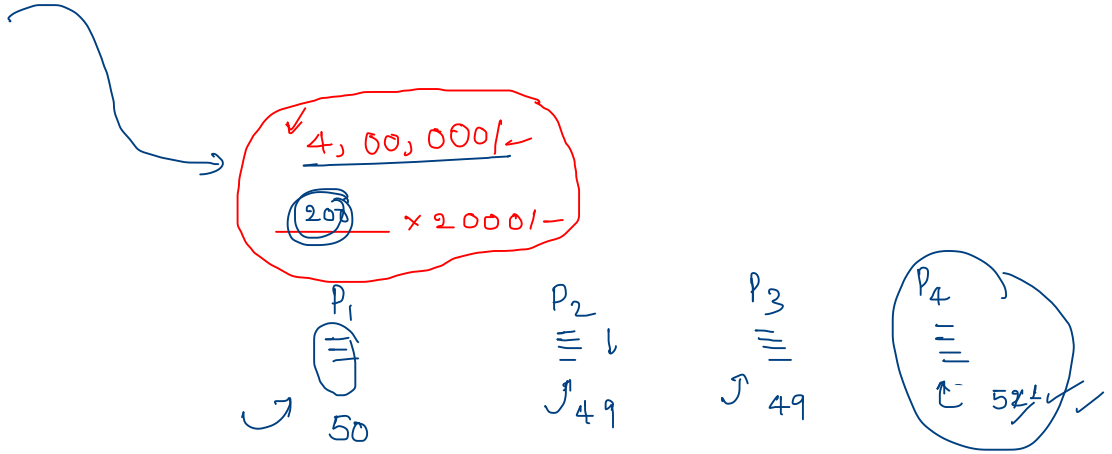
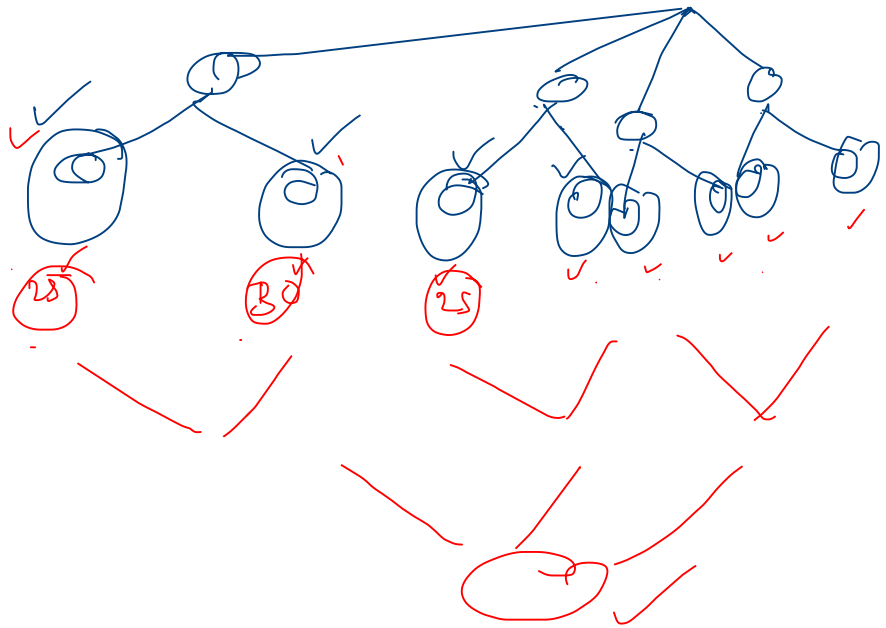
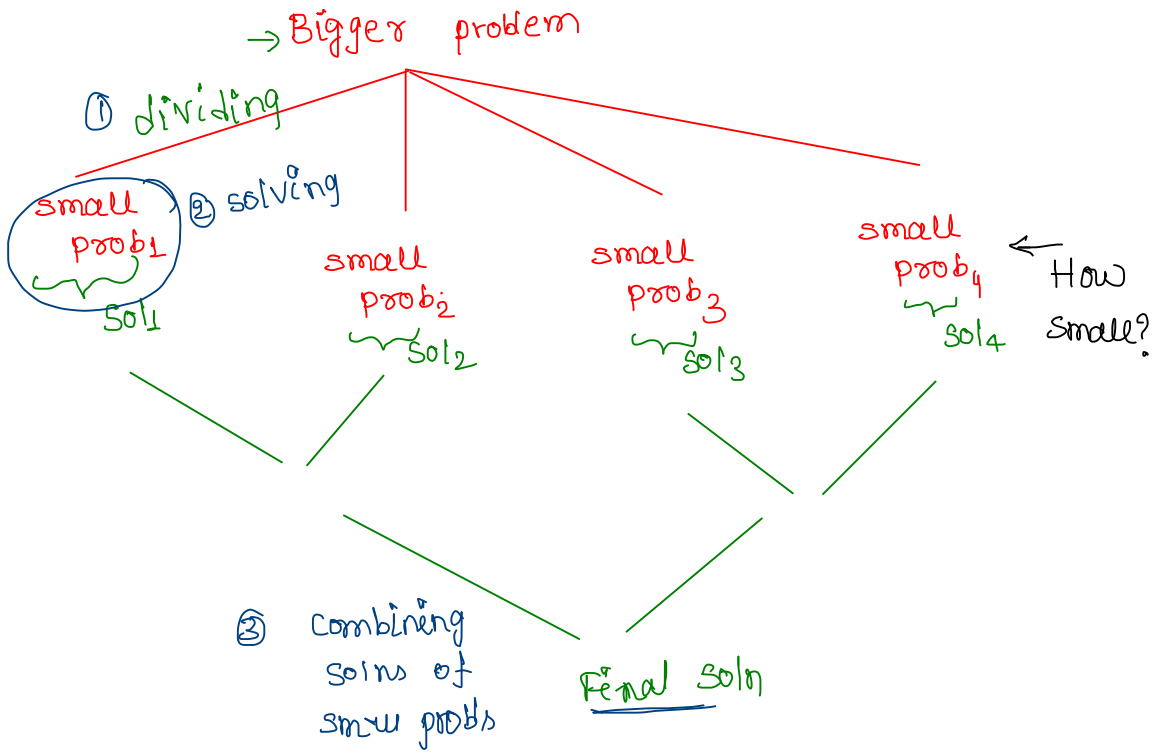
Merge sort



Divide and Conquer :-





Loop ✓

↳ rec ✓

> n ✓

n log n }

n² }

↦ Finding Max ele in an array.

```
public int fun(int arr[], int low, int high)
{
    if(low==high)
        return arr[low];
    if(low<high)
    {
        int mid=low+(high-low)/2;
        return Math.max(fun(arr,low,mid), fun(arr,mid+1,high));
    }
    return Integer.MIN_VALUE;
}
```

	0	1	2	3	4	5
a	-2	1	4	7	9	2

l, h; n = 6

op: -9

$\frac{l + (h-l)}{2} \equiv \frac{l+h}{2}$

$\frac{2l+h-l}{2} = \frac{l+h}{2}$

① ② —∞

f(a, 0, 5) → max ele among indices 0 to 5

↺ 9

m = 2

ret m.max (① f(a, 0, 2) ↺ 4 , ② f(a, 3, 5) ↺ 9)

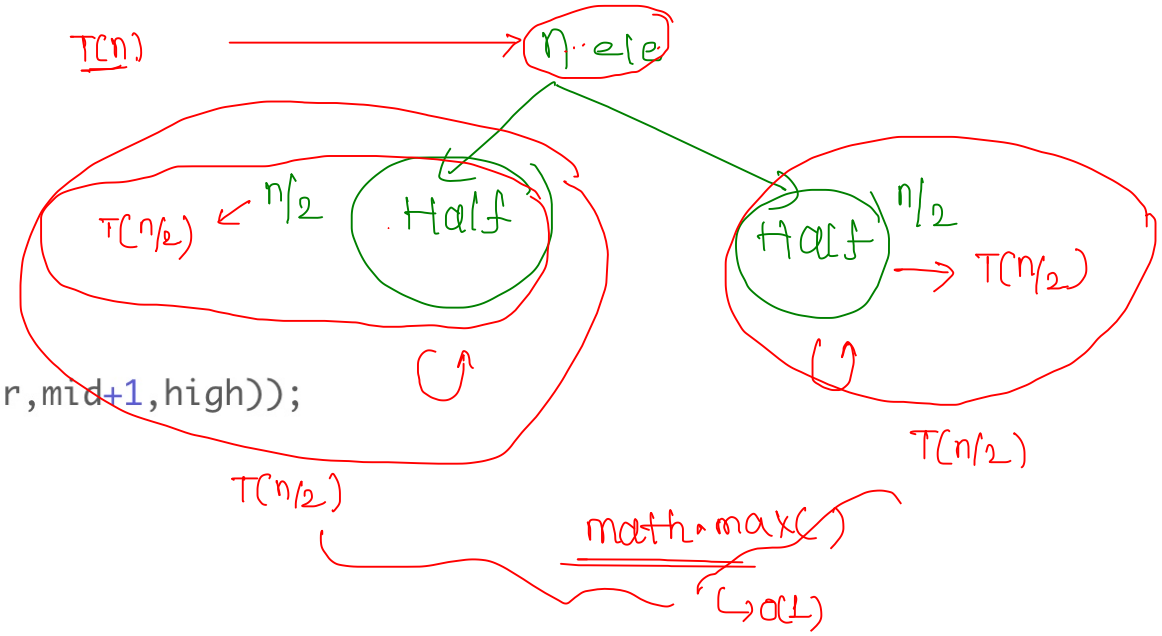
m = 1

ret m.max (① f(a, 0, 1) ↺ 1 , ② f(a, 2, 2) ↺ 4)

m = 0

ret m.max (① f(a, 0, 0) ↺ -2 , ② f(a, 1, 1) ↺ 1).

```java
public int fun(int arr[], int low, int high)
{
    if(low==high)
        return arr[low];
    if(low<high)
    {
        int mid=low+(high-low)/2;
        return Math.max(fun(arr,low,mid), fun(arr,mid+1,high));
    }
    return Integer.MIN_VALUE;
}
```



Let T(n) : time taken to find the maximum element among array of n elements

$$T(n) = \begin{cases} 1 & ; \ n == 1 \\ T(n/2) + T(n/2) + 1 & ; \ n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} 1 & ; \ n == 1 \\ 2T(n/2) + 1 & ; \ n \geq 2 \end{cases}$$

$$T(n) = \begin{cases} 1 & ; \ n == 1 \ (\text{base-case}) \\ 2T(n/2) + 1 & ; \ n \geq 2 \end{cases}$$

Sol)

$T(n) = 2 \cdot T(n/2) + 1 \rightarrow (1)$ (original eqn)

$T(n/2) = 2 \cdot T(n/4) + 1 \rightarrow (2)$

substitute (2) in (1)

$T(n) = 2 \cdot [2 \cdot T(n/4) + 1] + 1$

$= 2^2 \cdot T(n/2^2) + 2 + 1 \rightarrow (3)$

$T(n/2^2) = 2 \cdot T(n/2^3) + 1 \rightarrow (4)$

substitute (4) in (3)

$T(n) = 2^2 [2 \cdot T(n/2^3) + 1] + 2 + 1$

$= 2^3 \cdot T(n/2^3) + 2^2 + 2 + 1$

$T(n) = 2^4 \cdot T(n/2^4) + 2^3 + 2^2 + 2 + 1$

$T(n) = 2^5 \cdot T(n/2^5) + 2^4 + 2^3 + 2^2 + 2 + 1$

$S_n = \frac{a(r^n - 1)}{r - 1}, \ r \neq 1$

$\vdots$

after k-steps (it will stops)

$= 2^k \cdot T(n/2^k) + 2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2 + 1$

$\frac{n}{2^k} = 1$ i.e $T(1)$

$n = 2^k \Rightarrow k = ?$

$\log_2 n = \log_2 2^k$

$= k \cdot \log_2 2$

$\log_2 n = k$

$= 2^k \cdot T(\frac{n}{2^k}) + \frac{2^k - 1}{2 - 1}$ (1 + 2 + 4 + 8 + ... + 2^k)

$a = 1, \ r = 2$

$= n \cdot T(1) + \frac{1(2^k - 1)}{1}$

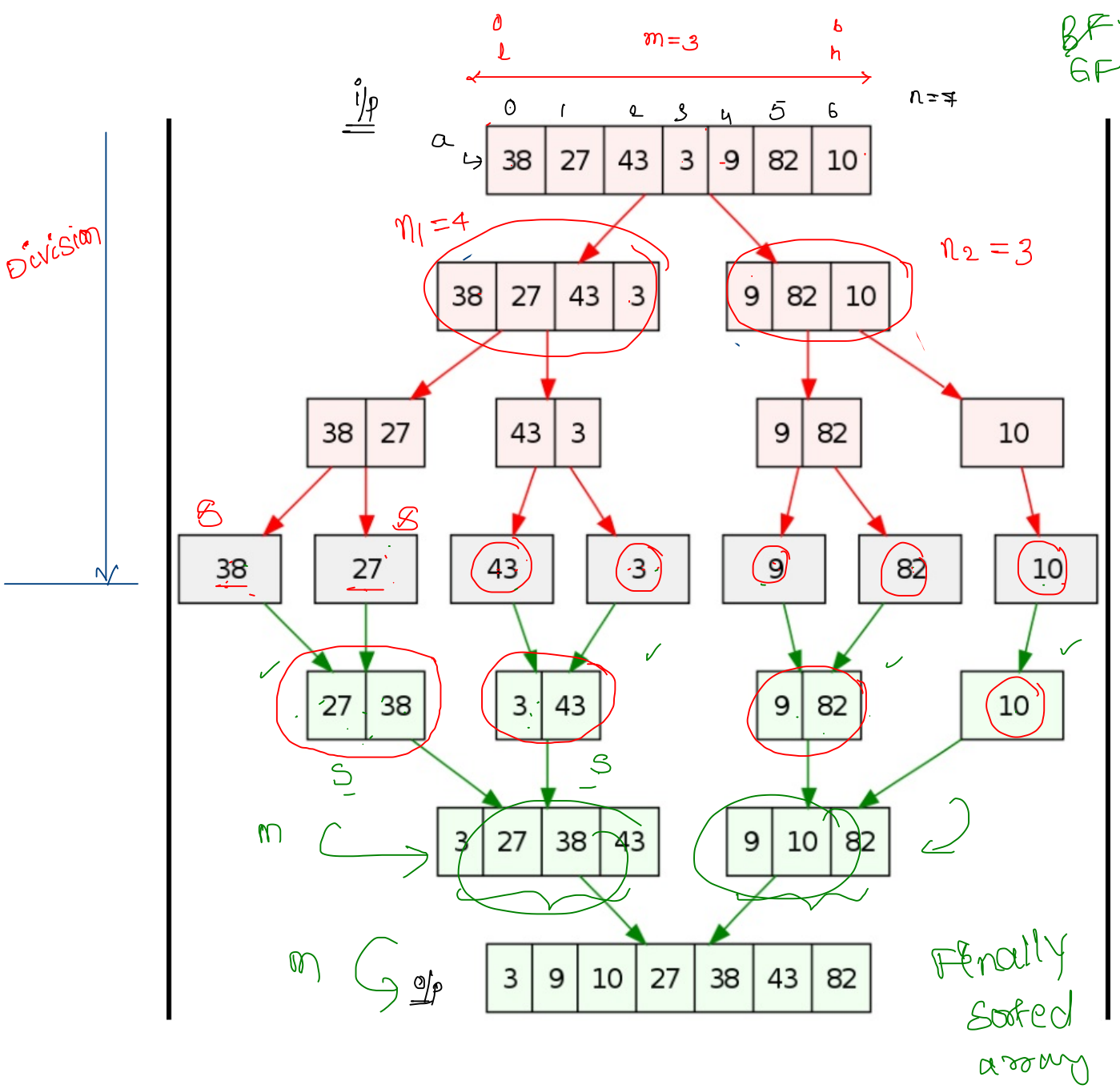
$= n \cdot (1) + (2^k - 1)$

$= n + n - 1 = 2n - 1$

$\hookrightarrow O(n)$

formal way of Finding Rec. code T.C

12:27 → 12:37



Merge
⇒ ALWAYS produce sorted array.

→ How long you should divide?

divide till, no sorting is required

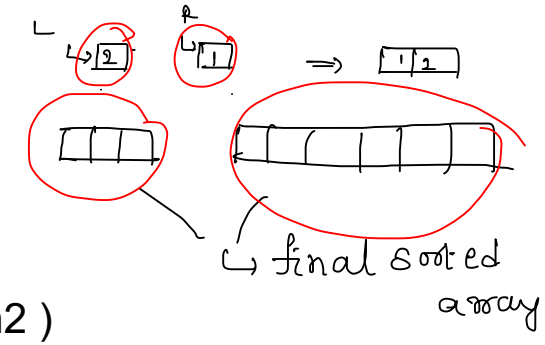
n=1

if you have ONLY one element in the array, THEN by default it is sorted

code : merge two sorted arrays (2ptr)

i/p :- 2 sorted arrays

O(n) → $n_1 + n_2$




```
Merge(L[ ],n1,R[ ],n2)
{
    i=0,j=0,k=0

    while( i<n1 && j<n2 )
    {
        if(L[i]<=R[j] )
        {
          A[k]=L[i]
          i++
        }
        else
        {
           A[k]=R[j]
           j++
        }
       k++
    }
```

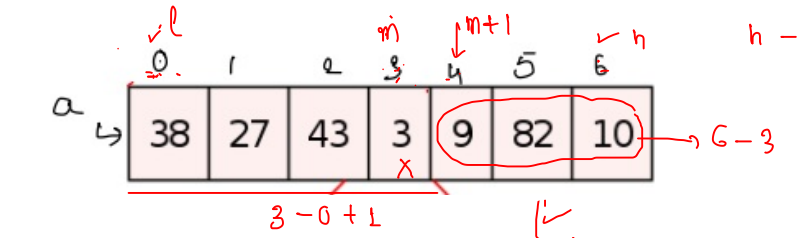
```
//R[] remaining  elements
while(j<n2)
{
   A[k]=R[j]
   j++
   k++
}
//L[] remaining elements
while(i<n1)
{
  A[k]=L[i]
  i++
  k++
 }

}
```

2ptr:
- → L[ ], n_1 ✓
- → R[ ], n_2 ✓

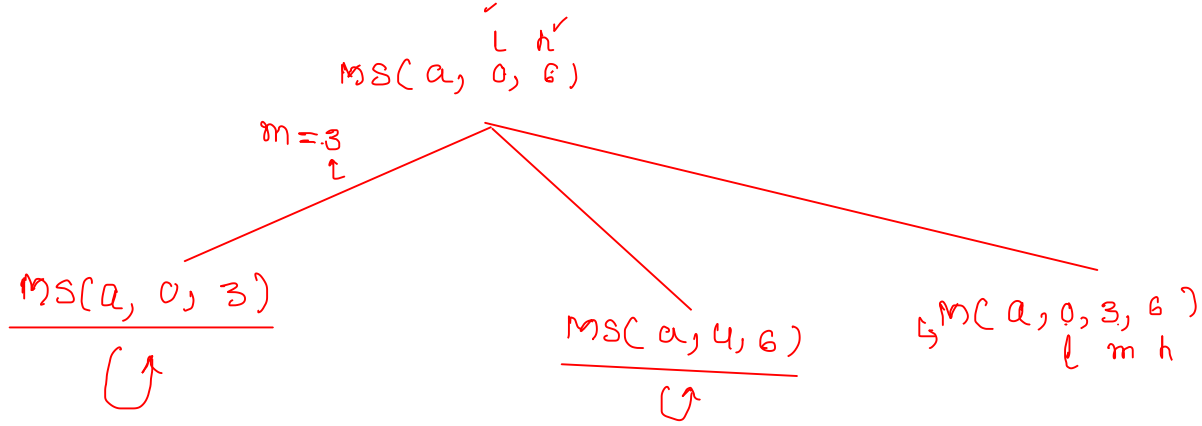
```
main()
{
    // A[] : input array
    // n : size of the array
    mergeSort(A,0,n-1);
}
```

start, ending

```
mergeSort(int arr[], int low, int high)
{
    if(low<high)
    {
        int mid=low+(high-low)/2;
        mergeSort(arr,low,mid);
        mergeSort(arr,mid+1,high);
        merge(arr,low,mid,high);
    }
}
```



o/p :- 3, 27, 38, 43



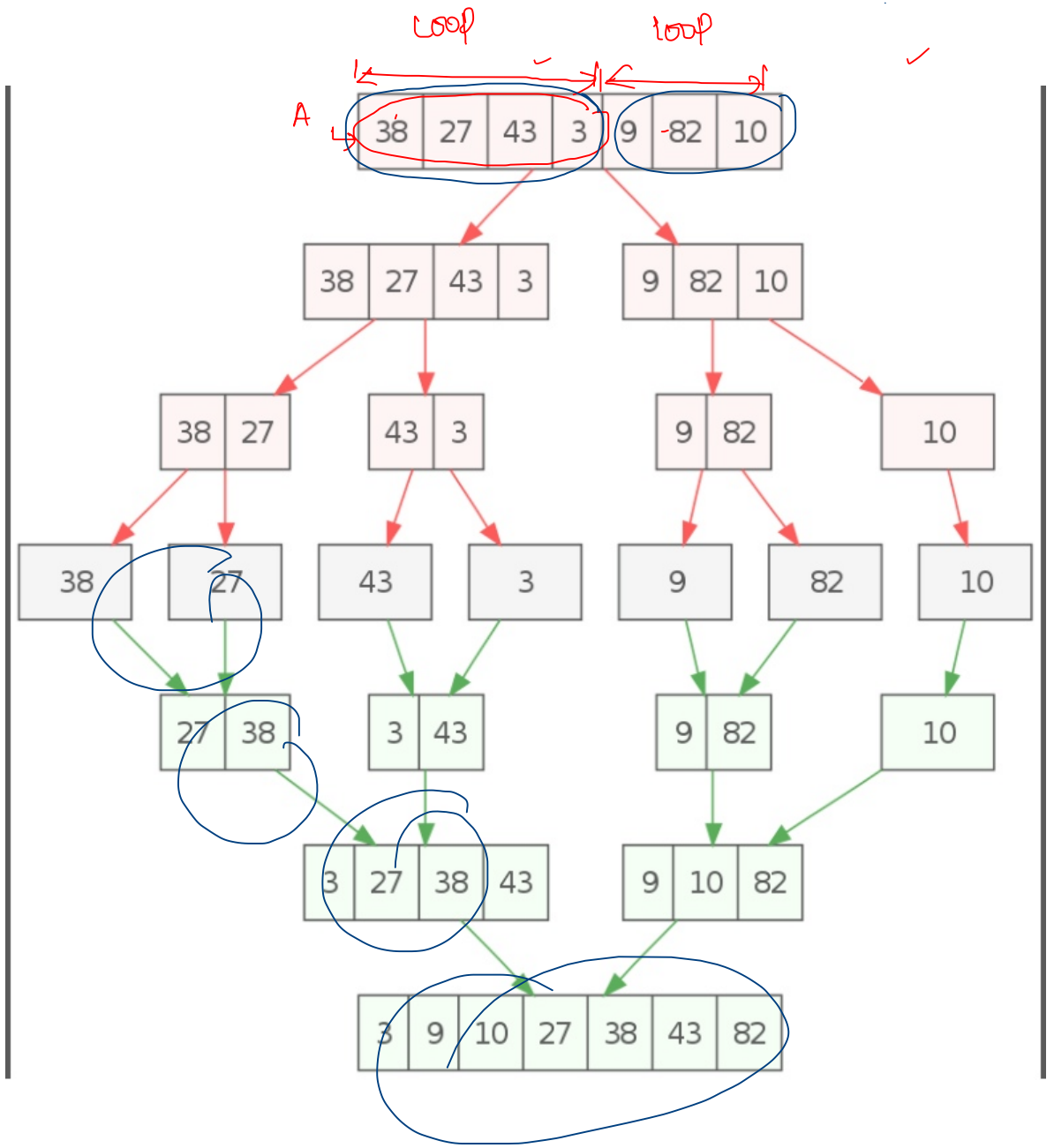
l = 0 ✓ | $n_1 = m - l + 1$ ✓

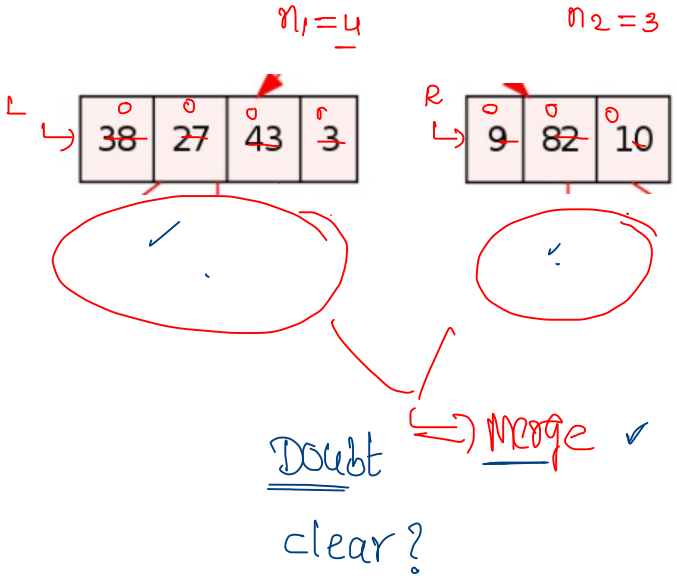
m = 3

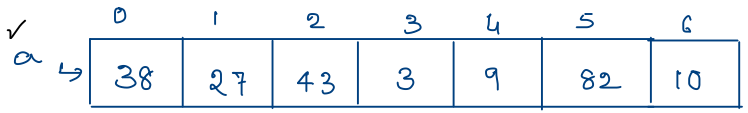
h = 6 ✓ | $n_2 = h - m$ ✓

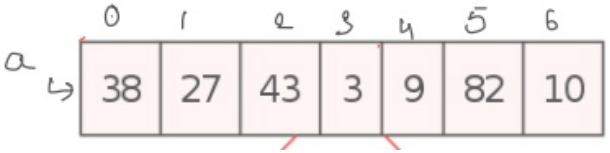
```
Merge(A,l,m,h)
{
   n1=m-l+1
   n2=h-m
    L[n1], R[n2]

   for(i=0;i<n1;i++)
      L[i]=A[l+i]

   for(i=0;i<n2;i++)
      R[i]=A[m+1+i]

   i=0,j=0,k=0

   while( i<n1 && j<n2 )
   {
       if(L[i]<=R[j] )
       {
          A[k]=L[i]
          i++
       }
       else
       {
           A[k]=R[j]
          j++
       }
      k++
   }
```

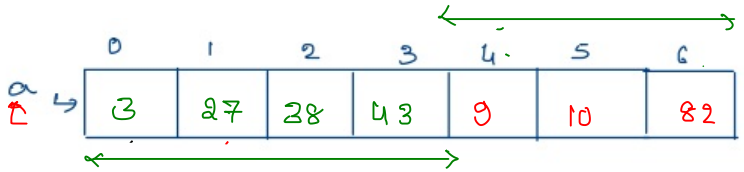
```
//R[] remaining  elements
while(j<n2)
{
   A[k]=R[j]
   j++
  k++
}
//L[] remaining elements
while(i<n1)
{
  A[k]=L[i]
  i++
  k++
 }
}
```

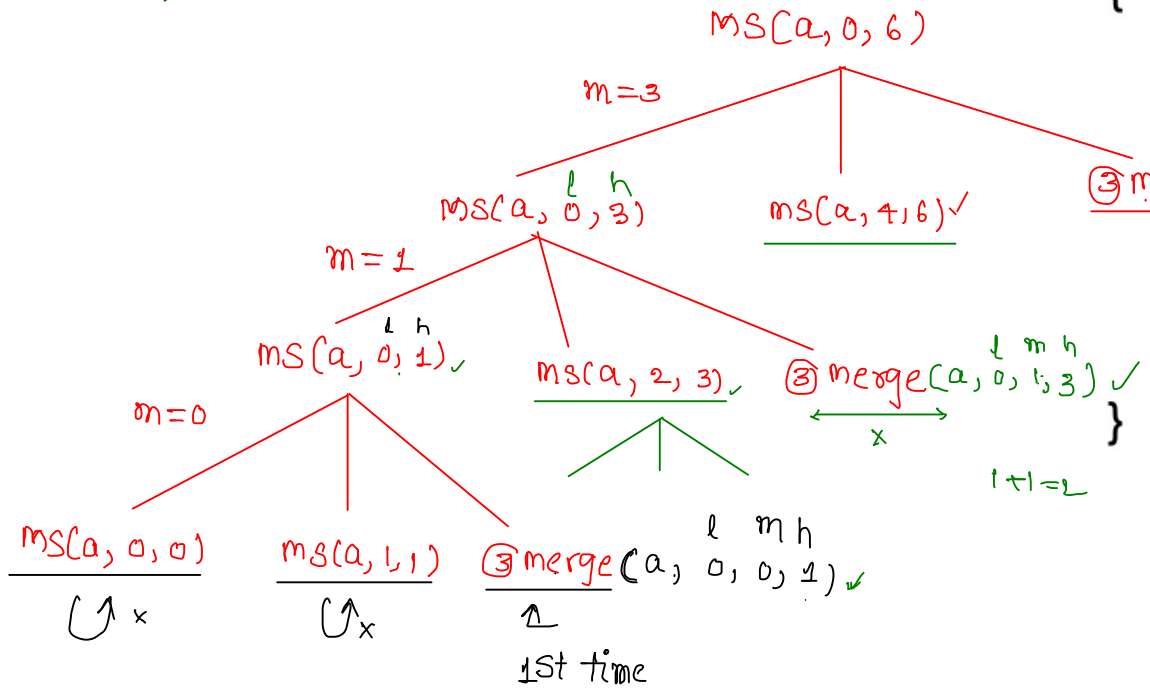

loop
loop
A
38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82


$n_1 = 4$
$n_2 = 3$
L
R
38 27 43 3
9 82 10
Doubt
Merge
clear?










```
② mergeSort(int arr[], int low, int high)
{
    if(low<high)
    {
        int mid=low+(high-low)/2;
①       mergeSort(arr,low,mid);
        mergeSort(arr,mid+1,high);
        merge(arr,low,mid,high);
    }
}
```