On Recursion

704. Binary Search

Easy 4389 103 Add to List Share

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

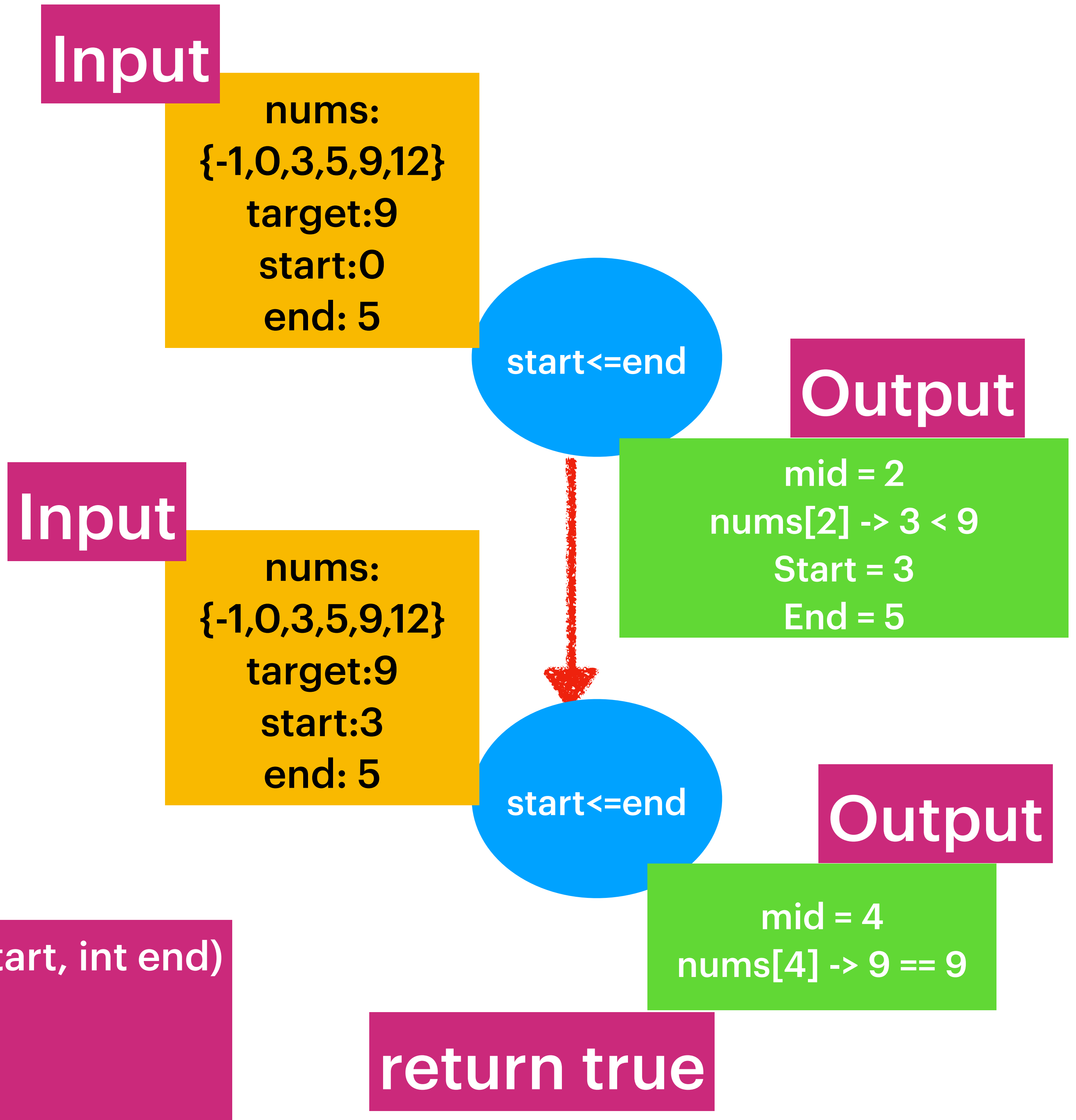
Constraints:

- $1 <= \texttt{nums.length} <= 10^4$
- $-10^4 < \texttt{nums[i], target} < 10^4$
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

nums = [-1,0,3,5,9,12], target = 9

Time Complexity : O(logn)
Space Complexity : O(logn)

BaseCheck —> start > end

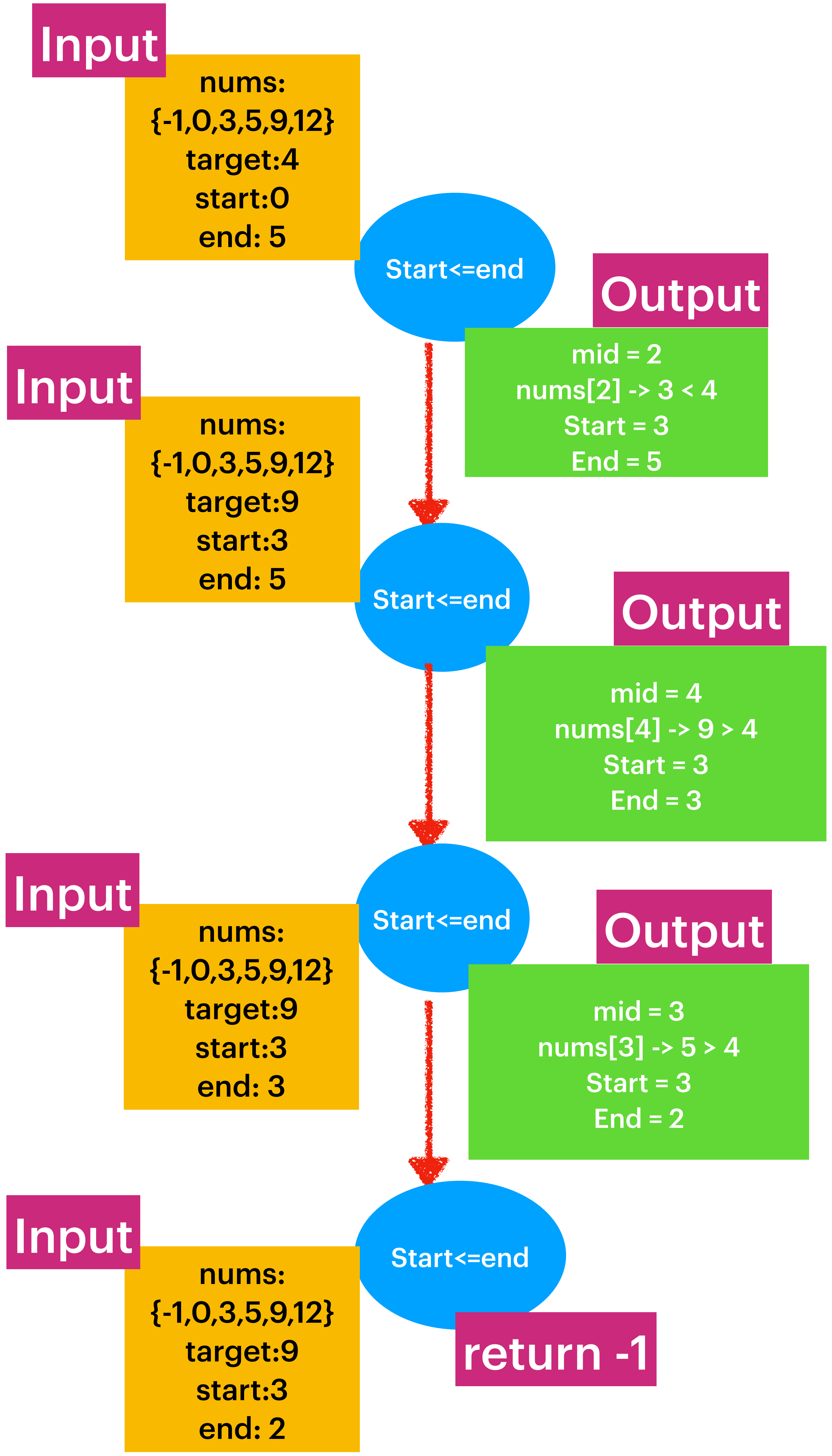



```
private int binarySearch(int[] nums, int target , int start, int end)
{
if( start > end)
{
return -1;
}

int mid = start + (end-start)/2;

if(nums[mid] == target )
{
return mid;
}
else if(nums[mid] < target )
{
return binarySearch(nums, target, mid+1, end);
}else // nums[mid] > target
{
return binarySearch(nums, target, start, mid-1);
}
}
```

Time Complexity : O(logn)
Space Complexity : O(logn)



nums = [-1,0,3,5,9,12], target = 4

BaseCheck —> start > end

Divide And Conquer :
Divide the main problem into possible subproblems, solve the each sub problem recursively so that we can obtain the solution to main problem.
Main Problem
Sub Problem 1
Sub Problem 2
Solution Of Sp1
Solution Of Sp2
Obtain the result for Main Problem
Divide
Conquer
Sort & Merge
Main Problem
Sort Elements
4
1
3
2
Base check, Array is sorted If the length is One. So that we can divide the array till the length is '1'.
leftArr
rightArr
4
1
3
2
SubProblem1
SubProblem2
leftArr
rightArr
leftArr
rightArr
4
1
3
2
SP1'1
SP1'2
SP2'1
SP2'2
Compare , swap & Merge
Compare , swap & Merge
1
4
Solution to SubProblem1
2
3
Solution to SubProblem2
Compare , swap & Merge
1
2
3
4
Solution to Main Problem

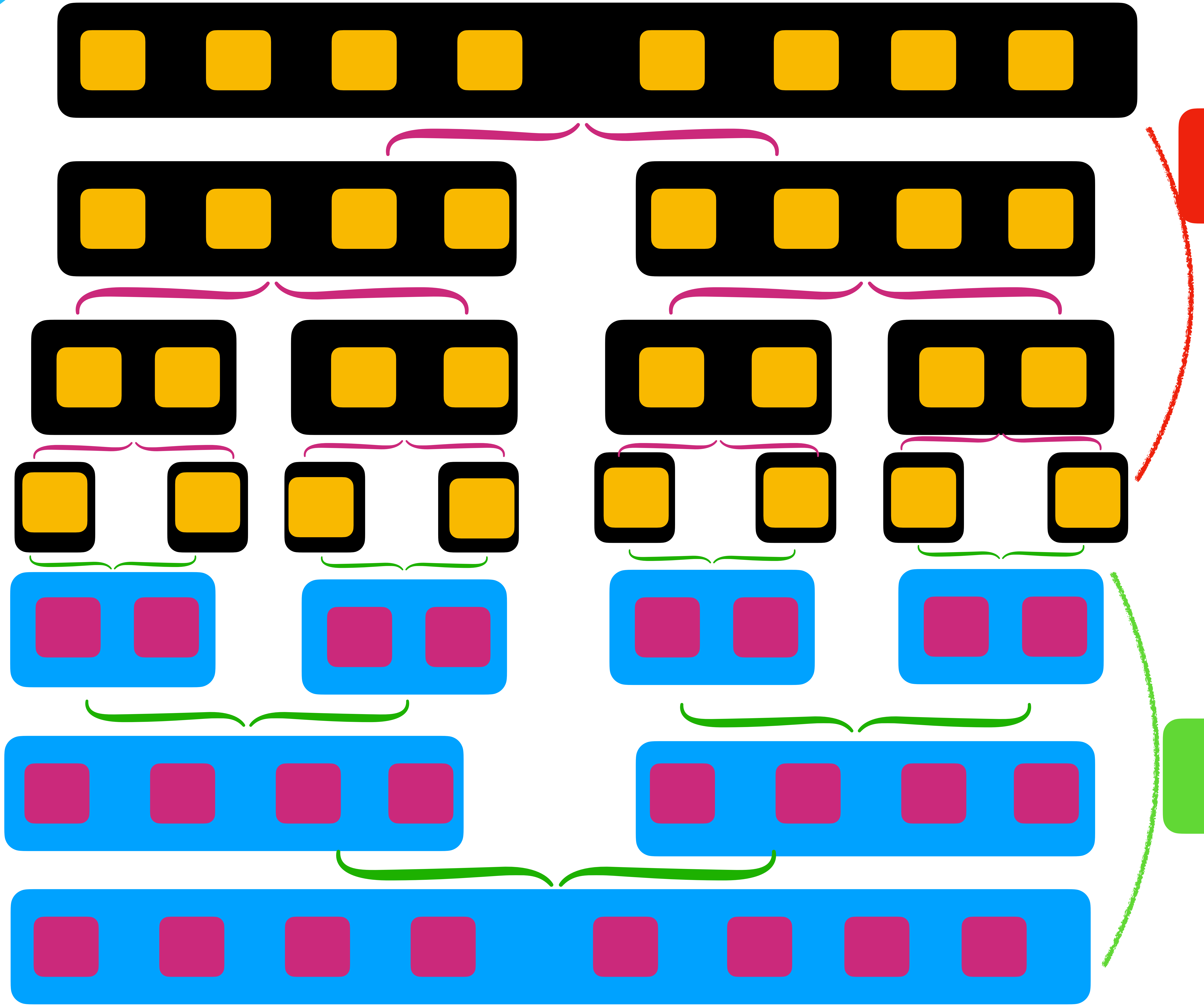
MergeSort

Merge Sort uses divide and conquer pattern.

Merge sort divides the problem into possible small problems then applies sorting recursively.

Divide => divides source collection into possible n/2 sub problems recursively.

Conquer=> Applies the sorting at subproblem level (compare, swap & merge) then



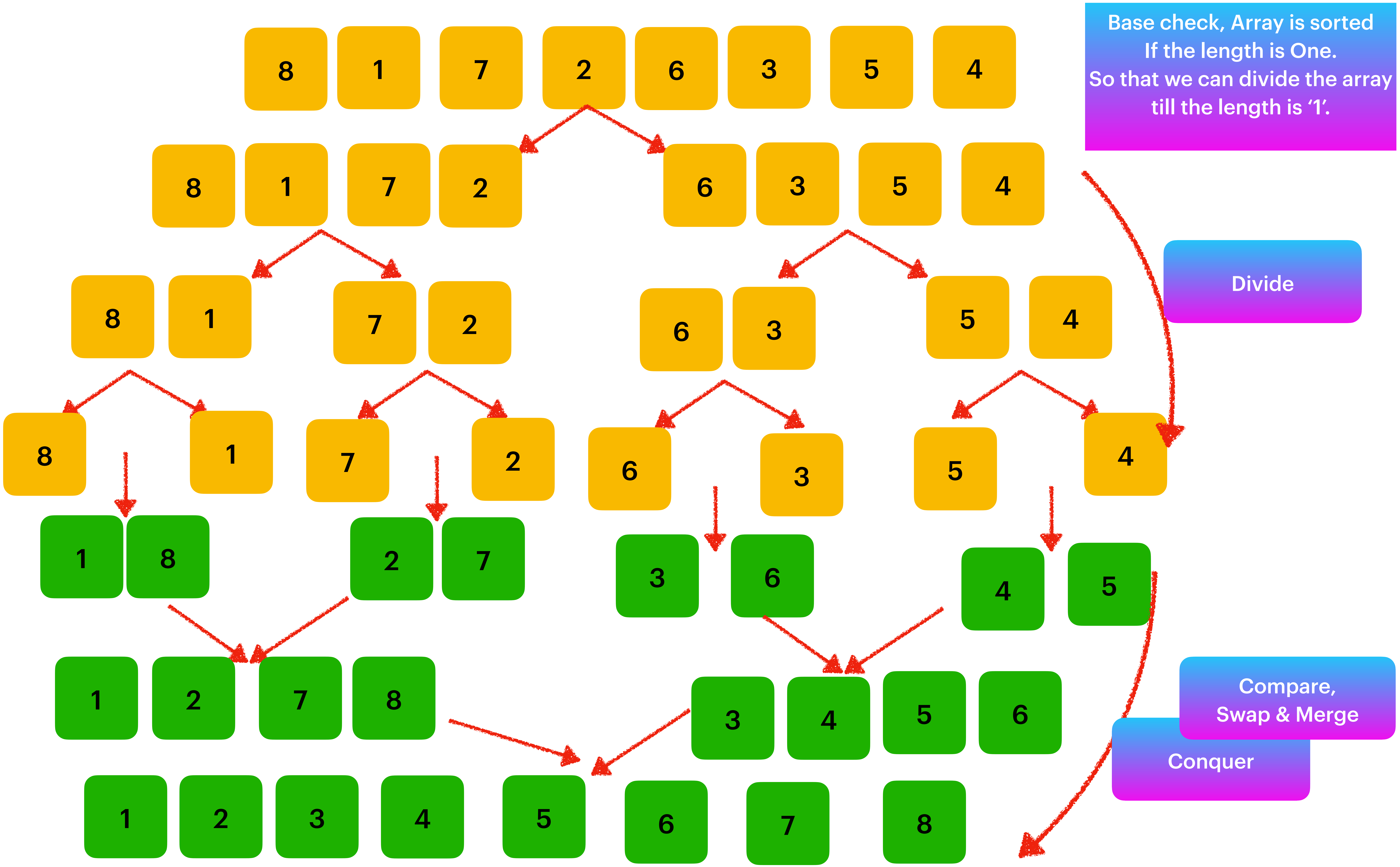
Divide

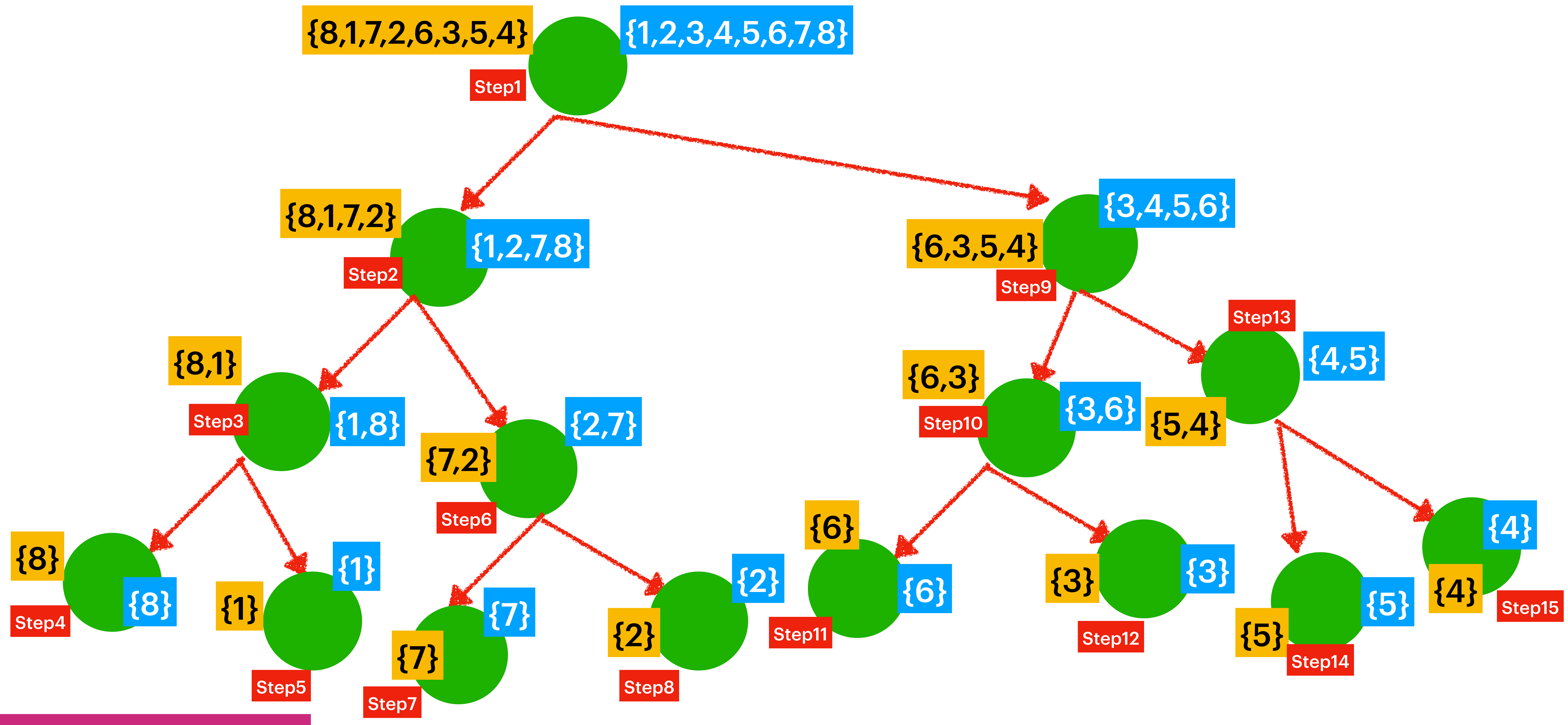
Divide=> Break up the problem into smallest possible sub problems.

Compare, Swap & Merge

Conquer

Conquer=> Figure out the solution for the smallest sub problem, then apply the same technique to solve larger problems recursively .


8 1 7 2 6 3 5 4
Base check, Array is sorted
If the length is One.
So that we can divide the array
till the length is '1'.
8 1 7 2
6 3 5 4
Divide
8 1
7 2
6 3
5 4
8
1
7
2
6
3
5
4
1 8
2 7
3 6
4 5
1 2 7 8
3 4 5 6
Compare,
Swap & Merge
Conquer
1 2 3 4 5 6 7 8



Time Complexity :

Lets analyse divide & conquer.
Divide operation
is always constant.
Conquering involves processing and
is varied
based on input size and behaviour of
elements.

So in the above use case we had 3 levels
and in each level in worst case there could
be 8 swaps.
8 * 3= 24 swaps .
Here 8 is the input length and log2^(8) = 3
So I can replace 8 * 3 = n * log(n).

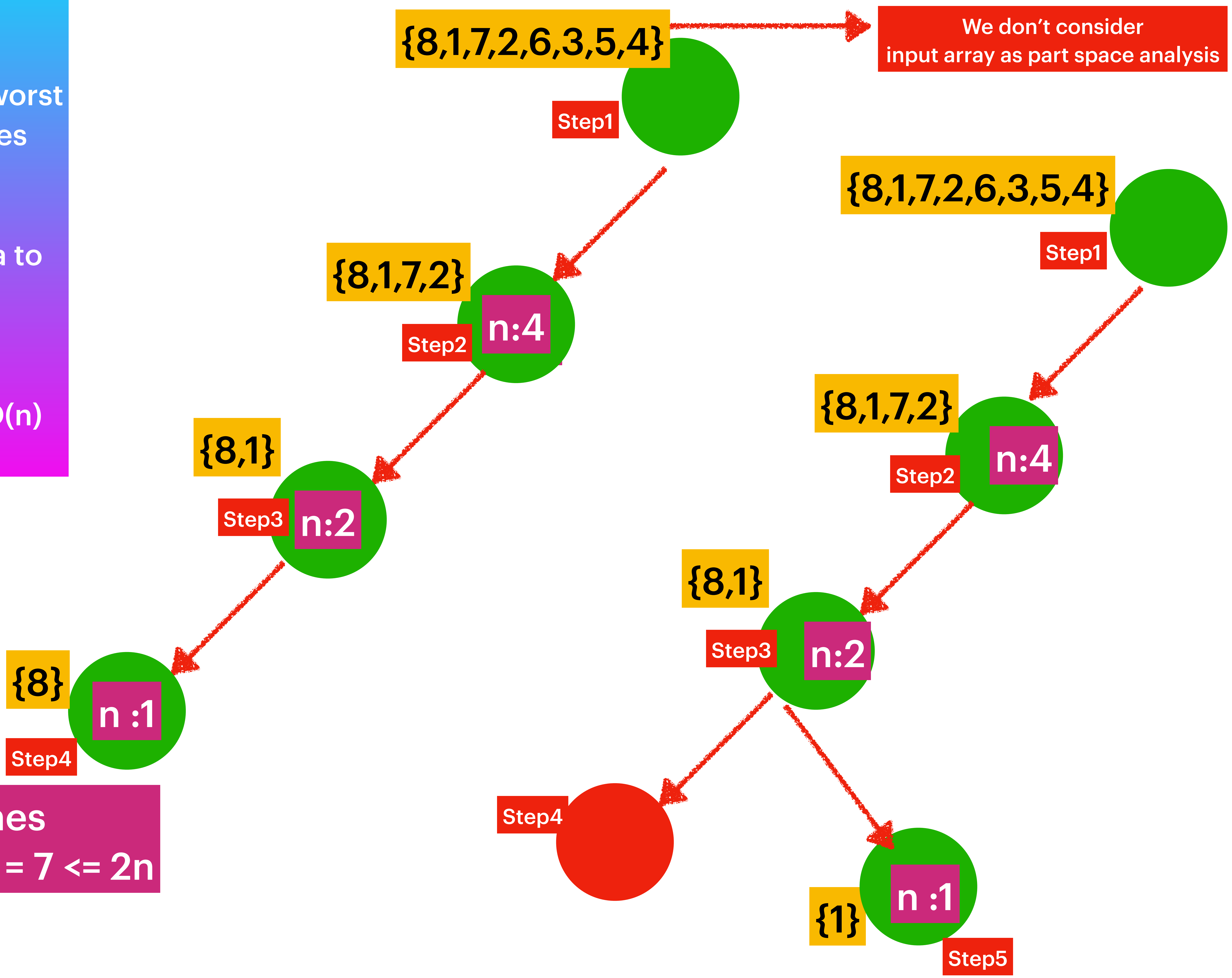
Time Complexity : nlog(n)

Space Complexity :

Which is little tricky, let's break it. In worst case we can have log(n) stack frames active.

The space occupies for copying data to subarrays in total log(n) active StackFrames is always <= 2n

Space Complexity = log(n) + O(n) = O(n)



In this active stack frames copied of array elements size = 7 <= 2n

In this active stack frames Observe by the time we reach to step5, step4 StackFrame was terminated so that total copied of array elements size again 7 <= 2n