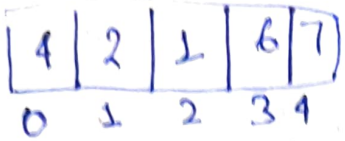
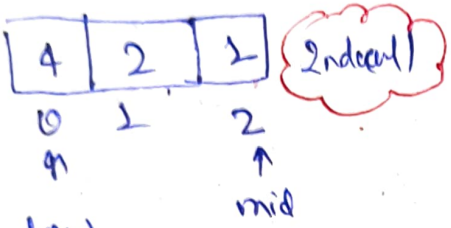
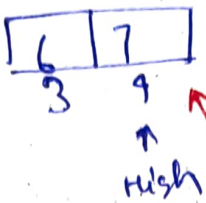


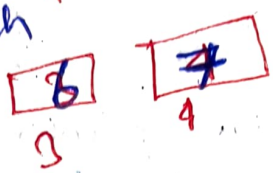

1st call from main

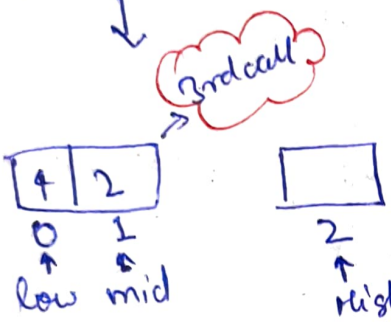
```
m.s (arr, low, high)
{ if (low >= high)
    return;
  mid = (low+high)/2
  m.s (arr, low, mid)
  m.s (arr, mid+1, high)
  m.s (arr, low, mid, high)
```

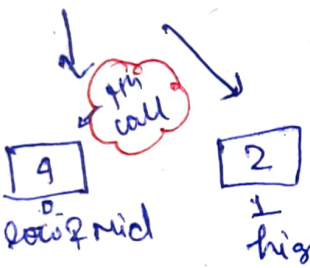
mid = 2
low = 0
high = 4




Now → m.s(arr, low, mid) //for left half (5th call)

m.s(arr, low, high) // This the function

so if (low >= high)

return; // because left half work is done & suppose

[0] size arra
it is "

→ for right half we know → mid+1 & high because low to mid is left half and Mid+1 to high is right half.

→ 5th call returned to 4th call as left half is completed of imaginary array (0, 0) on 5th call and 5th call destroyed because its work done (returned)

→ Now back to 4th call. here → (0, 1) left half

So → (mid+1 and high) is right half

here high was 1 and low was 0 because here size of array was

| 4 | 2 |
|---|---|
| 0 | 1 |

→ So m.s (mid+1, high) // for right array

m.s (1, 1) // 5th call (New call) → previous 5th destroyed because of return

→ return (because) low == high in that function

→ Now again back to 4th call because 4th call function had call 5th call for left and base condition returned it and destroyed caller and when it returns back to 4th call, it calls for right half of array, which new Stack in Recursion Stack (Previous 5th was destroyed)

→ in 5th call (1 1)

low >= high so returned to 4th call again

Now in 4th call m.s (arr, low, mid) // left half

m.s (arr, mid+1, high) // right half

completed 4th function will merge both array left half and right half

→ m.s(arr, low, mid, high);

low to mid → left half; mid+1 to high → right half

In 4th call → (0, 0) left half → Single Element
(1, 1) right half → Single Element

low = 0, mid = 0, high = 1

Now Merging → m.s (arr, low, mid, high)

→ (arr, 0, 0, 1) // 5th call

left half

Although here for right half it is (0, 1) but as we know right half is (1, 1) so no worries in Merge function we will merge like

(low to mid & mid+1 to high)

→ Merge function (arr, low, mid, high)

Now we are again 5th call. previous (5th call destroyed as it return to 4th call).

```
left_start = low;
mid ;  // Ending of left array
right_start = mid+1; // Starting index of Right array.
high; // Ending of right array
```

→ Now we will check every element from left array to right array & which one will be smaller we will shift in temp array. for this function only

→ and after shifting we will copy this temp in our original array, after completion of this function temp array and this function will deleted from recursion stack.

→ Now we are again back to (4th call) now 4th call has nothing to executed it will go back where it was called

→ In (3rd call) where low = 0, high = 2 and mid = 1

3rd call

```
m.s (arr, 0, 2)
{
if (low >= high)
return
mid = (0+2)/2 = 1
m.s (arr, low, mid) // this line totall executed
```

→ now `m.s (arr, mid+1, high)` → it will be called (new 4th call) (2, 2)

in (new 4th call) low >= high it will be returned.

and 4th call destroyed from recursion stack

now again in (3rd call) left array and right array executed

now → merge (arr, 0, 1, 2) Left (0,1) Right (2,2)

again (new 4th call) in which will merge

Now in New 4th call
0 1 2

m (arr, low, mid, high)
{

left_start = low;

mid; // left_end.

~~mid+1~~ 2

right_start = mid+1;

high; // right end


new temp array will created

while (left_start <= mid && right_start <= high)

if ( ~~left_start~~ arr[left_start] <= arr[right_start] )
{

temp.add ( arr[left_start] );
left_start ++;

else
{
temp.add ( arr[right_start] )

right_start ++;

}


temp → 1 now left array is already sorted in previous time so go push all

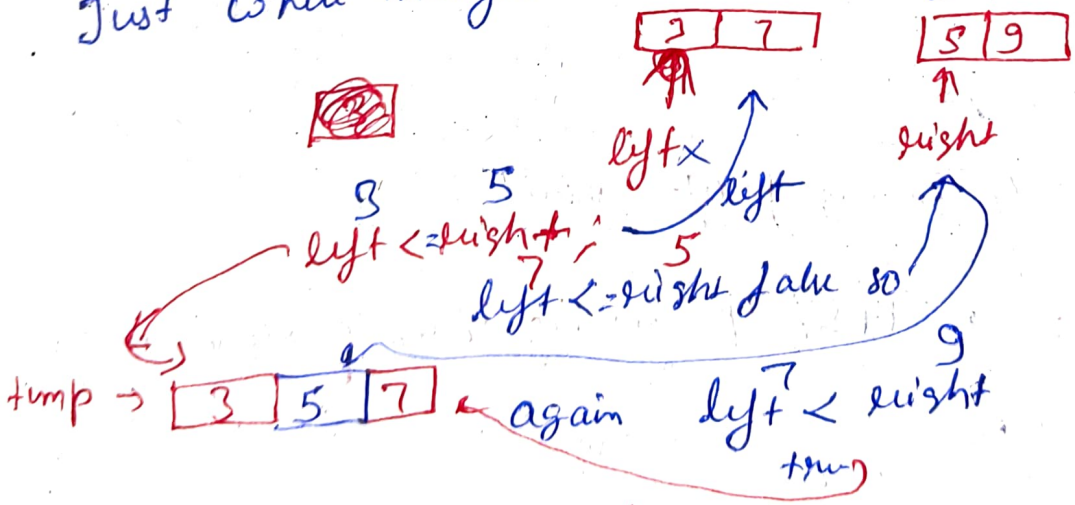
```
while (left_start <= mid)          // if right array is finished,
{                                     in this function right array
    temp.add(arr[left_start]);        is finished this
    left_start++;                     function will run.
}

while (right_start <= high)
{
    temp.add(arr[right_start]);   // if left array is finished
    right_start++;
}
```

Note: → In Every Merge function left array and right array is already sorted

like ex → [3 | 7] (sorted) [5 | 9] (sorted)

Just while merging we are sorting both array.




Now left is finished so all right sorted will merged at end of the left array.

After completion of 4th call, 4th call will be destroyed we are again in 3rd call which called merging

(0, 1, 2)

Now 3rd call destroyed from recursion stack, we are again in 2nd call which called left half array

(0, 1, 2)

In 2nd call

low = 0, high = 4 so ⇒ mid = (2)

Now second line of m.s will execute for right half.

→ m.s (low, mid+1, high); // new 3rd call

---

In 3rd call we are which is new recursion stack

low = 3, high = 4

mid = (3+4)/2 = 3.5 = 3

left array will call in 3rd function

m.s (arr, low, mid) // new 4th call

In 4th call low >= high

return; and 4th call destroyed.

Now Again in 3rd call function

left array was called (3, 3)

Now

right array will be called (arr, mid+1, high) → (arr, 4, 9)

So new (4th call) will generate and return back to 3rd call

now left and right array called

now merge function will called to merge both left and right array.

m (arr, low, mid, high) → m (arr, 0, 3, 4)

New 4th call in recursion stack to merge.

this will merge entire array, and our array is sorted and 3rd call be destroyed Temp.

then come back 2nd call, no statement to execute

now so (2nd call destroyed) from recursion stack

now it will go main function which had called m.s

and now function executing of next-line will start.