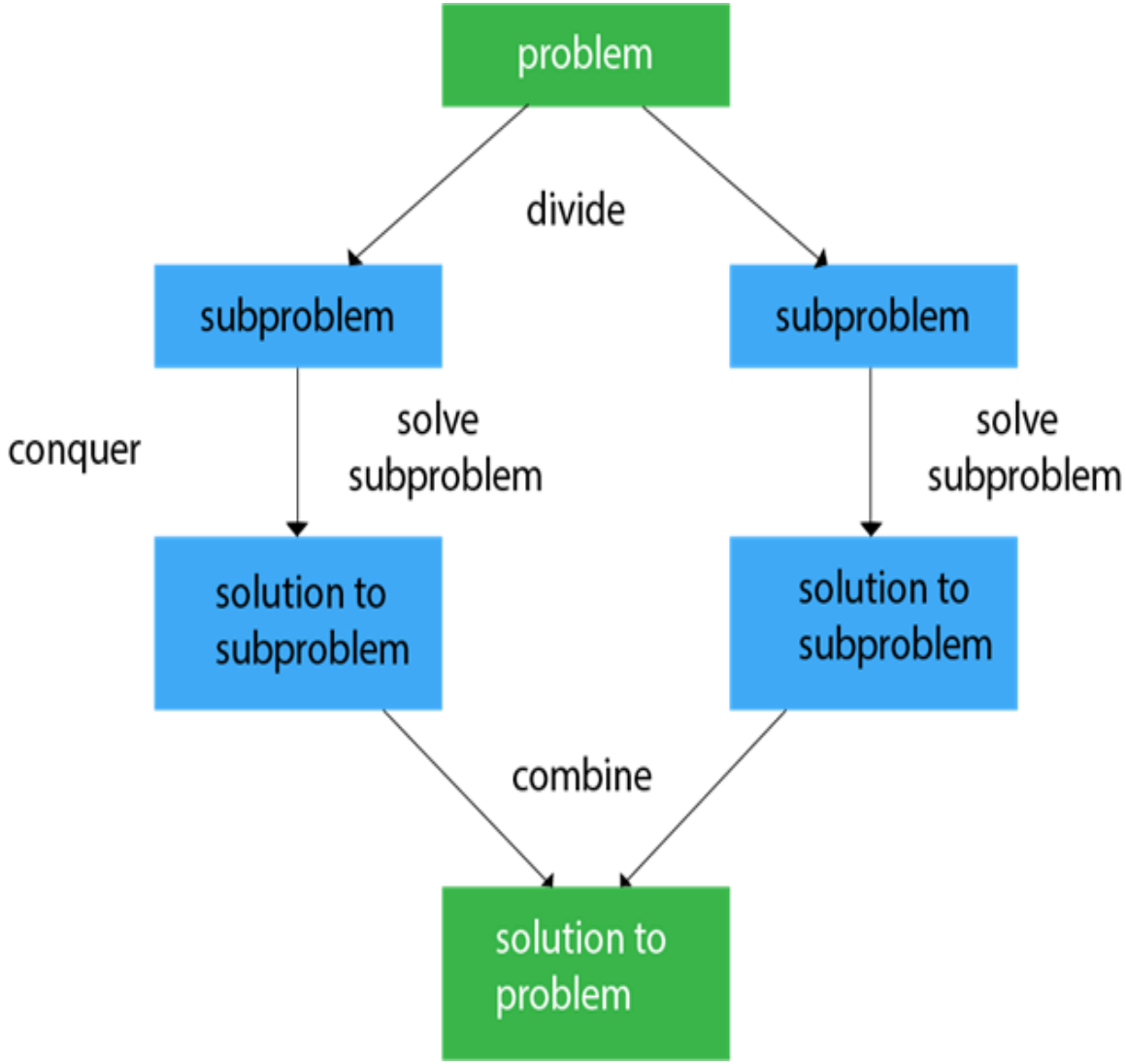
Divide And Conquer

This technique/Algorithm can be divided into the following three parts:

1. ***Divide:*** *This involves dividing the problem into smaller sub-problems.*

2. ***Conquer:*** *Solve sub-problems by calling recursively until solved.*

3. ***Combine:*** *Combine the sub-problems to get the final solution of the whole problem.*



Some algorithms that follow Divide and Conquer algorithm:

- ***Merge Sort*** *is a sorting algorithm. The algorithm divides the array into two halves, recursively sorts them, and finally merges the two sorted halves.*

- ***Quicksort*** *is a sorting algorithm. The algorithm picks a pivot element and rearranges the array elements so that all elements smaller than the picked pivot element move to the left side of the pivot, and all greater elements move to the right side. Finally, the algorithm recursively sorts the subarrays on the left and right of the pivot element.*

Advantage

1. *Solves difficult problems with less time complexity than its brute-force counterpart.*
2. *Since the sub-problems are independent, they can be computed in parallel.*

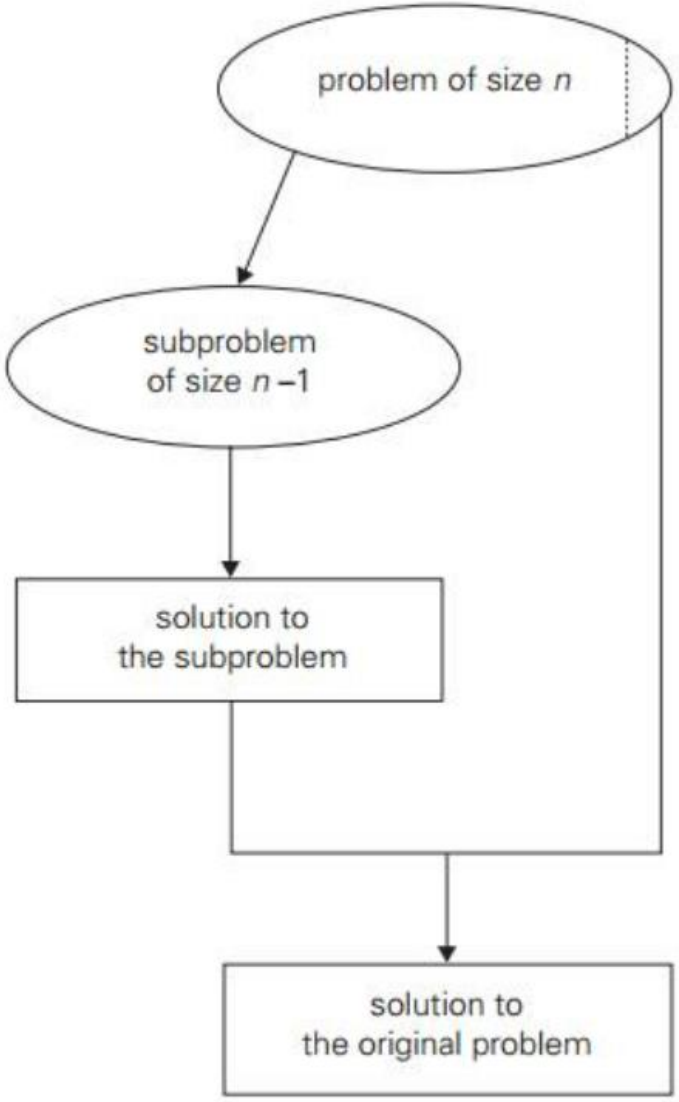
Disadvantage

1. *Includes recursion which consumes more space.*
2. *It may even crash the system if the recursion is performed rigorously greater than the stack present in the CPU.*

is binary search follows divide and conquer?

- *Binary Search is **not** a divide and conquer approach. It is a **decrease and conquer** approach.*

- *In divide and conquer approach, each subproblem must contribute to the solution but in binary search, all subdivision does not contribute to the solution.*
  - *We divide into two parts and discard one part because we know that the solution does not exist in this part and look for the solution only in one part.*

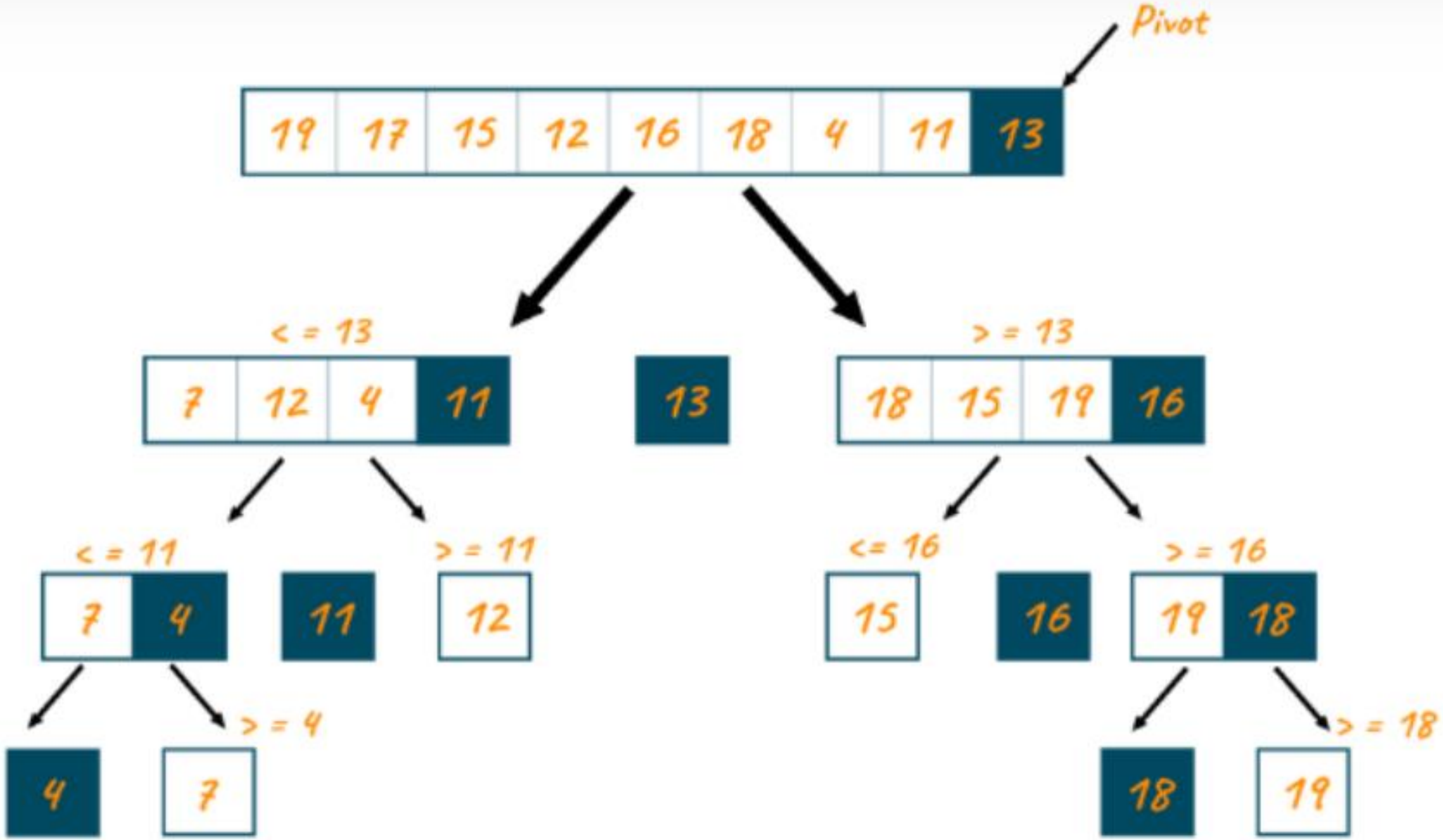


Some other decrease and conquer approach we solved:

- ***Euclidean algorithm** to compute the greatest common divisor of two numbers*
- ***Tower of Hanoi** puzzle, which reduces moving a tower of height n to moving a tower of height n-1.*

Quick Sort

- *is quick or fast in terms of speed*
- *based on the divide and conquer approach*
- *Here,*
  - *We select a pivot element, and we position the pivot in such a manner that all the elements smaller than the pivot element are to its left and all the elements greater than the pivot are to its right.*
  - *The elements to the left and right of the pivot form two separate arrays.*
  - *Now the left and right subarrays are sorted using this same above approach.*
  - *This process continues until each subarray consists of a single element. When this situation occurs, the array is now sorted, and we can merge the elements to get the required array.*
- ***How to select the pivot element?***
  - *We can select either one of the below 4 elements as pivot*
    - *First element*
    - *Last element*
    - *Any random element*
    - *Middle element*
  - *Let's consider last element as pivot*


Pivot
19 17 15 12 16 18 4 11 13
< = 13
7 12 4 11
13
> = 13
18 15 19 16
< = 11
7 4
11
> = 11
12
<= 16
15
16
> = 16
19 18
> = 4
4
7
18
19
> = 18

Quick Sort Algorithm

Step 1: Take pivot as the last element in the array

Step 2: Find the correct index of the pivot using the partition algorithm

Step 3: Recursively call Quicksort on the left subarray

Step 4: Recursively call Quicksort on the right subarray

Partition Algorithm

Step 1: Take two elements low and high

Step 2: Make low store the starting index of the array

Step 3: Make high store the last index of the array

Step 4: Take a variable j and initialize it to low

Step 5: Take pivot as the last element is the array

Step 6: Traverse through the entire array using the variable i and compare each element with pivot

Step 7: If arr[i] < pivot then increment j and swap element as position j and i

Step 8: Increment i by 1

Step 9: When the loop terminates, return j – 1

Let’s Code

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)

What kind of sorting algorithm is used in Arrays.sort(arr) or Collections.sort(list)?

Internally, JDK uses

- Dual Pivot Quick Sort for primitive arrays
- TimSort for Object[] arrays