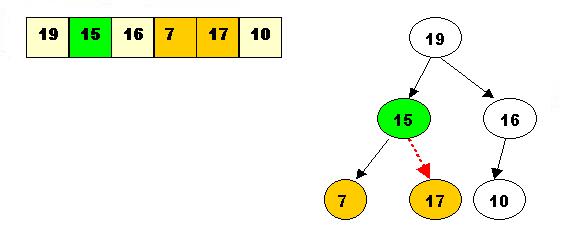
* Next comes array[2]. Its children are smaller, so no percolation is needed.



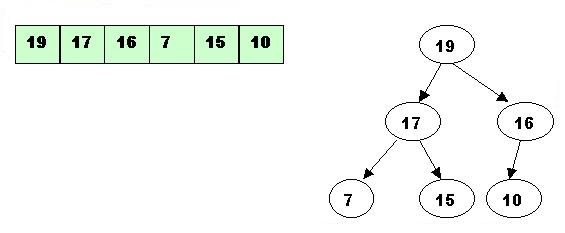
* The last node to be processed is array[1]. Its left child is the greater of the children.
* The item at array[1] has to be percolated down to the left, swapped with array[2].



As a result the situation is:

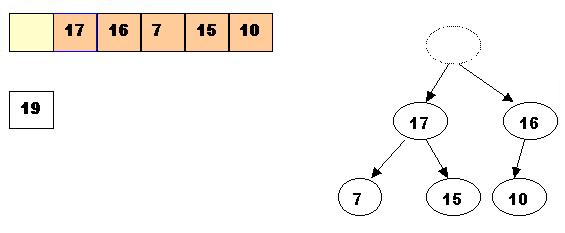


The children of array[2] are greater, and item 15 has to be moved down further, swapped with array[5].

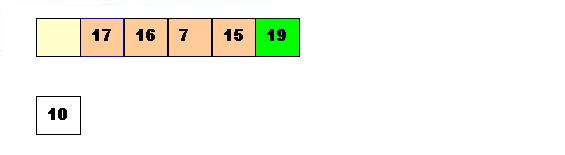


Performing deleteMax operations:

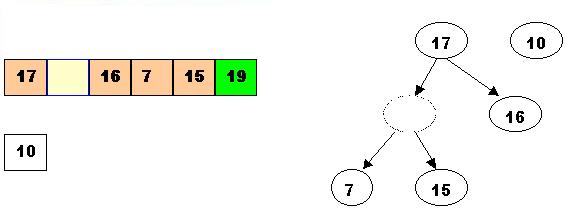
Delete the top element 19. Store 19 in a temporary place. A hole is created at the top



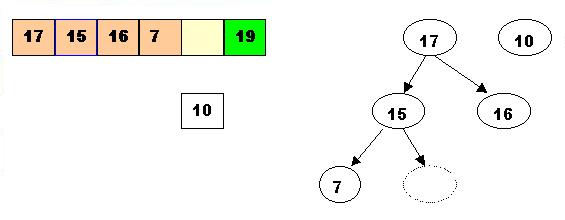
Swap 19 with the last element of the heap. As 10 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array



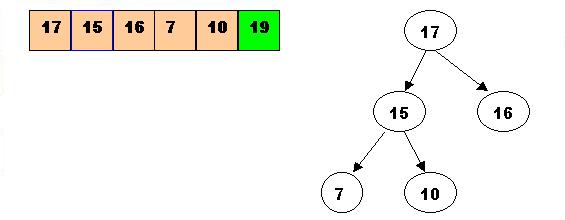
Percolate down the hole



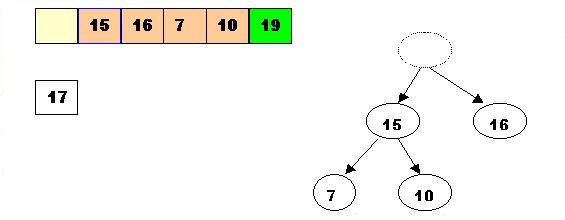
Percolate once more (10 is less that 15, so it cannot be inserted in the previous hole)



Now 10 can be inserted in the hole

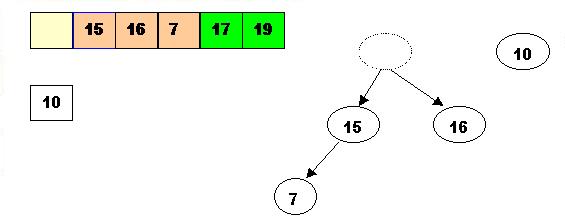


DeleteMax the top element 17. Store 17 in a temporary place. A hole is created at the top

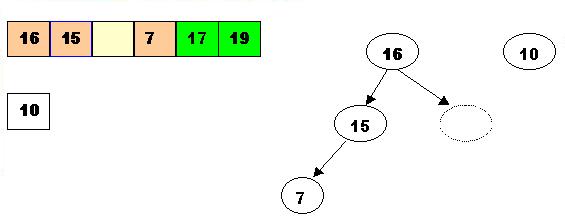


Swap 17 with the last element of the heap.

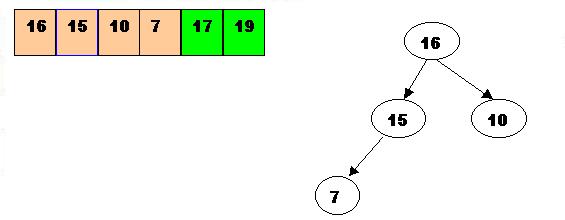
As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.  
Instead it becomes a cell from the sorted array



2.3. The element 10 is less than the children of the hole, and we percolate the hole down:

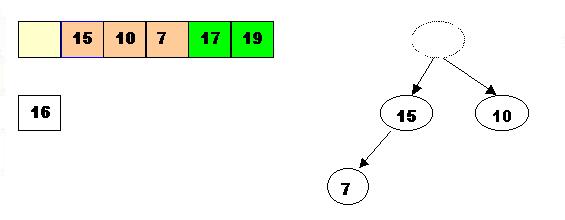


Insert 10 in the hole



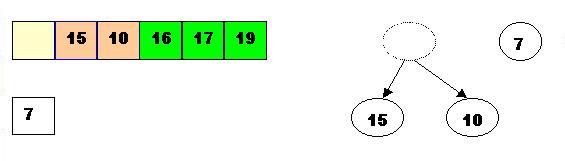
DeleteMax 16

Store 16 in a temporary place. A hole is created at the top

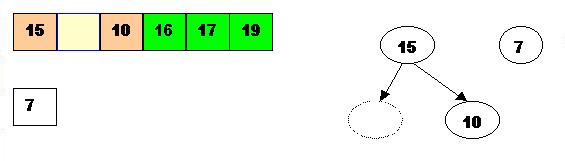


Swap 16 with the last element of the heap.

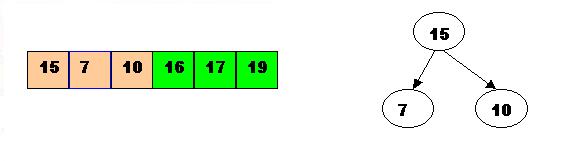
As 7 will be adjusted in the heap, its cell will no longer be a part of the heap.  
Instead it becomes a cell from the sorted array



Percolate the hole down (7 cannot be inserted there - it is less than the children of the hole)

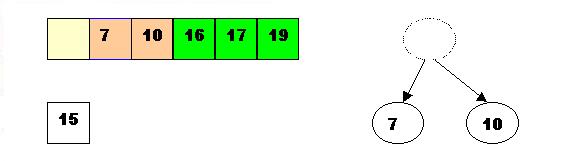


Insert 7 in the hole



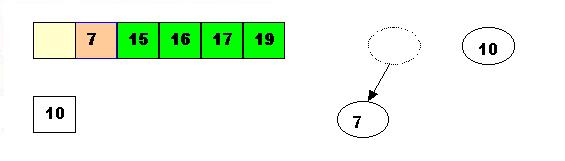
DeleteMax the top element 15

Store 15 in a temporary location. A hole is created.

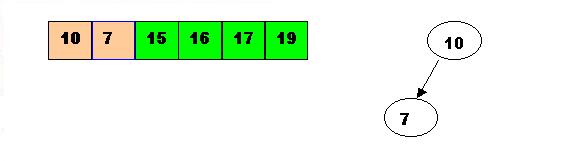


Swap 15 with the last element of the heap.

As 10 will be adjusted in the heap, its cell will no longer be a part of the heap.   
Instead it becomes a position from the sorted array

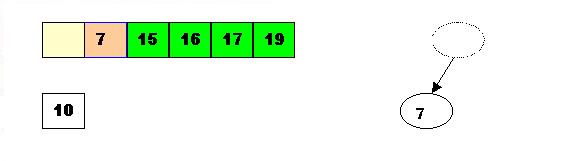


Store 10 in the hole (10 is greater than the children of the hole)



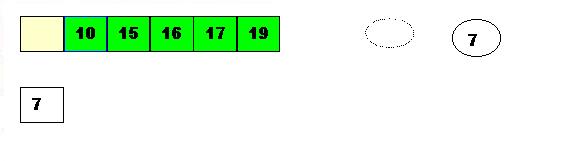
DeleteMax the top element 10.

Remove 10 from the heap and store it into a temporary location.



Swap 10 with the last element of the heap.

As 7 will be adjusted in the heap, its cell will no longer be a part of the heap. Instead it becomes a cell from the sorted array



Store 7 in the hole (as the only remaining element in the heap



7 is the last element from the heap, so now the array is sorted



**Code program of Heapsort:**

#include<stdio.h>

void constructHeap(int[] , int);

void deleteMaxAndReConstruct(int[] , int , int);

int main()

{

int list[50], n, i, j ;

printf("Enter number of elements : ");

scanf("%d", &n);

for(i=1 ; i<=n ; i++)

{

list[i] = rand()%32767 ;

}

printf("list of elements before sort : \n");

for( i=1 ; i<=n ; i++ )

{

printf("%d\t", list[i]);

}

printf("\n\n");

for(i=1 ; i<=n ; i++)

{

constructHeap(list, i);

}

j=n;

for(i=1 ; i<=j ; i++)

{

int temp;

temp=list[1];

list[1]=list[n];

list[n]=temp;

n--;

deleteMaxAndReConstruct(list,1,n);

}

n=j;

printf("list of elements after sort : \n");

for( i=1 ; i<=n ; i++ )

{

printf("%d\t",list[i]);

}

printf("\n\n");

return 0;

}

void constructHeap(int a[] , int i)

{

int v=a[i];

while((i>1)&&(a[i/2]<v))

{

a[i]=a[i/2];

i=i/2;

}

a[i]=v;

}

void deleteMaxAndReConstruct(int a[], int i, int n)

{

int v=a[i];

int j=i\*2;

while(j<=n)

{

if((j<n)&&(a[j]<a[j+1]))

j++;

if(a[j]<a[j/2])

break;

a[j/2]=a[j];

j=j\*2;

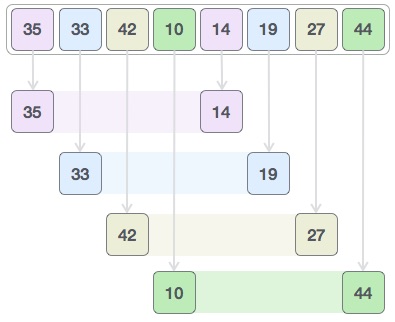
}

a[j/2]=v;

}

**Shell Sort**

* It is called Generalized sort of Insertion sort.
* The elements will be sorted in a sequence and the size of sequence decreases once element get sorted.
* Shell sort original sequence is N/2 , N/4…., 1
* Suppose we need to sort 8 elements then, in the first loop, the array elements at N and N/2 will be tested and sorted.
* The 0th element is compared with the 4th element.
* If the 0th element is greater than the 4th one then, the 4th element is first stored in temp variable and the 0th element (ie. greater element) is stored in the 4th position and the element stored in temp is stored in the 0th position.
* In the second pass, N/4 = 8/4 = 2 is taken and again the elements lying at these intervals are sorted.
* This process continues until all elements get sorted in the specified array.



#include<stdio.h>

#include<conio.h>

void shell\_sort(int[] , int);

int main()

{

int arr[50] , n , i ;

clrscr();

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%32767 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

shell\_sort(arr, n) ;

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t", arr[i]);

}

printf("\n\n");

return 0;

}

void shell\_sort(int arr[] , int n)

{

int i, j, k, temp ;

for(i=n/2; i>0; i=i/2)

{

for(j=i; j<n; j++)

{

for(k=j-i; k>=0; k=k-i)

{

if(arr[k+i]>=arr[k])

break;

else

{

temp=arr[k];

arr[k]=arr[k+i];

arr[k+i]=temp;

}

}

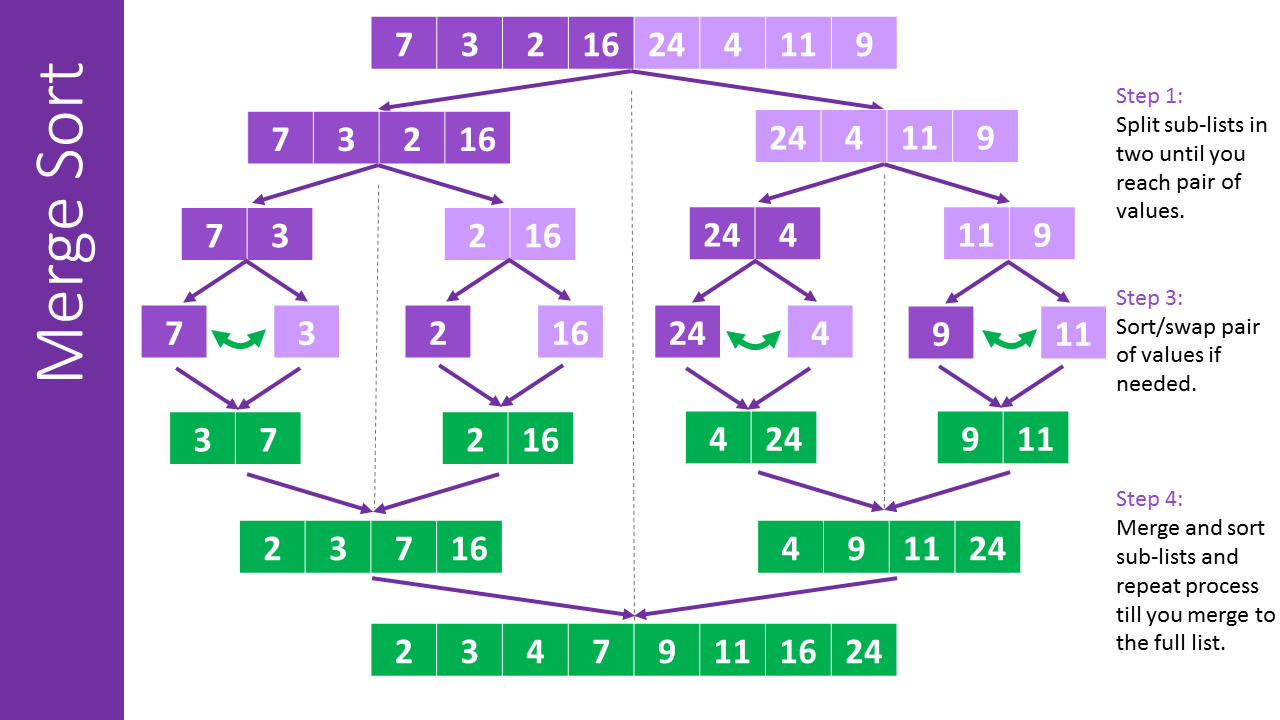
}

}

}

**Merge Sort**

* It is Divide and Conquer algorithm.
* Initially it divides the list into sub lists using recursion.
* During the Sorting process, the list is divided into two lists.
* To split a list, Merge sort will take the middle of the list and split into its left and its right part.
* The resultant arrays are again recursively splits via the Merge sort algorithm until they are broke to single element in each collection.
* It starts sorting elements in each sub list and start merging them.
* Merging also will be done with recursion with sort.



**The code implementation as follows:**

#include<stdio.h>

#include<conio.h>

void array\_partition(int[], int, int);

void merge\_sort(int[],int,int,int);

int main()

{

int arr[50] , n , i ;

clrscr();

printf("Enter number of elements : ");

scanf("%d", &n);

for( i=0 ; i<n ; i++ )

{

arr[i] = rand()%32767 ;

}

printf("Array elements before sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t",arr[i]);

}

printf("\n\n");

array\_partition(arr,0,n-1);

printf("Array elements after sort : \n");

for( i=0 ; i<n ; i++ )

{

printf("%d\t", arr[i]);

}

printf("\n\n");

return 0;

}

void array\_partition(int arr[], int low, int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

array\_partition(arr,low,mid);

array\_partition(arr,mid+1,high);

merge\_sort(arr,low,mid,high);

}

}

void merge\_sort(int arr[],int low,int mid,int high)

{

int i,j,k,l,b[20];

l=low;

i=low;

j=mid+1;

while((l<=mid)&&(j<=high))

{

if(arr[l]<=arr[j])

{

b[i]=arr[l];

l++;

}

else

{

b[i]=arr[j];

j++;

}

i++;

}

if(l>mid)

{

for(k=j;k<=high;k++)

{

b[i]=arr[k];

i++;

}

}

else

{

for(k=l;k<=mid;k++)

{

b[i]=arr[k];

i++;

}

}

for(k=low;k<=high;k++)

{

arr[k]=b[k];

}

}