```
Int fact(int N)
{
    Int i, fact=1 ;
   For(i=2; i<=N; i++)
         Fact  *=  i;
  return fact;
}
```

```
Int  Fact( int N)
 {
    If(N==0)  return 1;
   return   N * Fact(N-1);
}
```

```
Fact(4)                    Fact(3)                    Fact (2)                   Fact(1)
   {                          {                          {                          {
                                   Return 3 * Fact(2)         return   2 * Fact(1)     return 1*fact(0)
    Return 4 *  Fact(3)
   }                          }                          }                          }
```

```
Fact(0)
{
  return 1;
```

```
}


int power(int k, int n)  Power(2,3)
 {
  // raise k to the power n
  if (n == 0)
    return 1;
  else
    return  k * power(k, n – 1);
}

Power(2,3)
 {
  // raise k to the power n
  if (n == 0)
    return 1;
  else
    return k * power(k, n – 1);   //  2 *power(2, 2)
}

Power(2,2)
 {
  // raise k to the power n
  if (n == 0)
    return 1;
```

```
  else
    return k * power(k, n – 1); // 2* power(2, 1)
}

Power(2,1)
 {
  // raise k to the power n
  if (n == 0)
    return 1;
  else
    return k * power(k, n – 1);  //  2  * power(2, 0)
}

Power(2,0)
 {
  // raise k to the power n
  if (n == 0)
    return 1;
  else
    return k * power(k, n – 1);
}
```

```
Int GCD( int M, int N)
{
   If (N==0 ) return M;
  If ( M < N ) return GCD(N, M);
  Return GCD( N, M%N);
}

GCD(6, 10)
{     // If (N==0 ) return M;
    If ( M < N ) return GCD(10, 6);
   //  Return GCD( N, M%N);
}

GCD(10, 6)
{     // If (N==0 ) return M;
 //   If ( M < N ) return GCD();
     Return GCD( 6, 4);
}
GCD(6, 4)
{    //  If (N==0 ) return M;
 //   If ( M < N ) return GCD();
     Return GCD( 4, 2);
}
```

```
GCD(4,2)
{     If (N==0 ) return M;
   If ( M < N ) return GCD();
    Return GCD( 2, 0);
}
GCD(2,0)
{     If (N==0 ) return M;
  // If ( M < N ) return GCD();
   // Return GCD( );
}

Int max(int A[], int N)
{
   Int large;
   If(N==0) return A[N];
  large = max(A, N-1);
  If( A[N] > large return A[N];
 Return   large;
}
Void main()
{   int N, A[10], i;
     //Read N;   // 5
     // Read Array;  // 10 20 80 40 50
    Printf("largest element is: %d\n", max(A, N-1);
}
```

```
Int max(  [10, 20, 80, 40, 50],  4)
{
    Int large;
    If(N==0) return A[N];
   large = max(A, N-1); // max([10, 20, 80, 40, 50],  3)
 If( A[N] > large return A[N];
  Return   large;  // 80
}

Int max(  [10, 20, 80, 40, 50],  3)
{
    Int large;
    If(N==0) return A[N];
   large = max(A, N-1); // max([10, 20, 80, 40, 50],  2)
 If( A[N] > large return A[N];
  Return   large;
}

Int max(  [10, 20, 80, 40, 50],  2)
{
    Int large;
    If(N==0) return A[N];
   large = max(A, N-1); // max([10, 20, 80, 40, 50],  1)
 If( A[N] > large return A[N];
  Return   large;
```

```
}
Int max(  [10, 20, 80, 40, 50],  1)
{
    Int large;
    If(N==0) return A[N];
   large = max(A, N-1); // max([10, 20, 80, 40, 50],  0)
 If( A[N] > large return A[N];
  Return   large;
}

Int max(  [10, 20, 80, 40, 50],  0)
{
    Int large;
    If(N==0) return A[N];
   large = max(A, N-1); // max([10, 20, 80, 40, 50],  0)
 If( A[N] > large return A[N];
  Return   large;
}


Int sum ( int A[], int  N)   //sum(  [1,2,3,4,5] ,   4)   // sum([10], 0)
{
     If(N==0) return A[N];
    Return A[N]+ sum(A, N-1)  //   5 + sum([1,2,3,4,5] ,  3)  5 + 10 = 15

}
```

Int sum([1,2,3,4,5] , 3)

{

If(N==0) return A[N];

Return A[N]+ sum(A, N-1) // 4 + sum([1,2,3,4,5] , 2) 4 + 6 =10

}

Int sum([1,2,3,4,5] , 2)

{

If(N==0) return A[N];

Return A[N]+ sum(A, N-1) // 3 + sum([1,2,3,4,5] , 1) 3 +3 = 6

}

Int sum([1,2,3,4,5] , 1)

{

If(N==0) return A[N];

Return A[N]+ sum(A, N-1) // 2 + sum([1,2,3,4,5] , 0) 2 +1 = 3

}

Int sum([1,2,3,4,5] , 0)

{

```
If(N==0) return A[N];
    Return A[N]+ sum(A, N-1)  //   2 + sum([1,2,3,4,5] ,  0)

}
```

```
Void main()
{
  Int N, A[10], i;
  // Read  N and A
  Printf("Sum = %d", sum(A, N-1))
}

Int  Binary_serach( int A[ ], int low,  int  high,   int key)
{
   Int  mid;
   If( low>high)  return -1;
   mid = (low+high) / 2;
  If (A[mid]  ==  key)   return mid;
 If (A[mid]  < key)
    return  Binary_serach(A, mid+1,  high, key);
Else
  return Binary_serach(A, low,  mid-1, key);
```

```
}

Void main()
{
    Int A[10], N, key, keypos;
  // Read N and array  and key
  Keypos = Binary_serach(A, 0, N-1, key)
  // check for successful and unsuccessful cases
}
Int Binary_serach( [10,20,30,40],   0, 3, 40)
{
   Int mid;
   If( low>high)  return -1;
   mid = (low+high) / 2;
  If (A[mid]  ==  key)   return mid;
 If (A[mid]  < key)
    return  Binary_serach(A, mid+1,  high, key);
Else
  return Binary_serach(A, low,  mid-1, key);
}
Int Binary_serach([10,20,30,40], 2, 3, 5)
{
   Int mid;
   If( low>high)  return -1;
   mid = (low+high) / 2;
  If (A[mid]  ==  key)   return mid;
```

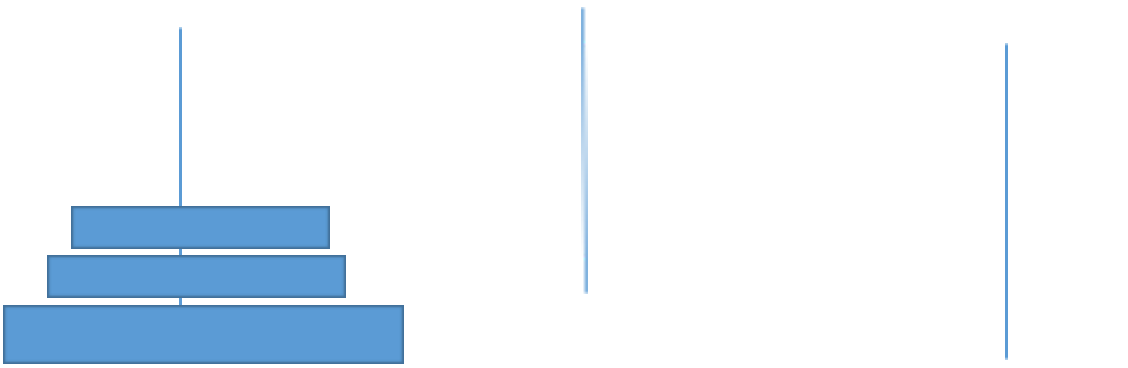
```
 If (A[mid]  < key)
    return  Binary_serach(A, mid+1,  high, key);
Else
  return Binary_serach(A, low,  mid-1, key);
}

Int Binary_serach([10,20,30,40], 3, 3, 40) // 50
{
   Int mid;
   If( low>high)  return -1;
   mid = (low+high) / 2;
  If (A[mid]  ==  key)   return mid;
 If (A[mid]  < key)
    return  Binary_serach(A, mid+1,  high, key);   //  A, 4, 3, 50
Else
  return Binary_serach(A, low,  mid-1, key);
}
Int Binary_serach([10,20,30,40], 4, 3, 50) // 50
{
   Int mid;
   If( low>high)  return -1;
   mid = (low+high) / 2;
  If (A[mid]  ==  key)   return mid;
 If (A[mid]  < key)
    return  Binary_serach(A, mid+1,  high, key);   //  A, 4, 3, 50
Else
```

```
  return Binary_serach(A, low,  mid-1, key);
}
```

Tower of Hanoi Problem:

Initial setup



Source S Temp T Destination D

Final Setup

Source A Temp Destination C

If N=1 move n from source to Destination

Move N-1 Discs from source to Temp

Move Nth disc from source to Destination

Move N-1 from Temp to Destination.

```
Void Tower(int  N,  char Source,  char  Temp,   char  Dest)
{
    If(N==1)
   {
      Printf("Move %d from %c to %c\n", N, Source, Dest);
  Return;
}
Tower(N-1, Source, Dest, Temp);
Printf("Move %d from %c to %c\n", N, Source, Dest);
Tower(N-1, Temp, Source, Dest);
}
```

```
Void Tower(int  3,  S, T, D)
{
    If(N==1)
   {
      Printf("Move %d from %c to %c\n", N, Source, Dest);
```

```
  Return;
}
Tower(2, S, D, T);
Printf("Move %d from %c to %c\n", N, Source, Dest);
Tower(N-1, Temp, Source, Dest);  // 2, T, S, D
}

Tower(2, S, D, T);    // 2, T,  S,  D
{    if(N==1)   {       }
  Tower(1, S, T, D);  //1, T, D, S  // 1, T,D,S
Printf("Move %d from %c to %c\n", N, Source, Dest);
Tower(N-1, Temp, Source, Dest);    //1 , D, S, T     //   1, S, T, D
}

Tower(1, S, T, D);   //   1, D, S,T    //  1, T, D, S   // 1, S, T, D
{ ----
  Printf(----------)
  Return;
----
}

Tower(1, D, S, T);
{ ----
  Printf(----------)
  Return;
----
```

```
}

Tower( 2, T,  S,  D)
{   if(N==1)   {       }
  Tower(1,  T, D, S)
Printf("Move %d from %c to %c\n", N, Source, Dest);
Tower(N-1, Temp, Source, Dest);    //1, D, S, T  //   1, S, T, D
}

Tower(1, T, D, S);
{ ----
  Printf(----------)
  Return;
----
}
```

Move 1 from S to D

Move 2 from S to T

Move 1 from D to T

Move 3 from S to D

Move 1 from T to S

Move 2 from T to D

Move 1 from S to D