UNIT-6

Sorting

* A sorting is a technique which is mainly used to arrange the given elements either in ascending (or) descending order.

* We have different type of sorting techniques such as bubble sort, heap sort, insertion sort, quick sort, merge sort, radic sort, selection sort etc....

4. **Insertion Sort:-** This sorting technique selects one element for each position from the given list of elements, and the selected element will be inserted into its proper position in the list.

Time complexity:-

Best case is order of n - $O(n)$

Worst case is order of n^2 . $O(n^2)$

Procedure:-

Eg:- 10, 5, 12, 6, 20, 7

consider the above elements are in the unsorted portion of the list and the sorted portion is empty.

empty

0	1	2	3	4	5
10	5	12	6	20	7

Sorted (S)

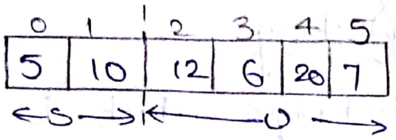
unsorted (U)

Pass(1):- move the first element 10 to the sorted portion of the list.

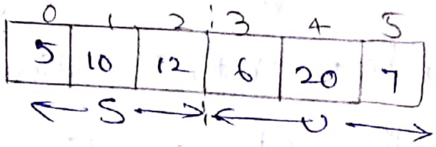
0	1	2	3	4	5
10	5	12	6	20	7

←S→|← U →

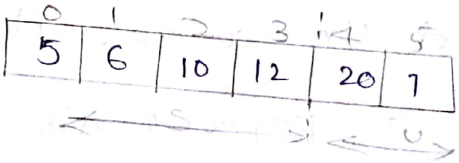
pass(2):- move the second element 5 from unsorted portion to sorted portion by comparin the existed elements in the sorted portion.



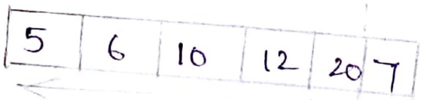
pass(3):- move the third element 12 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



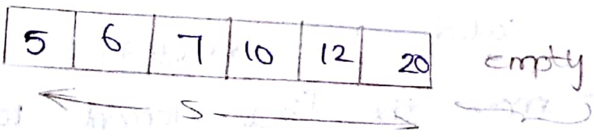
pass(4):- move the fourth element 6 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion.



pass (5):- move the fifth element 20 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



pass (6):- move the sixth element 7 from unsorted portion to sorted portion by comparing the existed elements in the sorted portion



```
# include <iostream.h>
# include <conio.h>
void main( )
{
int a[50],n,i,j,temp;
clrscr( );
 cout << "enter the no.of elements:";
 cin >>n;
 cout<<"\n enter the elements:";
 for (i=0; i<n; i++)
 {
  cin >> a[i];
  }
  for (i=1; i<n ;i++)
  {
   temp = a[i];
   j= i-1;
  while((j>=0) && (a[j]>temp))
   {
     a[j+1] =a[j];
     j--
   }
     a[j+1] = temp;
  }
 cout <<"\n the elements after insertion sort are:";
 for (i=0; i<n; i++)
 {
 cout<<a[i]<<" ";
 }
 getch();
}
```

Heap Sort:-

* A heap sort is one of the best sorting technique to organize the elements either in ascending (or) descendi order.

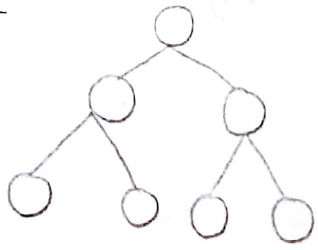
*Usually a heap is a tree based data structure. which organises the elements in the form of tree mann

* The heap follows two basic properties they are
  1. shape property
  2. heap property

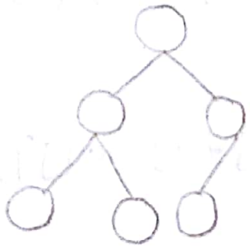
1. Shape Property:- This property states that the heap shoul be either complete binary tree (or) almost complete binary tree.

Complete binary Tree:- In a binary tree, if all the nodes are fully filled in each level except the last level then such binary tree can be called as complete binary tree.

Ex:-



Complete binary Tree



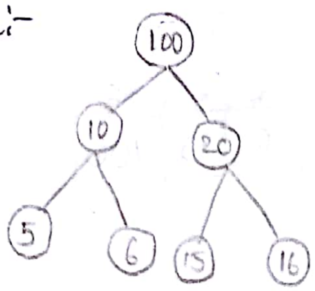
almost complete binary Tree

2. heap property:- In the complete binary tree are almost complete binary tree, if all the parent nodes should be greater than ~~(or)~~ equal to their child nodes (or) all the parent nodes less than ~~(or)~~ equal to their child nodes

* If all the parent nodes are greater than ~~equal to~~ (or) their child nodes then such heap can be called as Max-heap

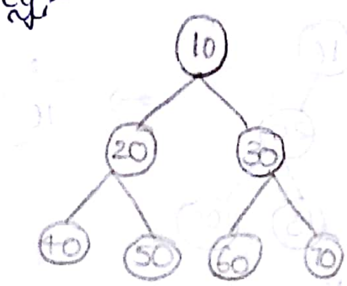
* If all the parent nodes are less than ~~equal to~~ (or) their child nodes then such heap can be called as Min-heap.

Eg:-



Max-heap

Eg:-



Min-heap

Binary Tree:- In a tree, each node contains 0 (or) 1 (or) 2 child nodes then such tree can be called binary tree.

Eg:-

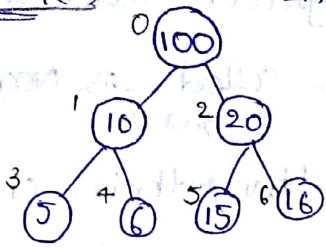
(a) (b) (c)

* The heap sort, first creates the heap either Max-heap (or) Min-heap with the given elements then it swaps the root element with the specific position element in the heap (or) array. and after the heap will be reconstructed either max-heap (or) min-heap depending upon the sorting order this process will be continued until all the elements are sorted

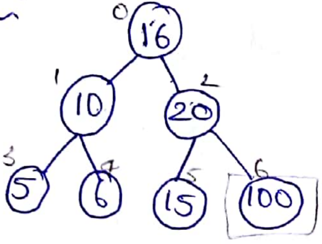
Eg:- 100, 10, 20, 5, 6, 15, 16

0	1	2	3	4	5	6
100	10	20	5	6	15	16

~~Step-1(1)~~ Initial elements



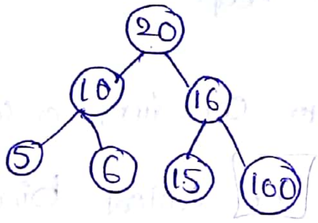
Step (1):-



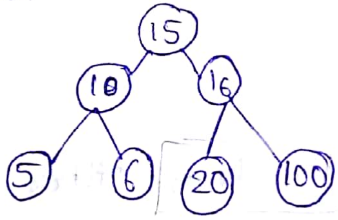
⇒

0	1	2	3	4	5	6
16	10	20	5	6	15	100

⇓



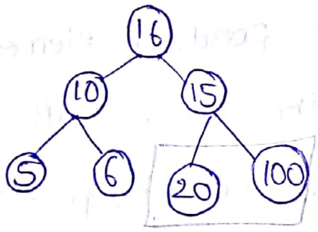
Step (2):-



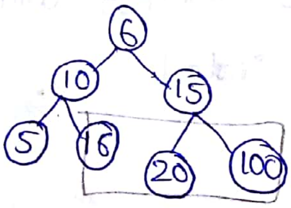
⇒

0	1	2	3	4	5	6
15	10	16	5	6	20	100

⇓



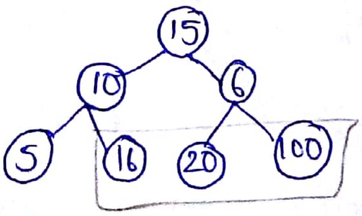
Step (3):-



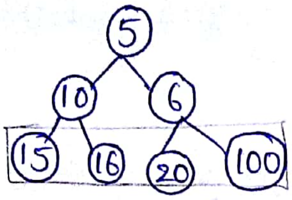
⇒

0	1	2	3	4	5	6
6	10	15	5	16	20	100

⇓



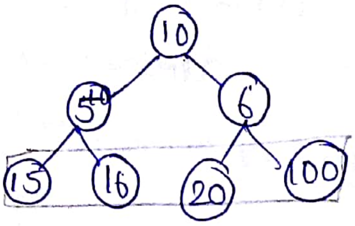
Step (4):-



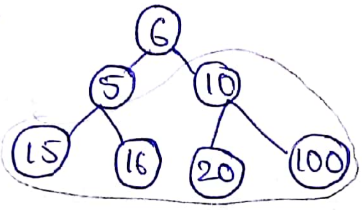
⇒

0	1	2	3	4	5	6
10	5	6	15	16	20	100

⇓



Step (5):-

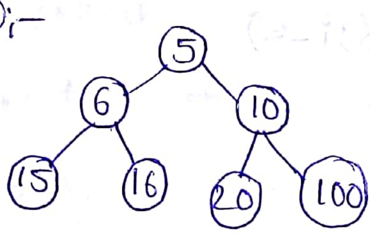


⇒

0	1	2	3	4	5	6
6	5	10	15	16	20	100

⇓

Step (6):-



⇒

0	1	2	3	4	5	6
5	6	10	15	16	20	100

Program to implement heap sort:-

```
#include <iostream.h>
#include <conio.h>
 void heapsort( );
 void heapadjust (int, int);
 int a[50], n, i, temp;
 void main( )
  {
   clrscr( );
   cout <<" Enter the no.of elements: ";
   cin >>n;
   cout<< "\n Enter the elements:";
   for (i=0; i<n;i++)
   {
    cin>>a[i];
```

```
}
heapSort( );
cout<<"\n the elements after heapsort are:";
for (i=0; i<n; i++)
{
cout <<a[i] <<" ";
}
getch( );
}
void heapsort( )
{
for (i=(n/2)-1; i>=0; i--)
{
 heapadjust (n, i);
 }
 for (i=n-1; i>=0; i--)
 {
  temp = a[i];
  a[i] = a[0];
  a[0] =temp;
  heapadjust (i,0);
  }
 }
void heapadjust (int n, int i)
{
int large=i, left=2*i+1, right=2*i+2;
if((left<n) && (a[left] >a [large]))
 large = right;
```

oot element swapping

ascending

descending

```
if(large != i)
{
 temp = a[large];
 a[large] = a[i];
 a[i] = temp;
 heapadjust (n, large);
 }
}
```

* quick Sort :-

* The quick sort is one of efficient sorting technique which uses divide and conquerer approach in order to sort the given list of elements either in ascending (or) Descending order.

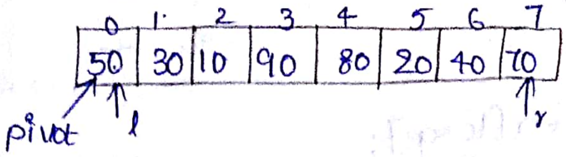
* In this technique the first element from the list of elements is selected as pivot element

* When the pivot element is placed at a respective position in the list then the elements which are lessthan pivot elements will be taken into left sub-Array and the elements which are greater than the pivot element will be taken into Right sub-array this procedure will be contined untill all the elements are placed at their respective positions.

Time complexity :-

```
1. while((pivot >= a[l]) && (l <= r)) => T
        l++;
2. while((pivot < a[r]) && (l <= r)) => T
        r--;
3. if (l<r) T => swap a[l] & a[r]
            F => Swap pivot & a[r]
```

Ex:- 50, 30, 10, 90, 80, 20, 40, 70



0	1	2	3	4	5	6	7
50	30	10	90	80	20	40	70

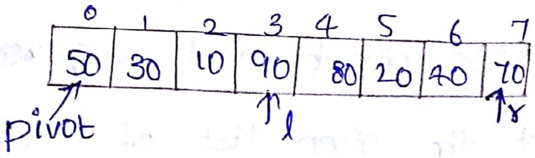
pivot >= a[l] && l <= r

50 >= 50 && 0 <= 7 } l++
T T l=1

50 >= 30 && 1 <= 7 } l++
T T l=2

50 >= 10 && 2 <= 7 } l++
T T l=3

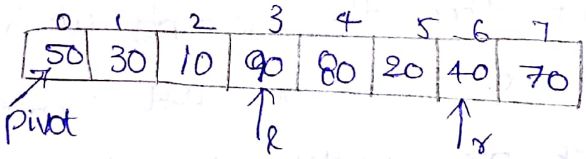
50 >= 90 && 3 <= 7 } stop incrementation l
F T



0	1	2	3	4	5	6	7
50	30	10	90	80	20	40	70

pivot < a[r] && l <= r } r--
r=6
50 < 70 && 3 <= 7
T T

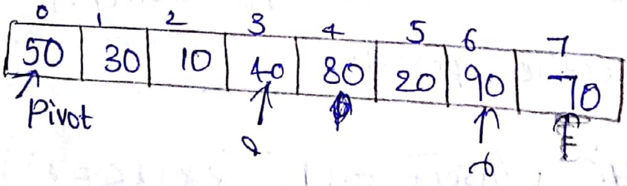
50 < 40 && 3 <= 6 } stop decrementing r
F T



0	1	2	3	4	5	6	7
50	30	10	90	80	20	40	70

(l < r)

3 < 6 ⇒ T, swap a[l] & a[r]



0	1	2	3	4	5	6	7
50	30	10	40	80	20	90	70

50 >= 40 && 3 <= 6 } l++
T T l=4

50 >= 80 && 4 <= 6 } stop incrementing l
F T

0	1	2	3	4	5	6	7
50	30	10	40	80	20	90	70
↗p				↑l		↑	

$\underset{T}{50<90}$ && $\underset{T}{4<=6}$ } r--
r=5

$\underset{F}{50<20}$ && $\underset{T}{4<=5}$ } F Stop decrementing r

0	1	2	3	4	5	6	7
50	30	10	40	80	20	90	70
↗p.				↑l	↑r		

l<r

4<5 ⇒ T, Swap $a[\overset{4}{l}]$ & $a[\overset{5}{r}]$

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
↗p				↑l	↑r		

$\underset{T}{50>=20}$ && $\underset{T}{4<=5}$ } l++
l=5

$\underset{F}{50>=80}$ && $\underset{T}{5<=5}$ } Stop incrementing l

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
p↗					↑↑ l r		

$\underset{T}{50<80}$ && $\underset{T}{5<=5}$ } r--
r=4

$\underset{F}{50<20}$ && $\underset{F}{5<=4}$ } Stop decrementing r

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
↗p				↑r	↑l		

(l<r)

5<4 ⇒ F Swap pivot & a[r]

0	1	2	3	4	5	6	7
20	30	10	40	50	80	90	70

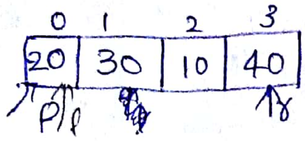
0	1	2	3
20	30	10	40

4
50

5	6	7
80	90	70

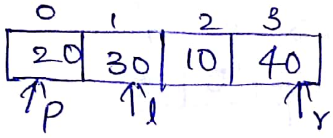
right sub-array

left sub-array



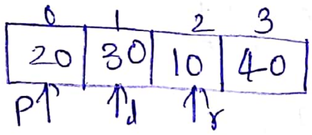
20>=20 && 0<=3 } l++
T T l=1

20>=30 && 1<=3 } F ⇒ stop incrementing l
F T



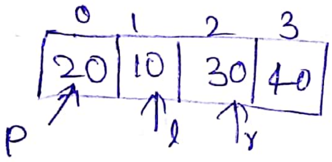
20<=40 && 1<=3 } T ⇒ r--
T T r=2

20<10 && 1<=2 } F ⇒ stop decrementing r
F T



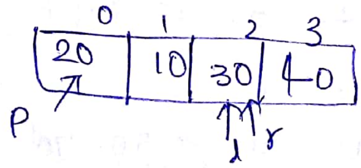
l<r 1<2 ⇒ T ⇒ swap a[l] & a[r]

a[1] & a[2]



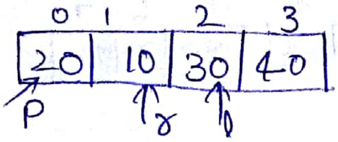
20>=10 && 1<=2 } T ⇒ l++
T T l=2

20>=30 && 2<=2 } F ⇒ Stop incrementing l
F T



20<30 && 2<=2 } T ⇒ r--
T T r=1

20<10 && 2<=1 } F ⇒ Stop decrementing r
F F

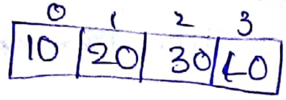


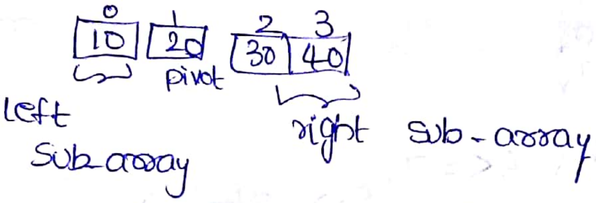
$l<r$

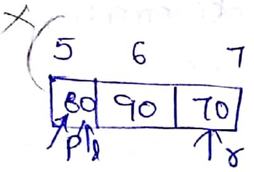
$2<1 \Rightarrow F \Rightarrow$ Swap

pivot & a[r]

20 & a[10]

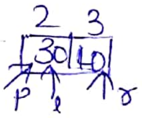






$\underset{T}{(80>=80)}\ \&\&\ \underset{T}{5<=7}\Big\}\ T \Rightarrow \underset{6}{l++}$

$\underset{F}{80>=90}\ \&\&\ \underset{T}{6<=7}\Big\}\ F \Rightarrow$ Stop incrementing l)



$\underset{T}{30>=30}\ \&\&\ \underset{T}{2<=3}\Big\}\ T \Rightarrow l++$

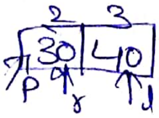
$l=3$

$\underset{F}{30>=40}\ \&\&\ \underset{T}{3<=3}\Big\}\ F \Rightarrow$ Stop incrementing l

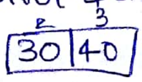
$\underset{T}{30<40}\ \&\&\ \underset{T}{3<=3}\Big\}\ T \Rightarrow r--$

$r=2$

$\underset{F}{30<30}\ \&\&\ \underset{F}{3<=2}\Big\}\ F \Rightarrow$ stop decrementing r



$l<r \Rightarrow 3<2 \Rightarrow F$ Swap $\Rightarrow$ pivot & a[r]



80 & 70

5	6	7
70	80	90

5	6	7
70	80	90

0	1	2	3	4	5	6	7
10	20	30	40	50	70	80	90

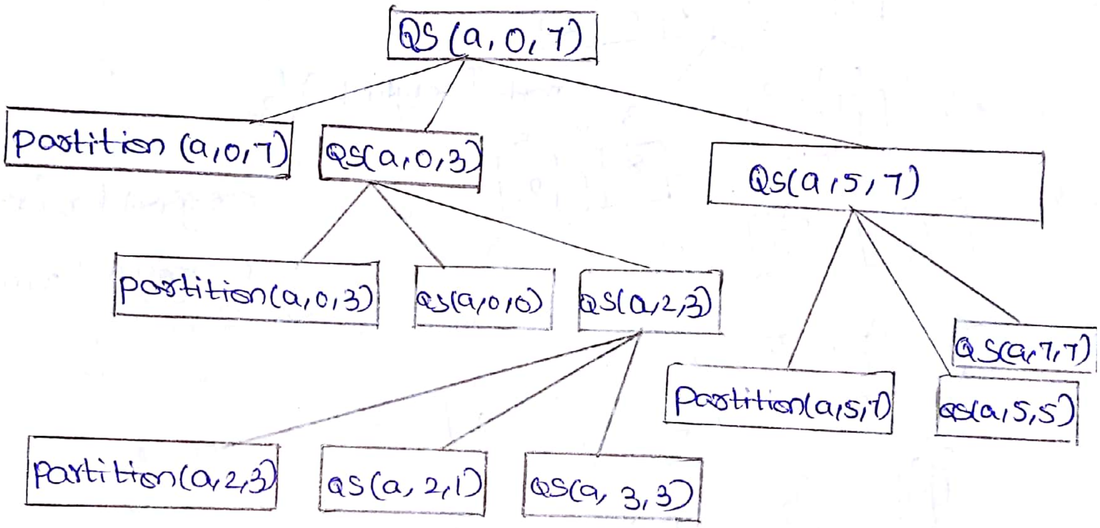
Program to implement quick sort:-

```
#include <iostream.h>
#include <conio.h>
void quicksort (int[ ], int, int);    } prototypes
int partition (int[ ], int, int);
void main( )
{
 int a[50], n, i;
 clrscr( );
 cout<<"Enter the no.of elements: ";
 cin>>n;
 cout<<"\n Enter the elements:";
 for (i=0; i<n; i++)
 {
 cin>>a[i];
  }
                 ->array
  quicksort(a, 0, n-1);
  cout<<"\n The elements after quick sort are:";
  for (i=0; i<n; i++)
  {
   cout<<a[i]<<"  ";
  }
```

```
    getch( );
  }
 void  quicksort (int a[ ],int first, int last)
  {
   int p;
   if (first <last)
   {                  declaration
   p =partition (a, first, last);
   quicksort (a,first, p-1);
   quicksort (a,p+1, last);
   }
  }
                       defination
   int   partition (int a[ ], int l, int r)
   {
    int  pivot , i,j,temp;
    pivot =a[l];
     i =l;
    j=r;
    while( i<j)
     {
     while ((pivot >= a[i]) && (i<=j))
         i++;
      while((pivot < a[j]) && (i<=j))
        j--;
      if (i<j)
       {
        temp =a[j];
        a[j]=a[i];
```

```
        a[i]=temp;
        }
    else
        {
        a[l] = a[j];
        a[j] = pivot;
        }
      }
     return j;
    }
```

Tree Structure for Quick Sort:-



Merge Sort:- The Merge sort is one of the efficient sorting technique which also follows Divide & conquer approach to sort the given list of elements into either ascending (or) Descending order

* In this technique the list of elements is divided into Two parts and each sub-list is again divided into two sub-parts untill each sub-list contains single element. This process is known as Divide process.

* After dividing each sub-list containing single element then all the sub-lists will be conquered (or)

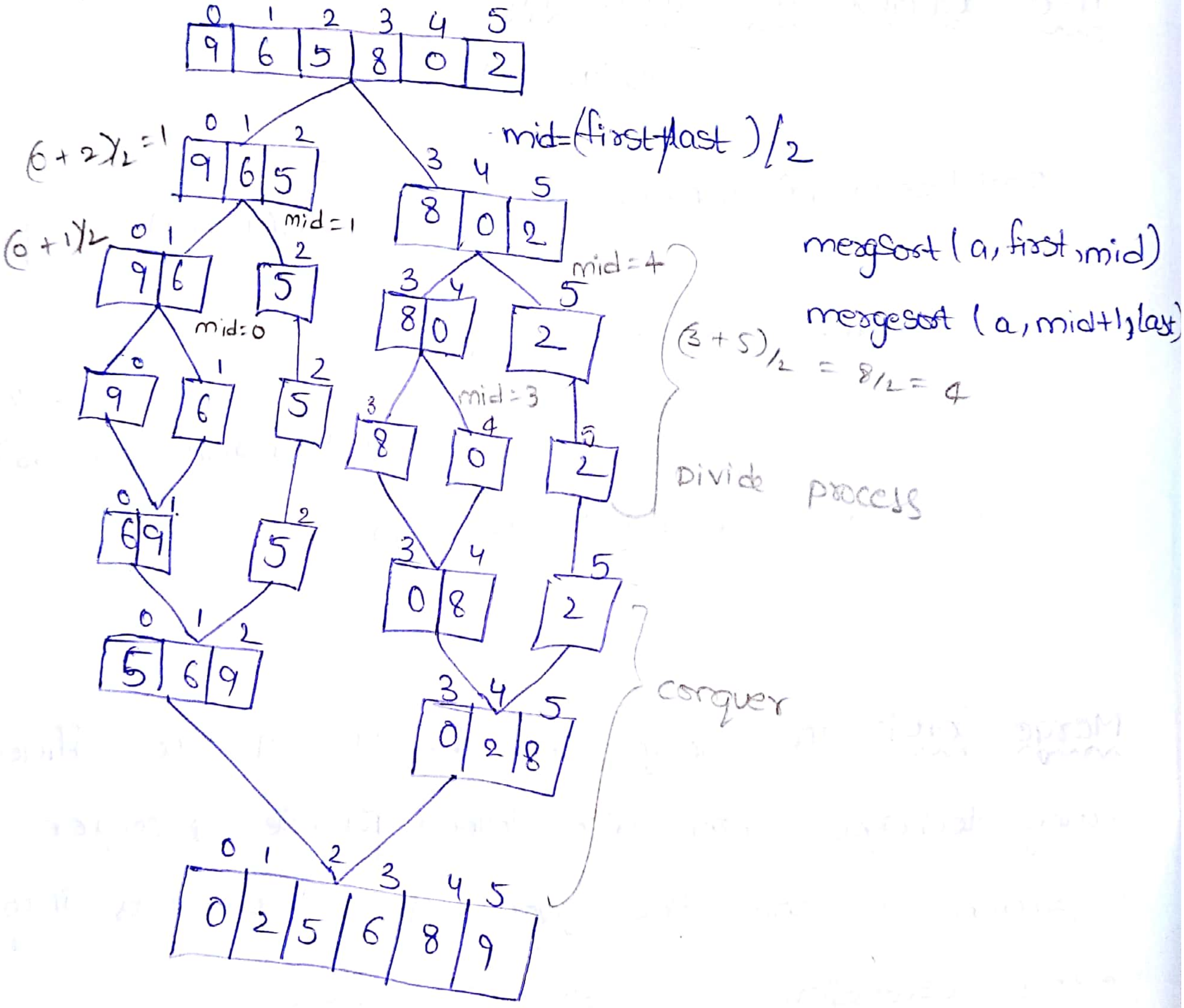
merged. This process is known as conquer process this process will be continued untill all the elements in the list are sorted.

Time complexity:-

For best case is $\overset{\text{order}}{O}(n\log(n))$

Worst case is $O(n\log(n))$

Eg:- 9, 6, 5, 8, 0, 2



Program to implement merge Sort :-

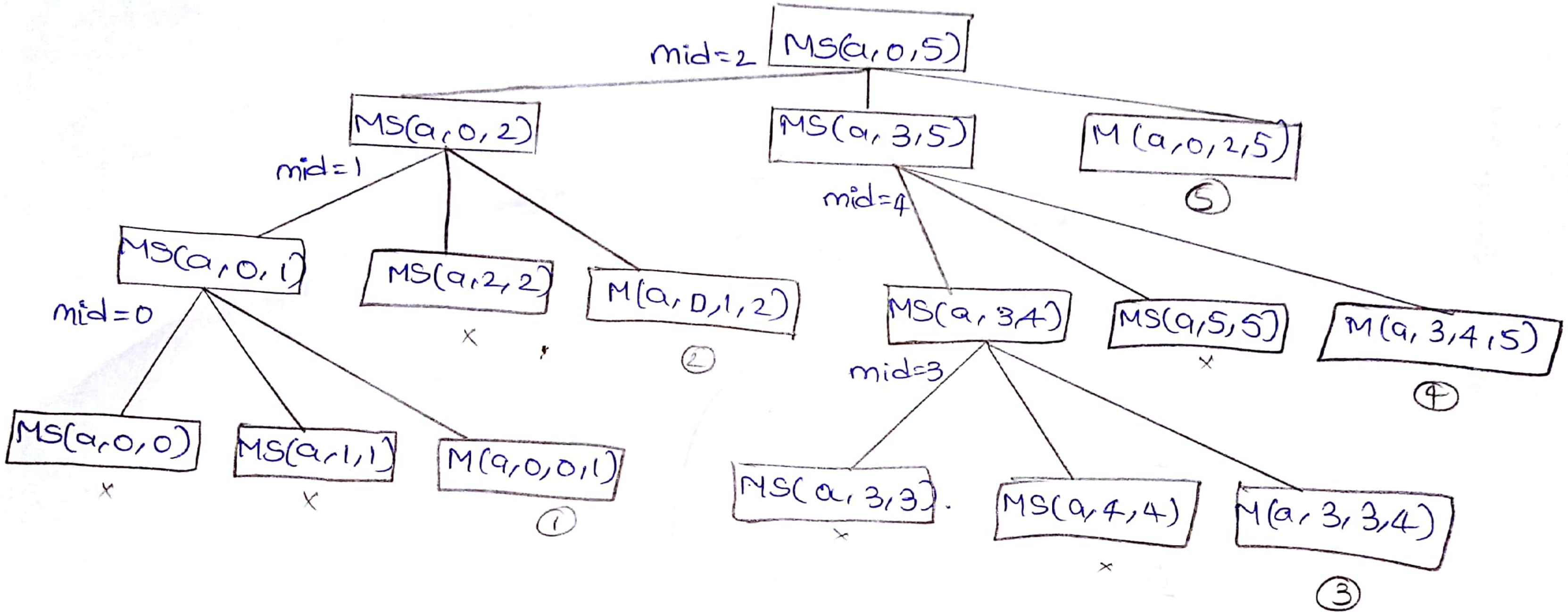
```
#include <iostream.h>
#include <conio.h>
 void mergesort (int[ ],int,int);      } declaration
  void merge (int[ ], int, int, int);
  int a[50], b[50];
   void main( )
   {
    int n, i;
    clrscr( );
    cout<<"Enter the no.of elements:";
    cin>>n;
    cout<<"\n Enter the elements:";
    for(i=0;i<n;i++)
    {
     cin>>a[i];
     }
    mergesort(a,0,n-1)
    cout<<"\n The elements after merge sort are:";
    for(i=0;i<n;i++)
    {
     cout<<a[i]<<" ";
     }
    getch( );
   }   _defination
 void mergesort (int a[ ], int first, int last)
  {
   int mid;
```

```
if (first <last)
  {
mid = (first +last)/2 ;
mergesort (a,first ,mid);
mergesort (a, mid +1, last);
mergesort(a,first, mid , last);
   }
}

Void  merge(int a[ ],int fst, int mid, int lst)
 {
  int f,i,j;
  f1 = fst ;
  i = fst;
  j = mid +1;
  while ((f1 <=mid) &&(j<=lst))
   {
    if (a[f1]< a[j ])
    {
     b[i]=a[f1];
     f1++;
    }
else
   {
    b[i] = a[j];
    j++;
   }
   i++;
 }
while (f1<=mid)
{
```

```
b[i] =a[f1]
f1++;
i++;
}
while(j<=lst)
 {
  b[i]=a[j];
   j++;
   i++;
  }
  for(i=fst;i<=lst;i++)
  {
   a[i]=b[i];
   }
  }
```

Tree structure for Merge Sort:



Iterative Merge Sort:-

program to implement merge sort using iterative loop:-

```
#include<iostream.h>
# include <conio.h>
 int  min(int, int);
 void  mergesort (int [ ], int, int);
 void   merge (int [ ], int, int);
 int   a[50], b[50];
 void  main( )
 {
  int  n, i;
  clrscr( );
  cout<<"Enter  the  no.of  elements:";
  cin>>n;
  cout <<"\n Enter the elements:";
  for(i=0;i<n;i++)
  {
   cin>> a[i];
  }
  mergesort (a, 0, n-1);
   cout<< "\n the  elements after merge  sort  are:";
   for( i=0; i<n; i++)
   {
    cout << a[i]<<"    ";
    }
    getch( );
  }
   int   min(int  x, int  y)
   {
   return(x<y) ? x:y;
```

```
}
void mergesort(int a[ ], int low, int high)
{
 int  m,k,first,mid,last;
 for (m=1; m<=high; m= 2*m)
  {
  for (k=low; k<high; k =k+(2*m))
   {
   first =k;
   mid = k+m-1;
   last =min (k+ (2*m) -1, high)
   merge (a,first, mid, last);
     }
   }
  }
 void  merge (int a[ ],int  fst, int  mid, int  lst)
  {
  int  f1 ,i,j;
  f1 =fst;
   i = fst;
  j = mid+1;
  while((f1 <=mid) && (j <=lst))
   {
    if (a[f1]<a[j])
    {
     b[i] = a[f];
     f1++;
   }
  else
   {
   b[i] = a[j];
   j++;
```

```
 }
 i++;
}
while(f1<=mid)
{
b[i]= a[f1];
 f1++;
 i++;
}
while(j<=lst)
{
 b[i]=a[j];
  j++;
  i++;
}
 for(i=fst; i<=lst; i++)
  {
   a[i]=b[i];
  }
 }
```