Find your way here

CSC 250: Foundations of Computer Science I

Fall 2023 - Lecture 7

Amitabha Dey

Department of Computer Science
University of North Carolina at Greensboro

Divide and Conquer

A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

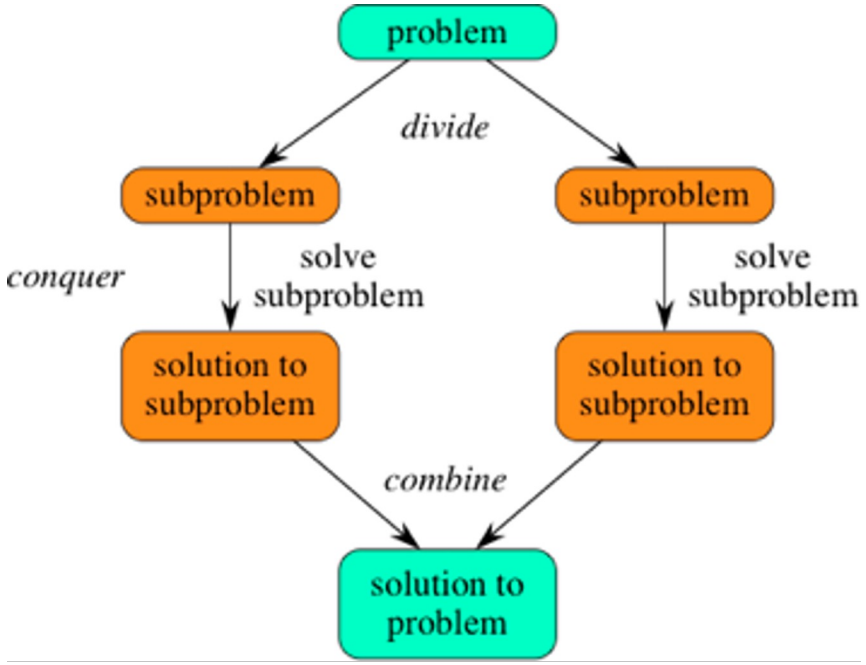


Image credit - https://www.khanacademy.org/

You should think of a divide-and-conquer algorithm as having three parts:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

The divide-and-conquer technique is the basis of efficient algorithms for many problems, such as **sorting** (e.g., quicksort, merge sort), **multiplying large numbers** (e.g., the Karatsuba algorithm), **finding the closest pair of points**, **syntactic analysis** (e.g., top-down parsers), and computing the **discrete Fourier transform** (FFT).

Divide and Conquer

Image credit - https://www.khanacademy.org/

You should think of a divide-and-conquer algorithm as having three parts:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

The divide-and-conquer technique is the basis of efficient algorithms for many problems, such as **sorting** (e.g., quicksort, merge sort), **multiplying large numbers** (e.g., the Karatsuba algorithm), **finding the closest pair of points**, **syntactic analysis** (e.g., top-down parsers), and computing the **discrete Fourier transform** (FFT).

Divide and Conquer
Implementation

Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

Combine
Combining sub-problem solutions to build the original problem solution

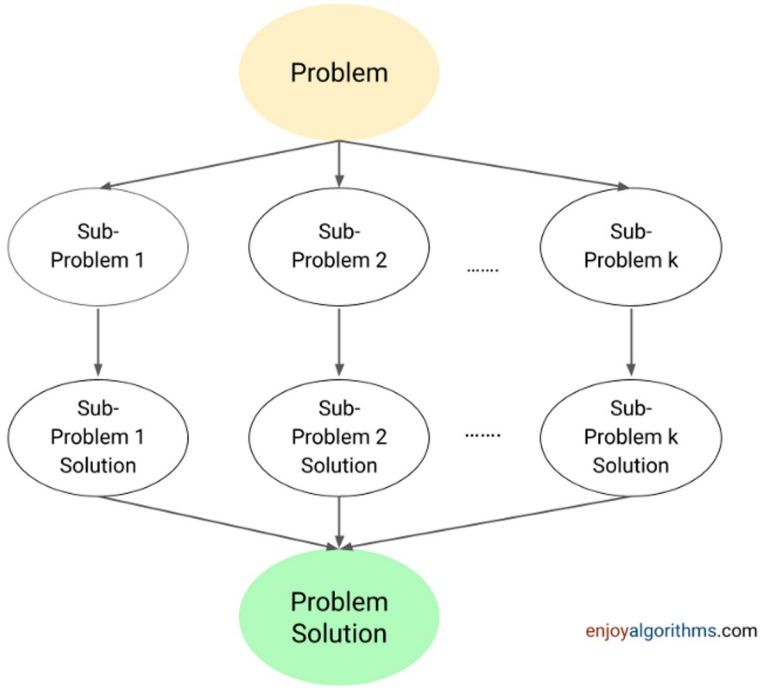


Image credit - https://www.enjoyalgorithms.com/

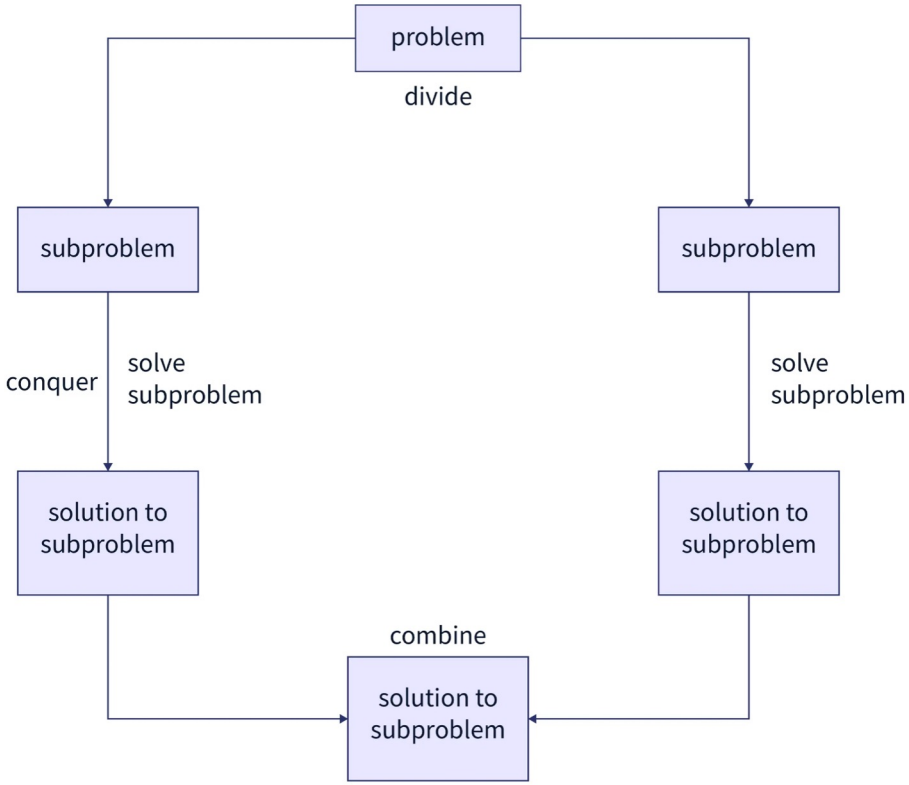


Image credit - https://www.scaler.com/

Divide and Conquer

DAC vs Dynamic Programming

- When dealing with subdivisions and conquer, divide and conquer are independent of one another, whereas dynamic programming is interdependent.
- A divide and conquer problem is twice as time-consuming as a dynamic programming one because each issue is independently addressed. With dynamic programming, the answers to previous questions are used.
- Dynamic programming is faster than divide and conquer because it is more efficient.
- Matrix chain multiplication and binary search tree optimization are performed via dynamic programming, whilst merge sort, quick sort, and binary searching, matrixes use divide and conquer.

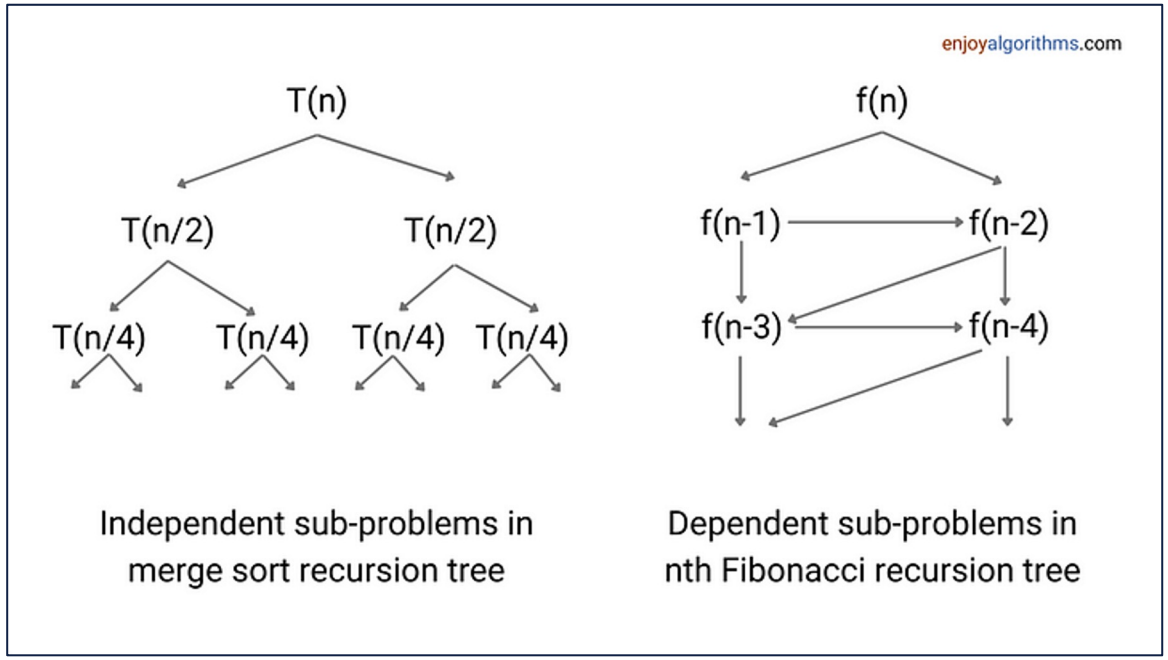


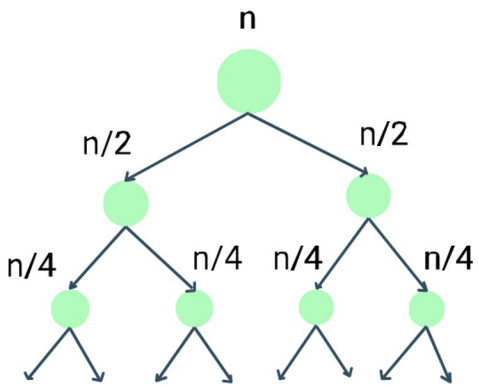
Image credit - https://www.enjoyalgorithms.com/

Divide and Conquer

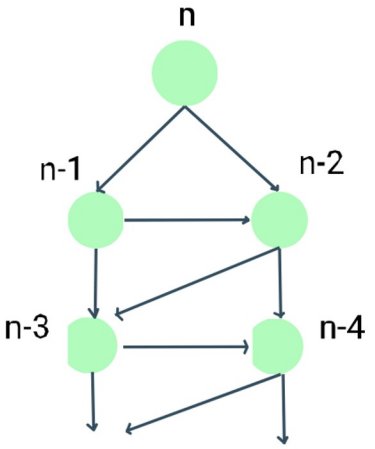
DAC vs Dynamic Programming

In the context of algorithms, **independent** means that the solution to one subproblem does not affect the solution to another subproblem. So, subproblems can be solved separately. An example of this is the idea of merge sort, where we divide the input array into two equal subarrays. Then, we recursively sort these two subproblems, treating each subarray as a separate problem.

On the other hand, **dependent** means that the solution to one subproblem relies on the solution to one or more smaller subproblems. So, the subproblems must be solved in a specific order. An example of this is the problem of finding the nth Fibonacci. For finding the nth Fibonacci, we use the solutions of the two subproblems: Finding the (n - 1)th and (n - 2)th Fibonacci. To calculate the (n - 1)th Fibonacci, we use the solution to the (n - 2)th and (n - 3)th Fibonacci. In other words, here subproblems are dependent because we can not solve nth Fibonacci and (n - 1)th Fibonacci until we find the solution of (n-2)th Fibonacci.



Divide and Conquer
(Independent Subproblems)



Dynamic Programming
(Dependent Subproblems)

Image credit - https://www.enjoyalgorithms.com/

Divide and Conquer

DAC vs Dynamic vs Greedy

Divide & Conquer	Dynamic Programming
1. Partitions a problem into independent smaller sub-problems	1. Partitions a problem into overlapping sub-problems
2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise - results in the same computations are performed repeatedly.)	2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice
3. Top down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances.	3. Bottom up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances

Image credit - https://en.differbetween.com/

Greedy	Divide and Conquer
Used when need to find optimal solution	No optimal solution, used when problem have only one solution
Does not work parallel	work parallel by dividing big problem in smaller sub problem and running then parallel
Example: Knapsack, Activity selection	Example: Sorting, Searching

Image credit - https://slidetodoc.com/

Greedy vs Divide & Conquer vs Dynamic Programming

Greedy	Divide & Conquer	Dynamic Programming
Optimises by making the best choice at the moment.	Optimises by breaking down a subproblem into simpler versions of itself and using multi-threading & recursion to solve.	Same as Divide and Conquer, but optimises by caching the answers to each subproblem as not to repeat the calculation twice.
Doesn't always find the optimal solution, but is very fast.	Always finds the optimal solution, but is slower than Greedy.	Always finds the optimal solution, but may be pointless on small datasets.
Requires almost no memory.	Requires some memory to remember recursive calls.	Requires a lot of memory for memoisation / tabulation

Image credit - https://dev.to/

Divide and Conquer
Max Element in an Array

- **Divide**: Divide array into two halves.
- **Conquer**: Recursively find maximum and minimum of both halves.
- **Combine**: Compare maximum of both halves to get overall maximum and compare minimum of both halves to get overall minimum

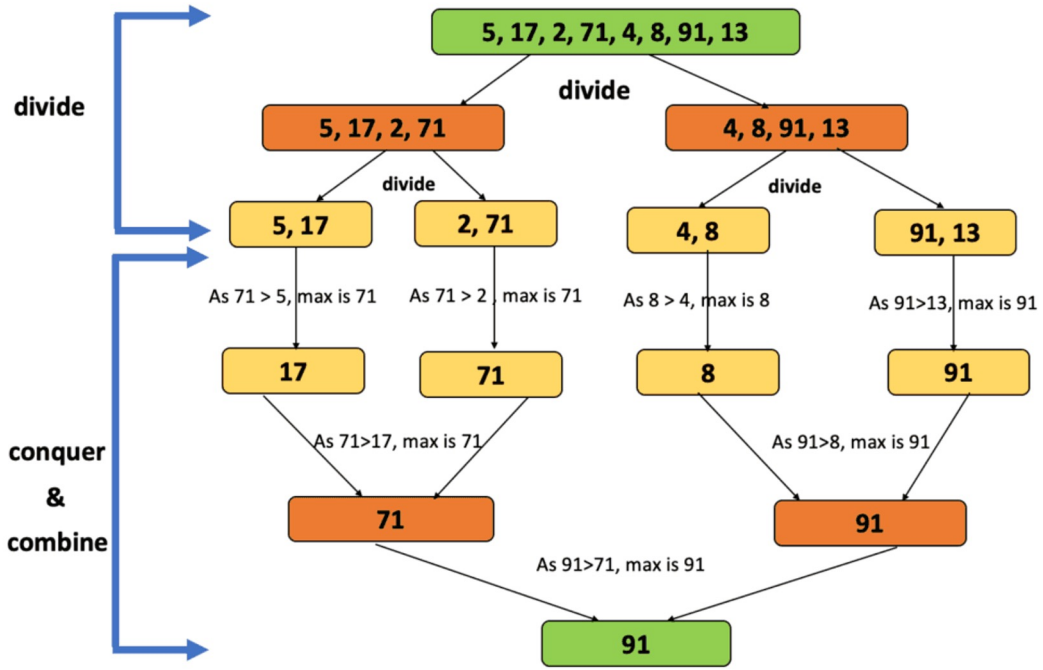


Image credit - https://technologystrive.com/

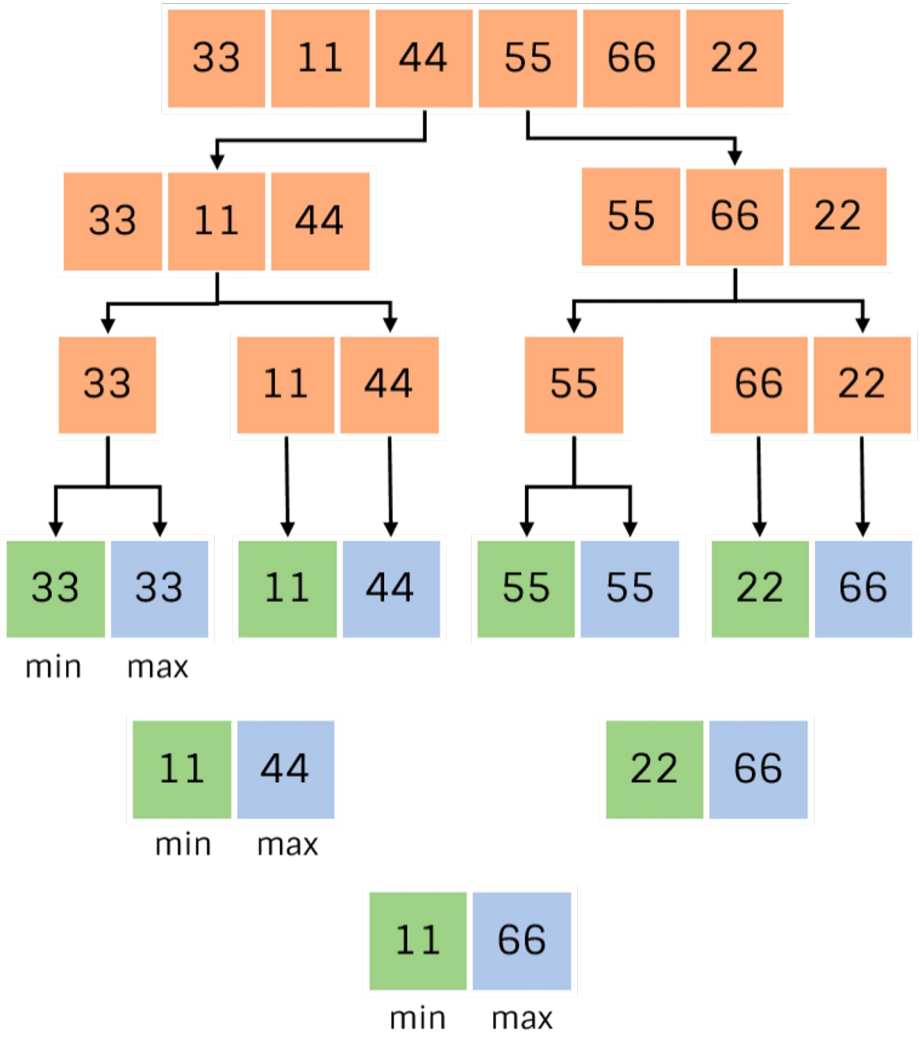


Image credit - https://codecrucks.com/

Divide and Conquer

Merge Sort Algorithm

Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

Think of it as a recursive algorithm continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves.

Finally, when both halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

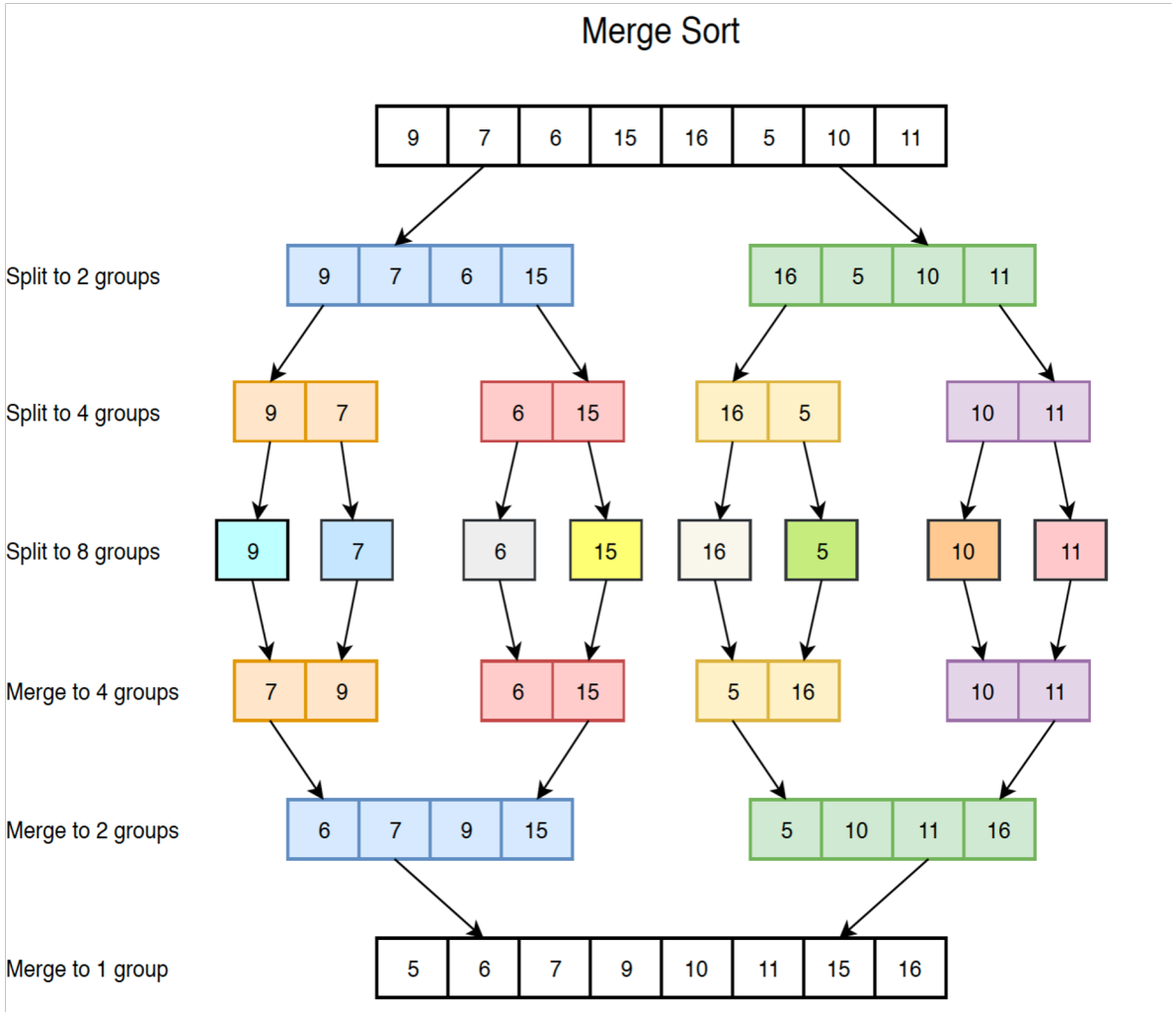
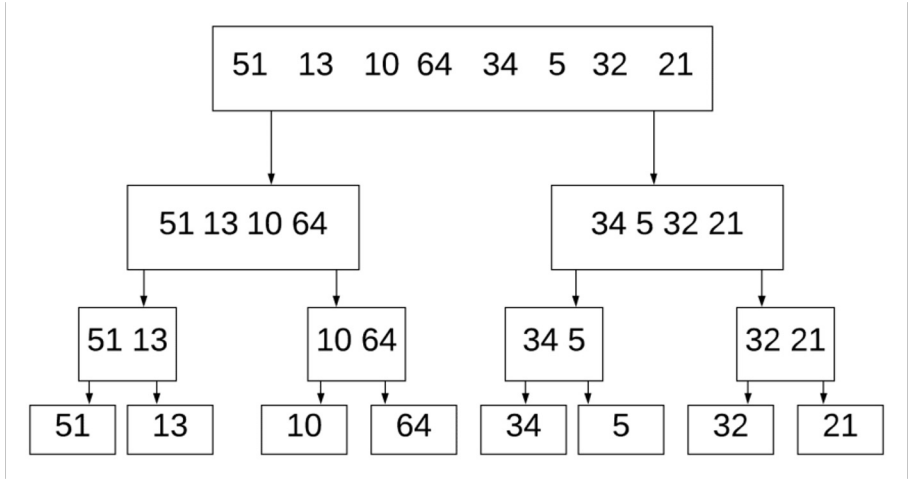


Image credit - https://jojozhuang.github.io/

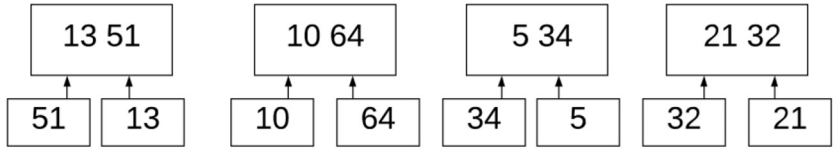
Divide and Conquer
Merge Sort Algorithm

The algorithm works as follows:

- Divide the sequence of n numbers into 2 halves
- Recursively sort the two halves
- Merge the two sorted halves into a single sorted sequence



We break down the 8 numbers into separate digits. It compares 51 and 13. Since 13 is smaller, it puts it in the left-hand side. It does this for (10, 64), (34, 5), (32, 21).



It then merges (13, 51) with (10, 64). It knows that 13 is the smallest in the first list, and 10 is the smallest in the right list. 10 is smaller than 13, therefore we don't need to compare 13 to 64. We're comparing & merging two sorted lists.

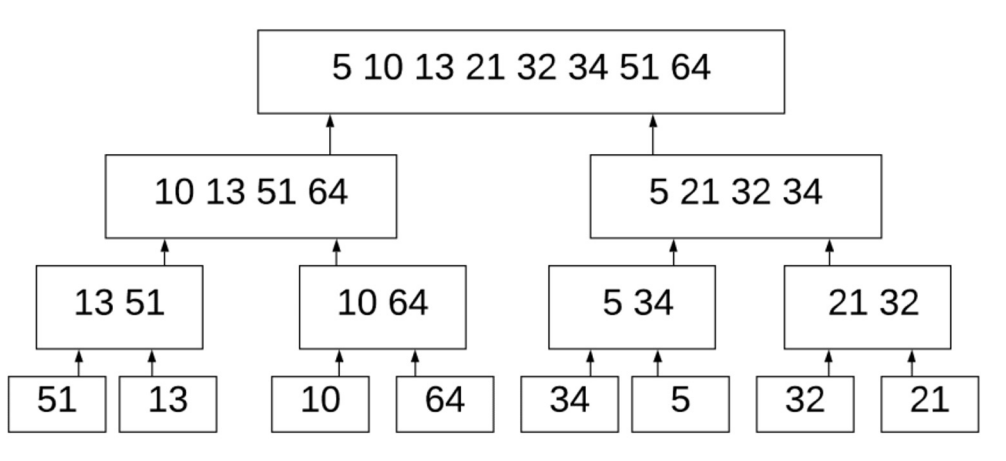


Image credit - https://dev.to/

Divide and Conquer

Quick Sort Algorithm

Quicksort uses divide-and-conquer, and so it's a recursive algorithm.

The way that quicksort uses divide-and-conquer is a little different from how merge sort does.

In merge sort, the divide step does hardly anything, and all the real work happens in the combine step.

Quicksort is the opposite: all the real work happens in the divide step. In fact, the combine step in quicksort does absolutely nothing.

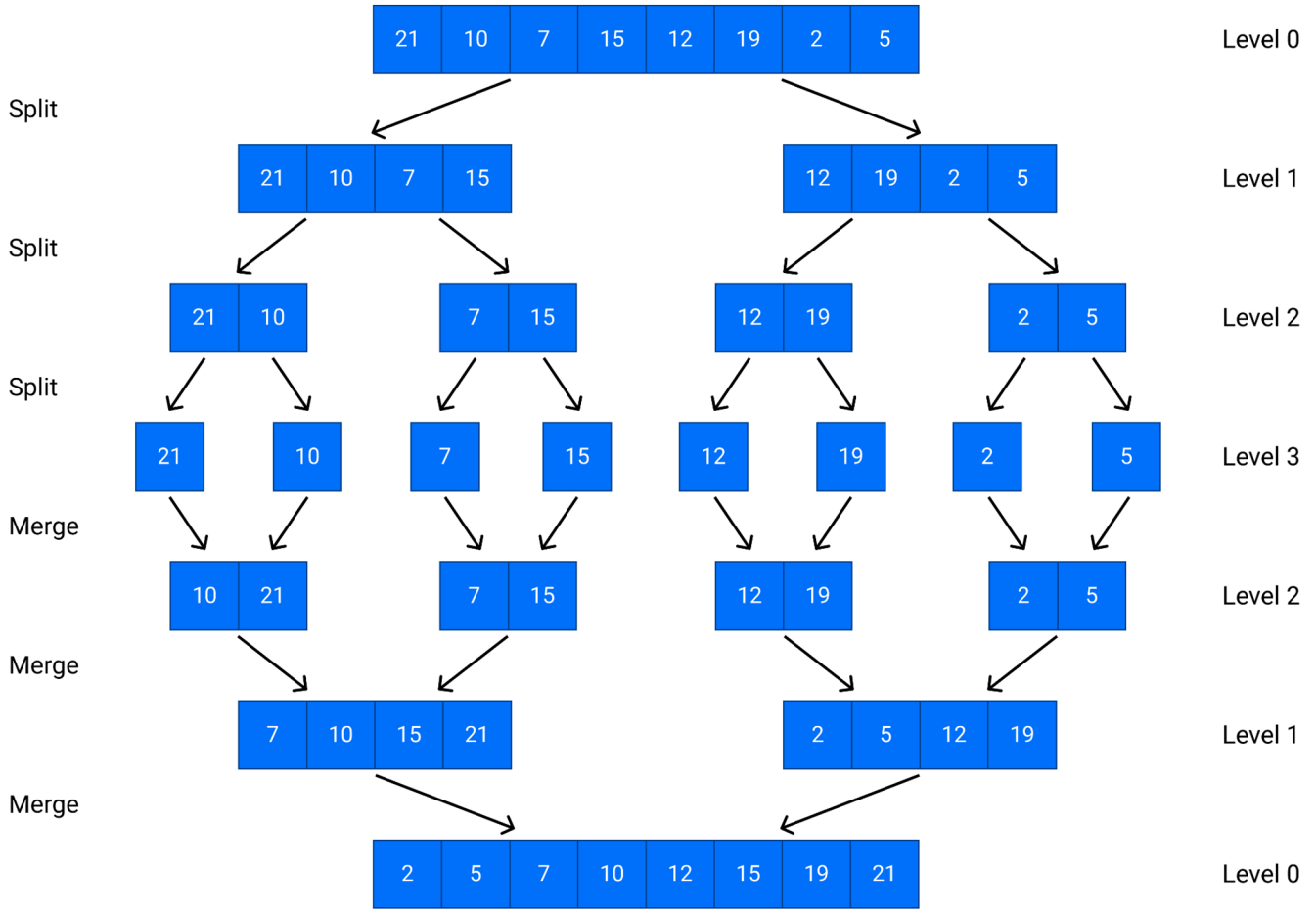


Image credit - https://dev.to/

Divide and Conquer
Quick Sort Algorithm

1. Select the Pivot Element

There are different variations of quicksort where the pivot element is selected from different positions. Here, we will be selecting the rightmost element of the array as the pivot element.

Select a pivot element

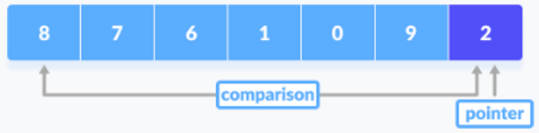
2. Rearrange the Array

Now the elements of the array are rearranged so that elements that are smaller than the pivot are put on the left and the elements greater than the pivot are put on the right.

Put all the smaller elements on the left and greater on the right of pivot element

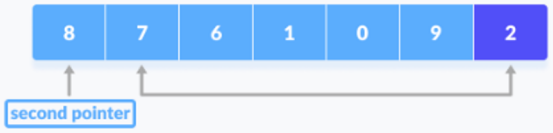
Here's how we rearrange the array:

1. A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index.



Comparison of pivot element with element beginning from the first index

2. If the element is greater than the pivot element, a second pointer is set for that element.



If the element is greater than the pivot element, a second pointer is set for that element.

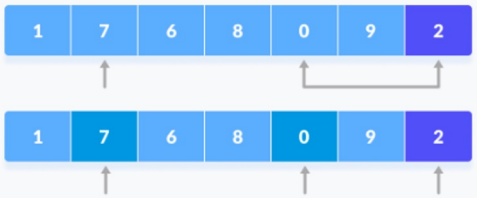
Image credit - https://www.programiz.com/

Divide and Conquer
Quick Sort Algorithm

3. Now, pivot is compared with other elements. If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.

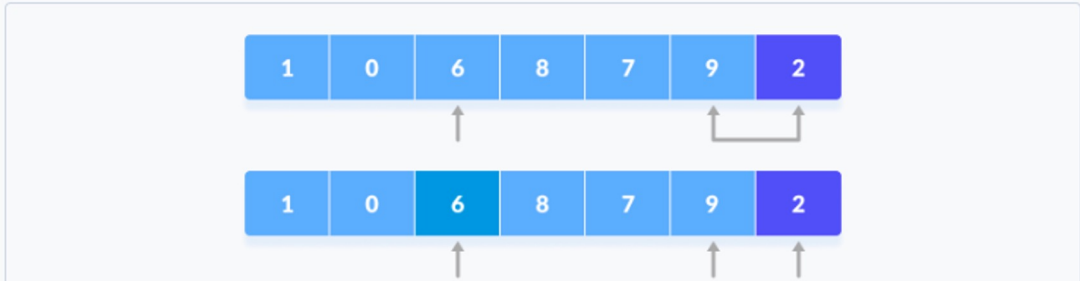
Pivot is compared with other elements.

4. Again, the process is repeated to set the next greater element as the second pointer. And, swap it with another smaller element.



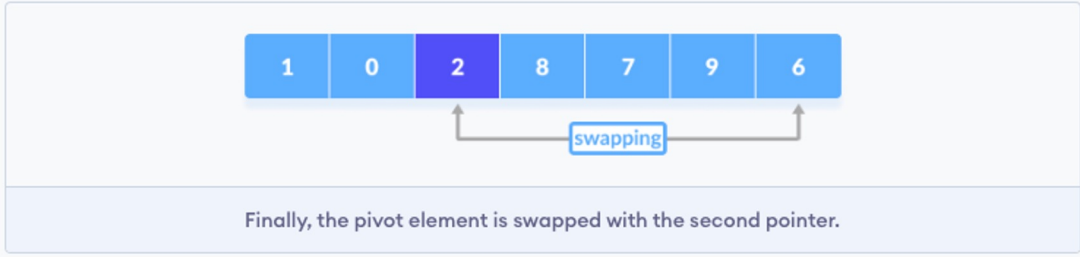
The process is repeated to set the next greater element as the second pointer.

5. The process goes on until the second last element is reached.



The process goes on until the second last element is reached.

6. Finally, the pivot element is swapped with the second pointer.



Finally, the pivot element is swapped with the second pointer.

Image credit - https://www.programiz.com/

Divide and Conquer
Quick Sort Algorithm

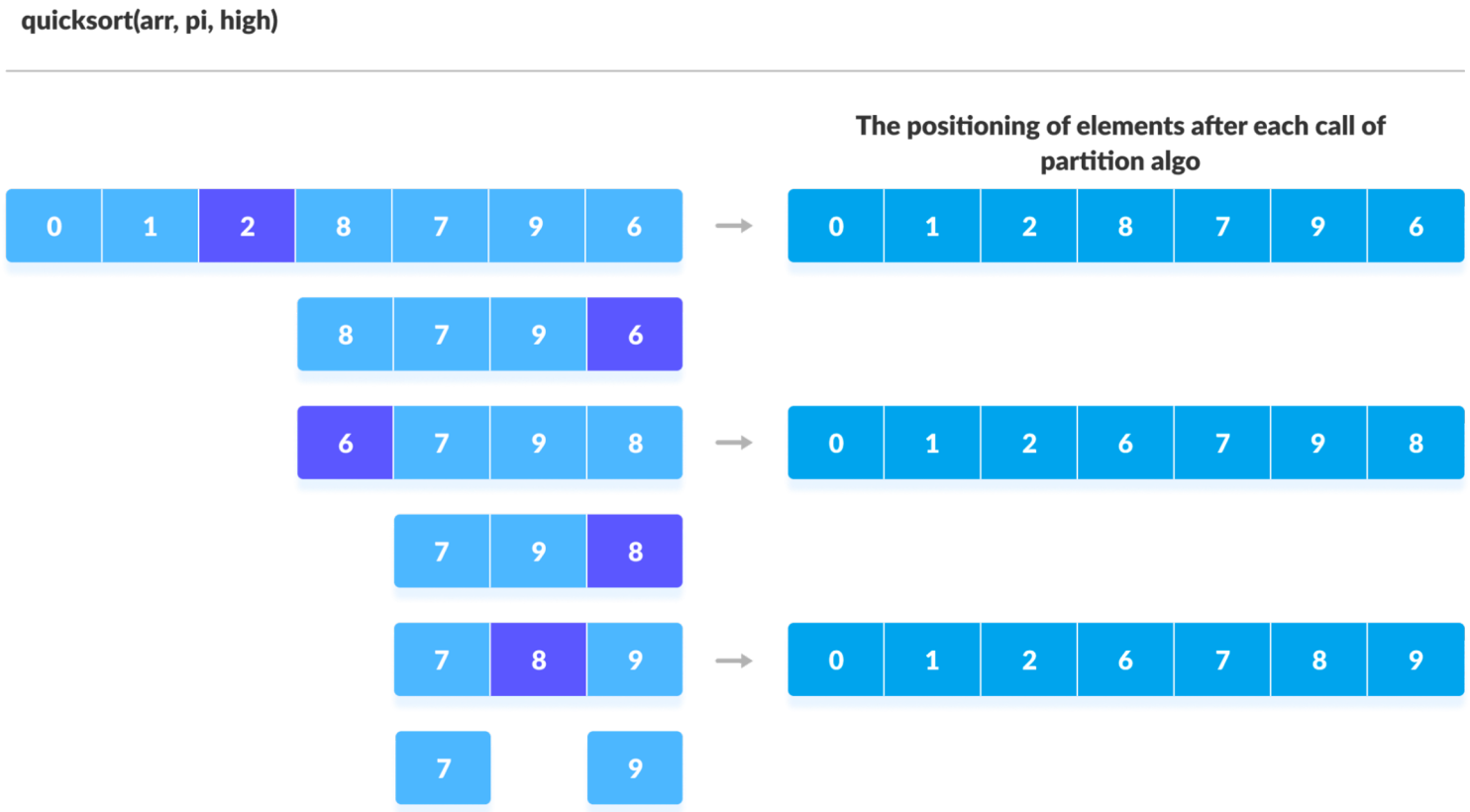


Image credit - https://www.programiz.com/

Divide and Conquer
Binary Search Algorithm

Divide Phase

In the divide phase of Binary Search, the algorithm splits the search space in half. It calculates the middle index of the array and compares the target element with the element at the middle index. This comparison determines which half of the array is more likely to contain the target element.

If the target element is equal to the middle element, the search is successful, and the algorithm terminates. Otherwise, if the target element is less than the middle element, the algorithm narrows down the search space to the left half of the array. Similarly, if the target element is greater than the middle element, the algorithm focuses on the right half of the array.

The divide phase involves dividing the problem into smaller subproblems by selecting the appropriate half of the array. This step plays a crucial role in reducing the search space and optimizing the search process.

Conquer Phase

In the conquer phase of Binary Search, the algorithm recursively applies the divide and conquer paradigm to the selected half of the array. It continues the search process on the reduced search space until the target element is found or the search space is exhausted.

By repeatedly dividing the search space and applying the divide and conquer approach, Binary Search efficiently homes in on the target element. The conquer phase involves solving the subproblems independently and combining their solutions to obtain the final result, which in this case is finding the target element.

Divide and Conquer
Binary Search Algorithm

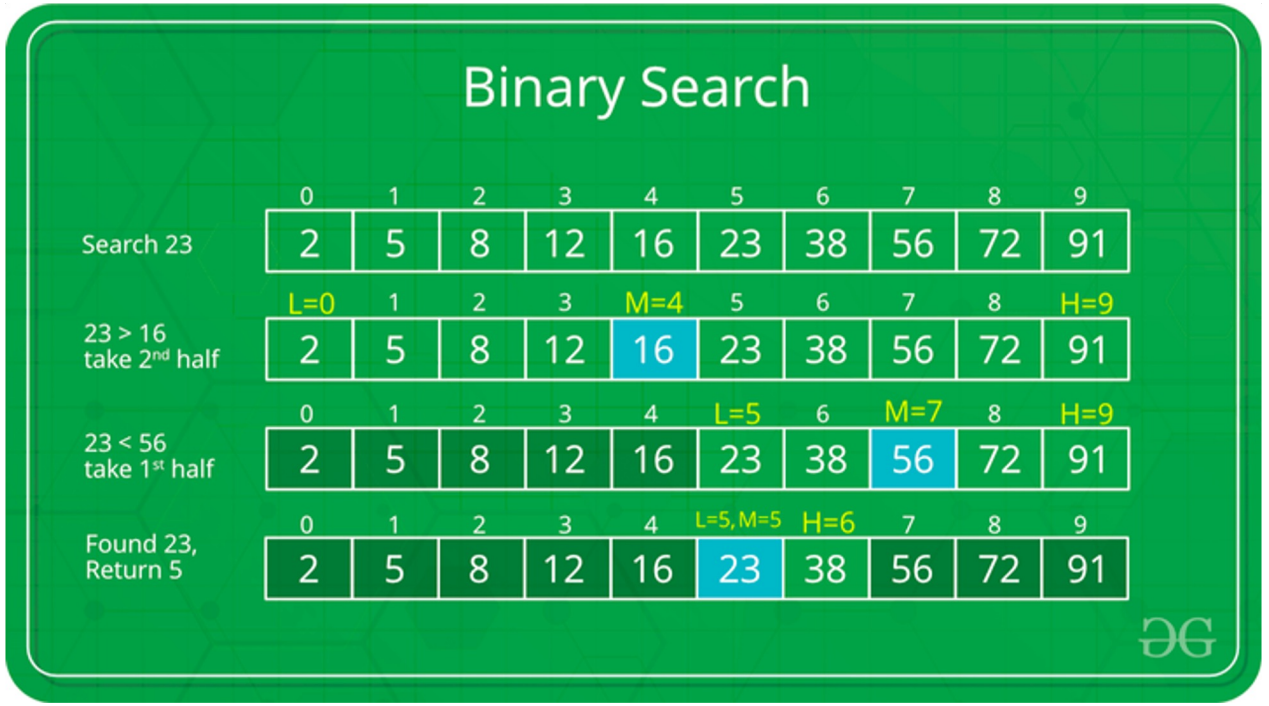


Image credit - https://www.geeksforgeeks.org/

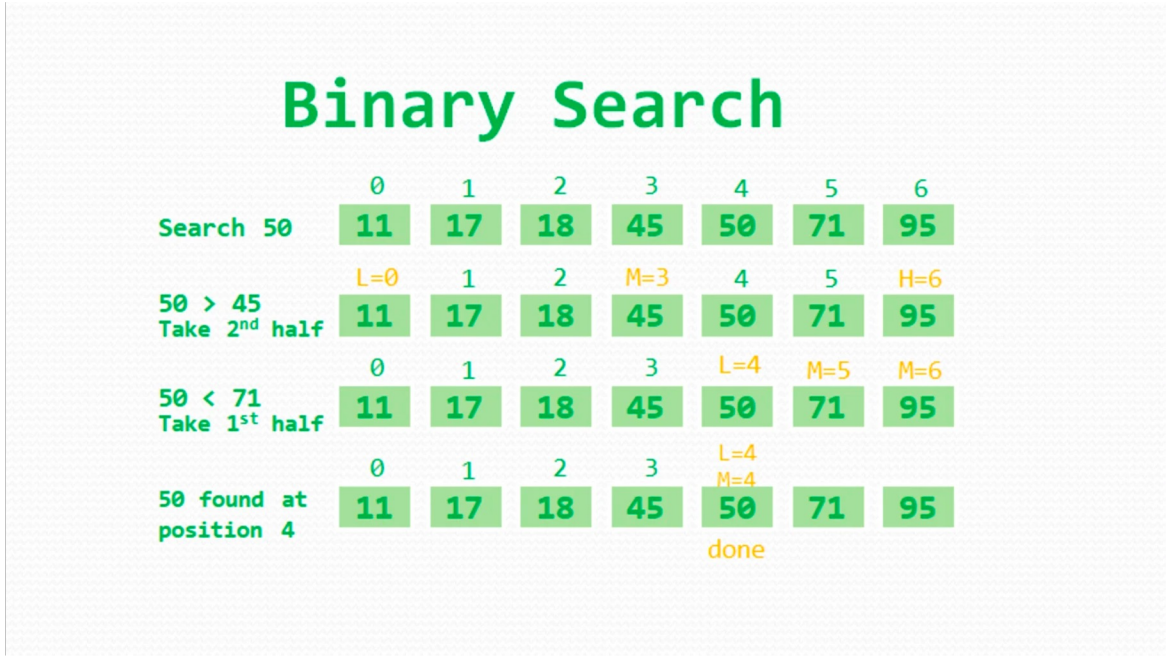


Image credit - https://dev.to/

Divide and Conquer

Towers of Hanoi

The Towers of Hanoi problem consists of three vertical rods, or towers, and *N* disks of different sizes, each with a hole in the center so that the rod can slide through it. The disks are originally stacked on one of the towers in order of descending size (i.e., the largest disc is on the bottom). The goal of the problem is to move all the disks to a different rod while complying with the following three rules:

1. Only one disk can be moved at a time.
2. Only the disk at the top of a stack may be moved.
3. A disk may not be placed on top of a smaller disk.

The following figure shows the steps of the solution to the Tower of Hanoi problem with three disks.

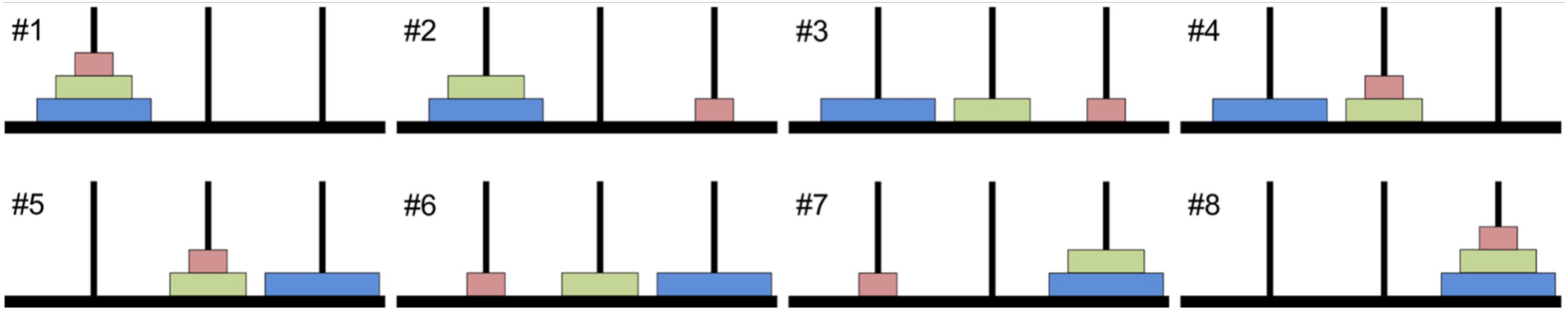


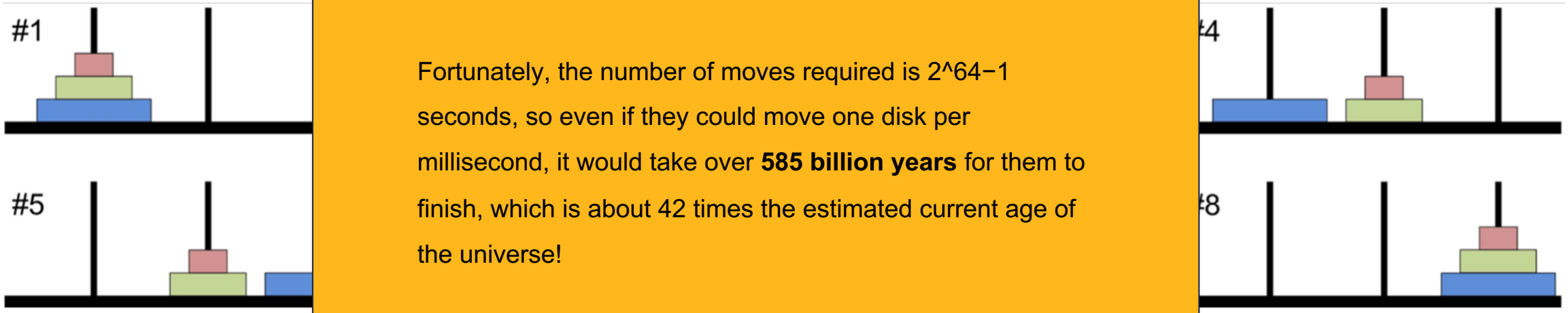
Image credit - https://pythonnumericalmethods.berkeley.edu/

Divide and Conquer
Towers of Hanoi

The Towers of Hanoi problem
ole in the center so that the rod can
slide through it. The disks are
isc is on the bottom). The goal of the
problem is to move all the disk

1. Only one disk can be m
2. Only the disk at the top
3. A disk may not be place

The following figure shows the

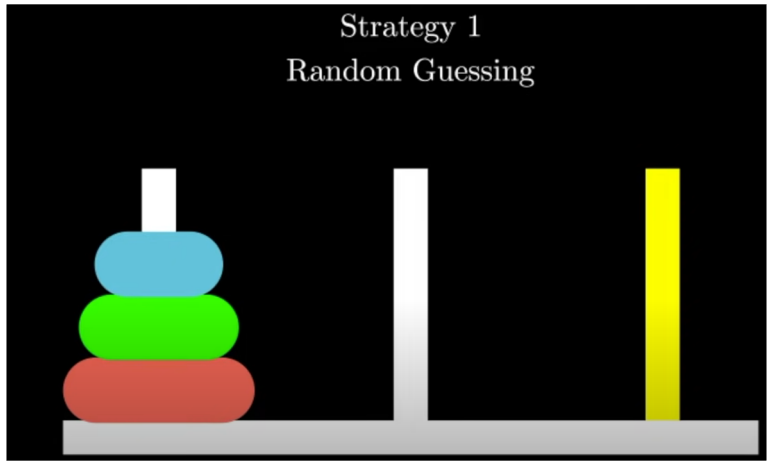


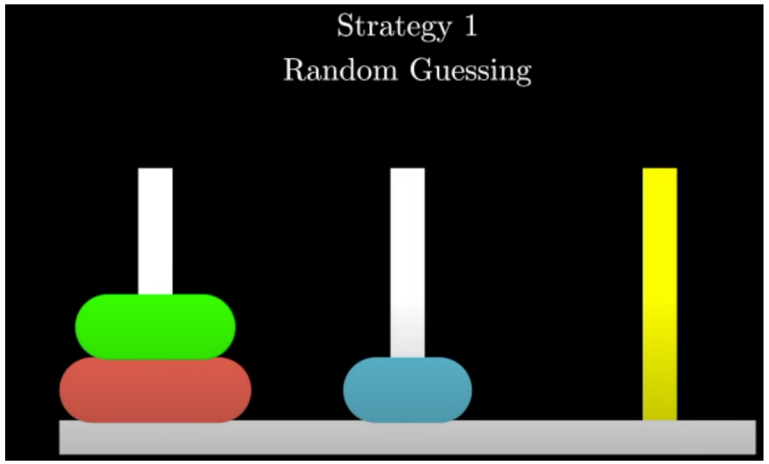
A legend goes that a group of Indian monks are in a monastery working to complete a Towers of Hanoi problem with 64 disks. When they complete the problem, the world will end.

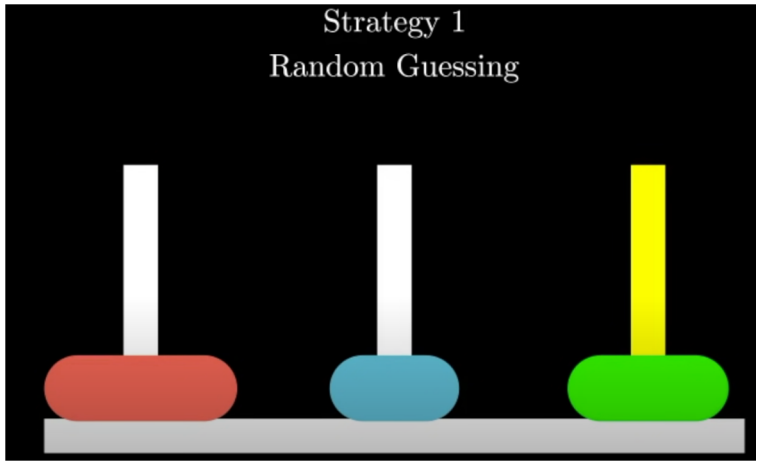
Fortunately, the number of moves required is 2^64−1 seconds, so even if they could move one disk per millisecond, it would take over **585 billion years** for them to finish, which is about 42 times the estimated current age of the universe!

Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi







We realize we are not getting anywhere. This is what we really want

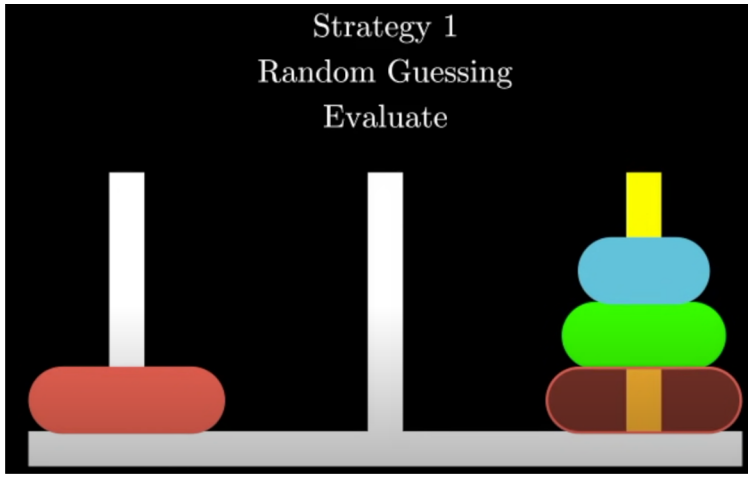


Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi

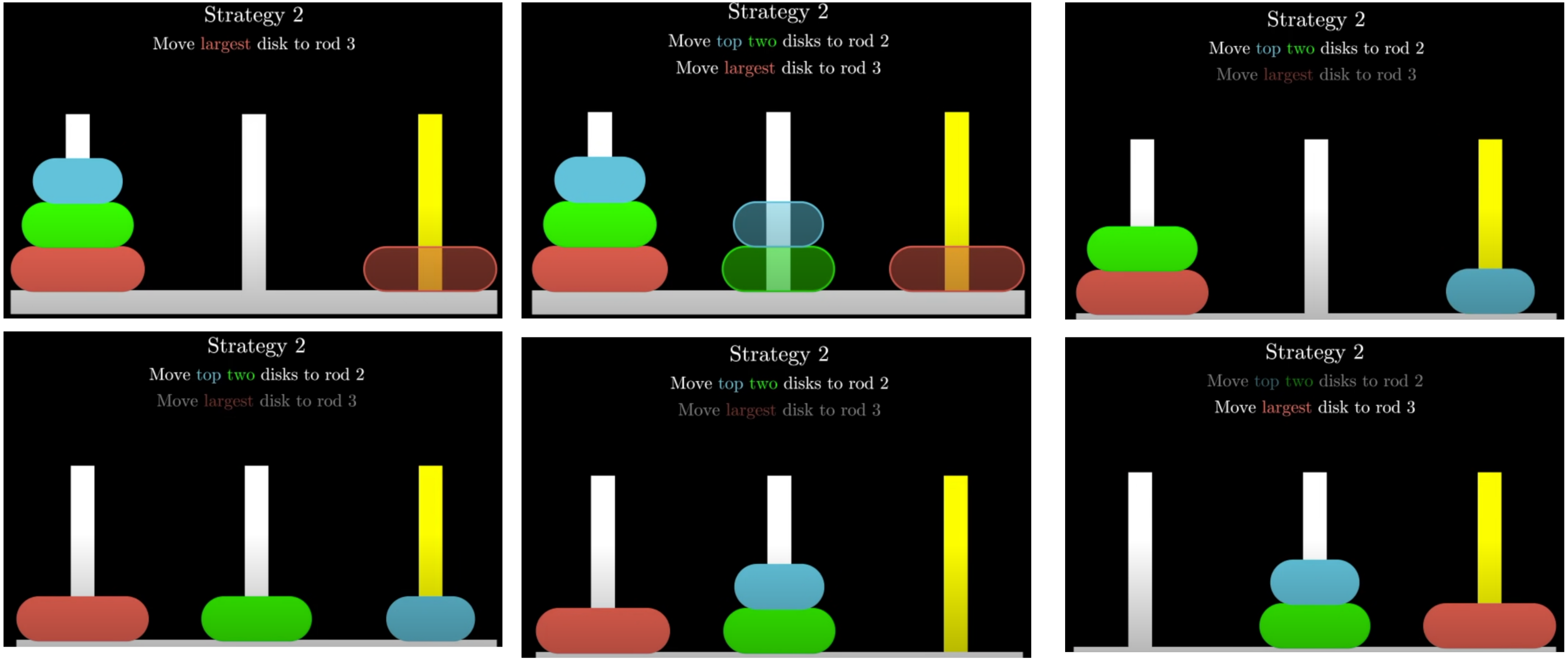
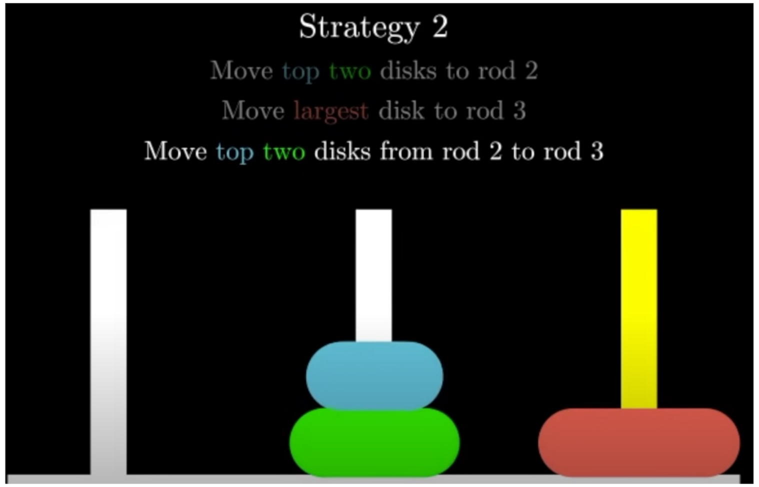
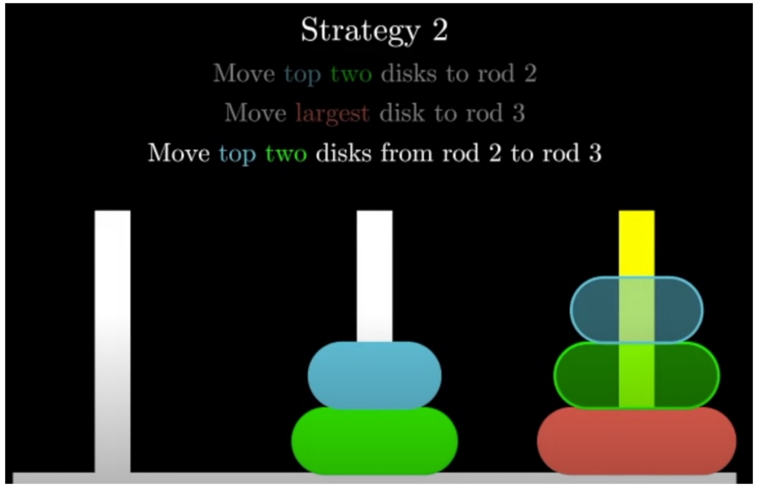
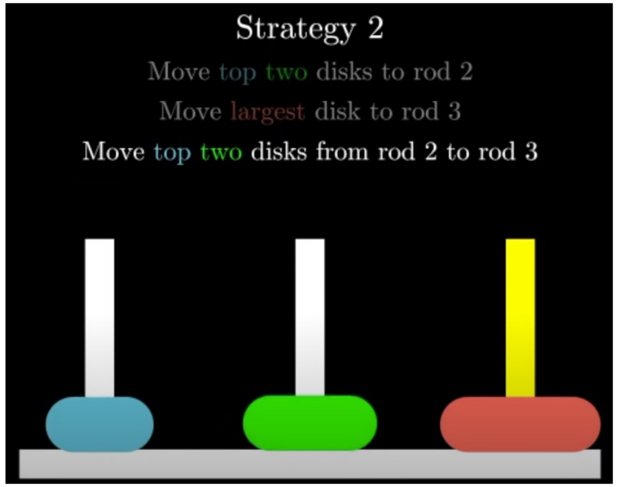


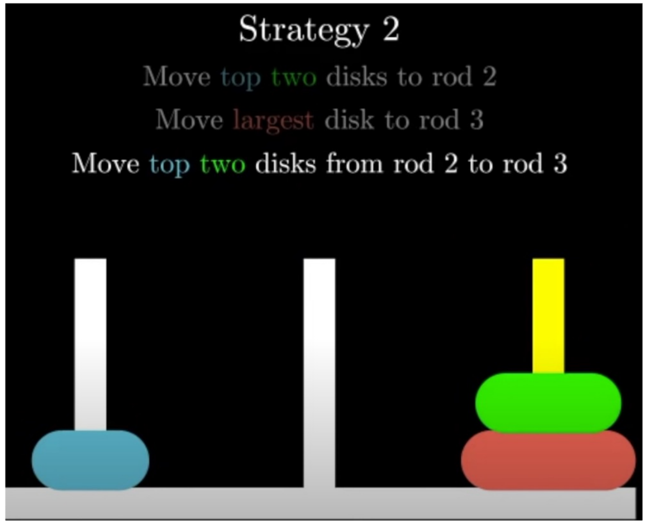
Image credit - Reducible on YouTube

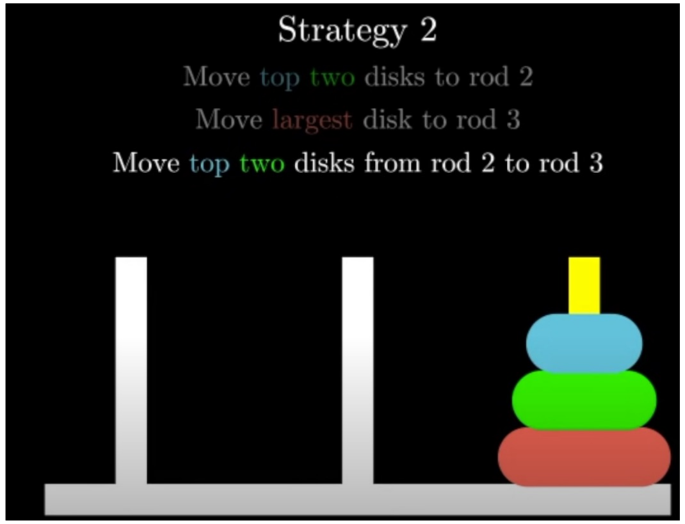
Divide and Conquer
Towers of Hanoi









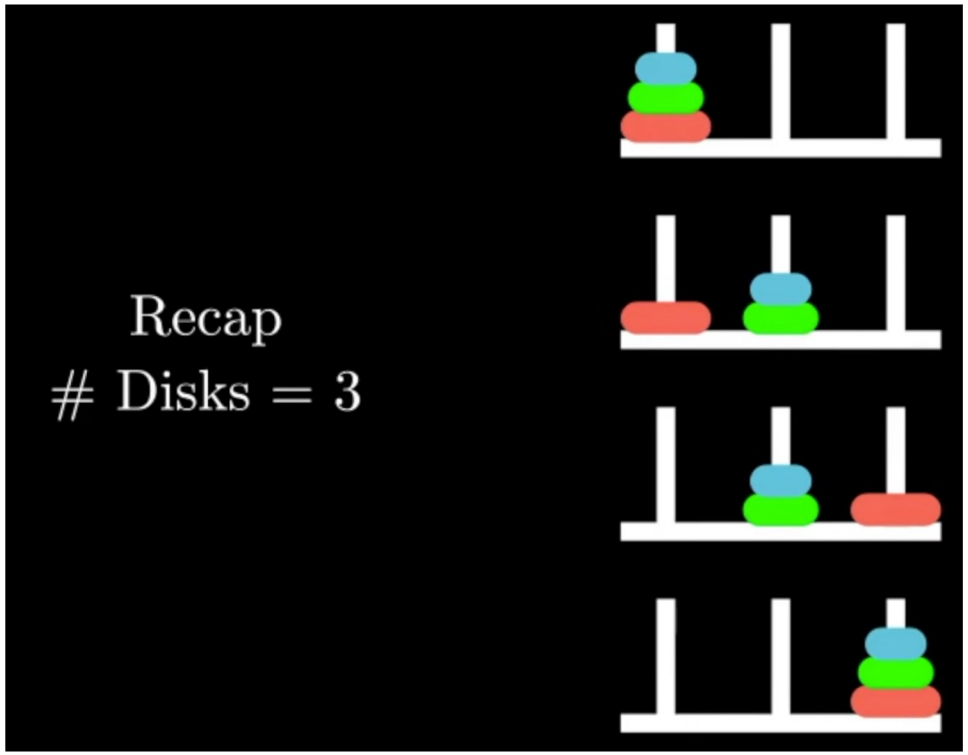


We just solved Towers of Hanoi for 3 discs! 🎉

Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi

- We first found a way to move the top 2 discs to the second rod
- Allowing us to move the largest disc to the destination rod
- Finally, we move the top 2 discs from the middle rod to the destination rod



Can we utilize the same strategy to move 4 discs?

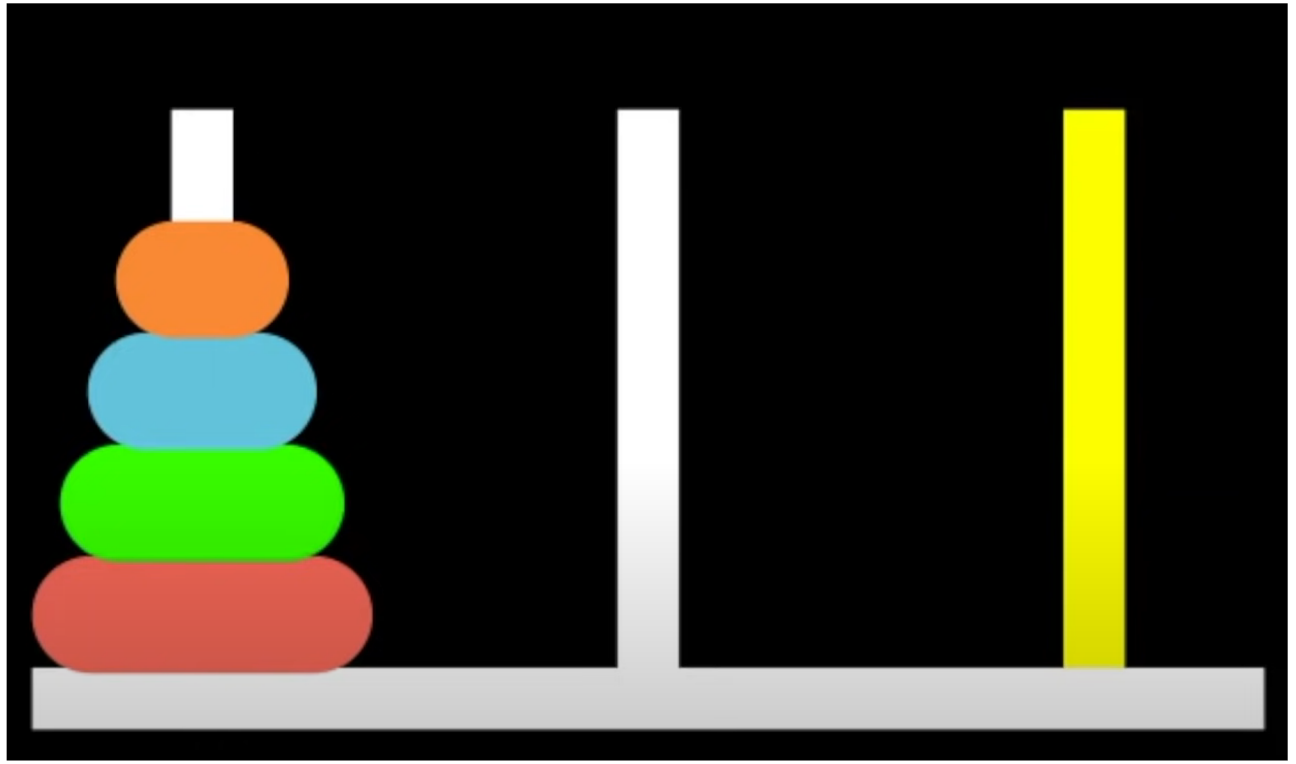
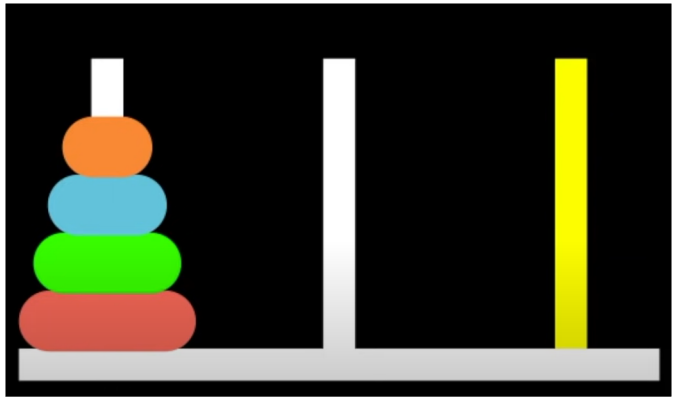
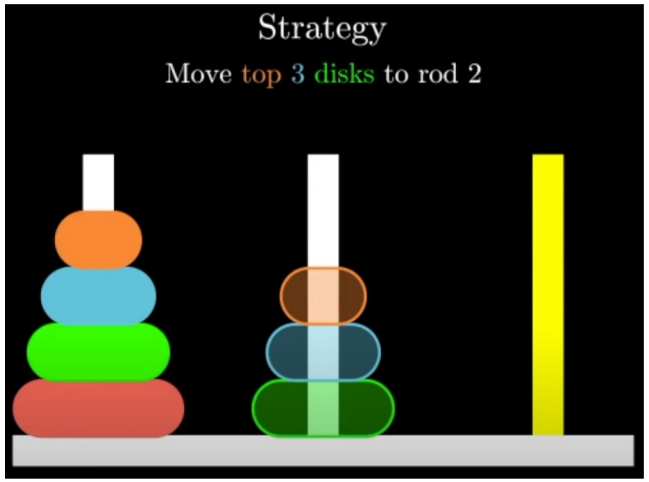


Image credit - Reducible on YouTube

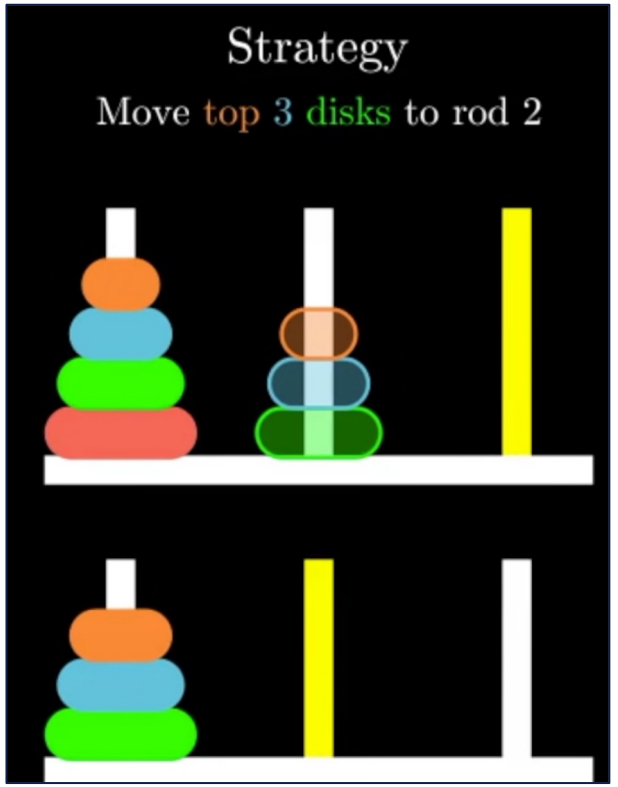
Divide and Conquer

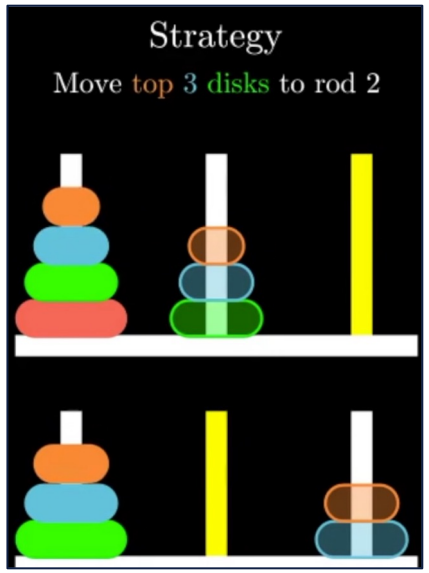
Towers of Hanoi

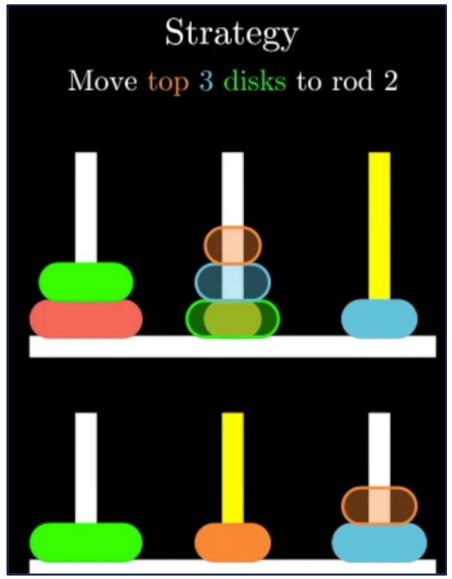


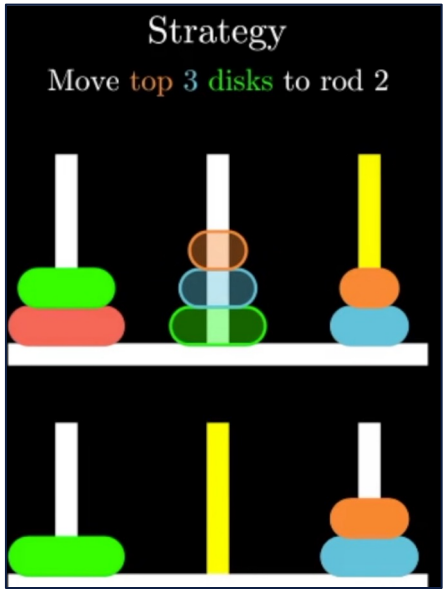


You can think of this problem as a 3 disc Tower of Hanoi problem and consider the middle rod as the destination rod.









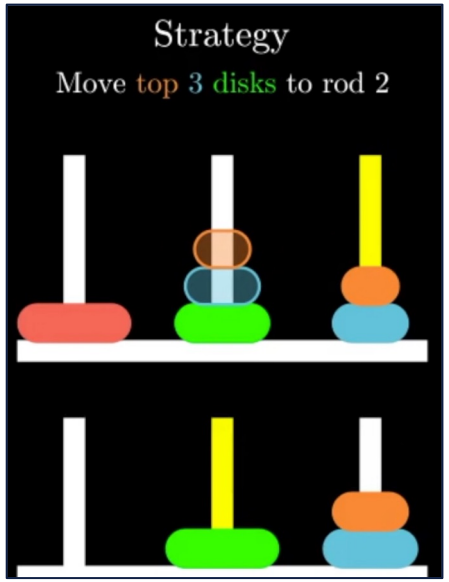
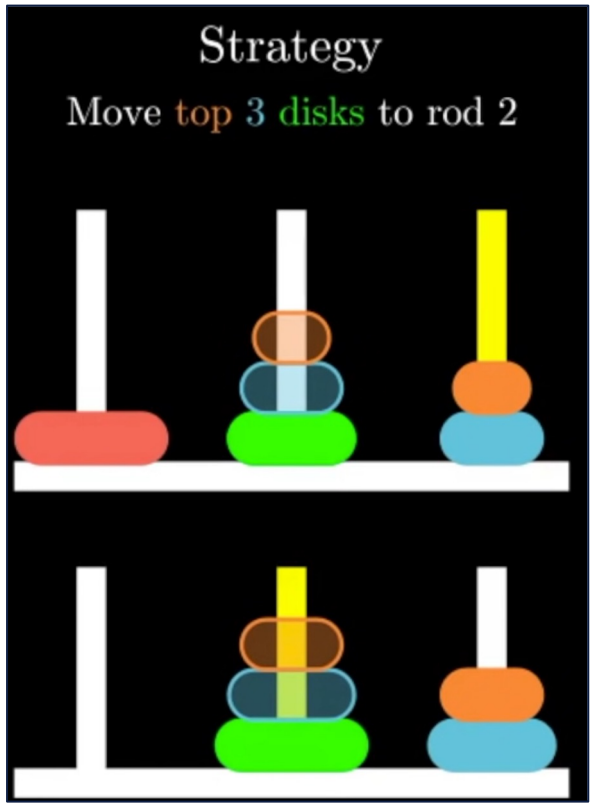
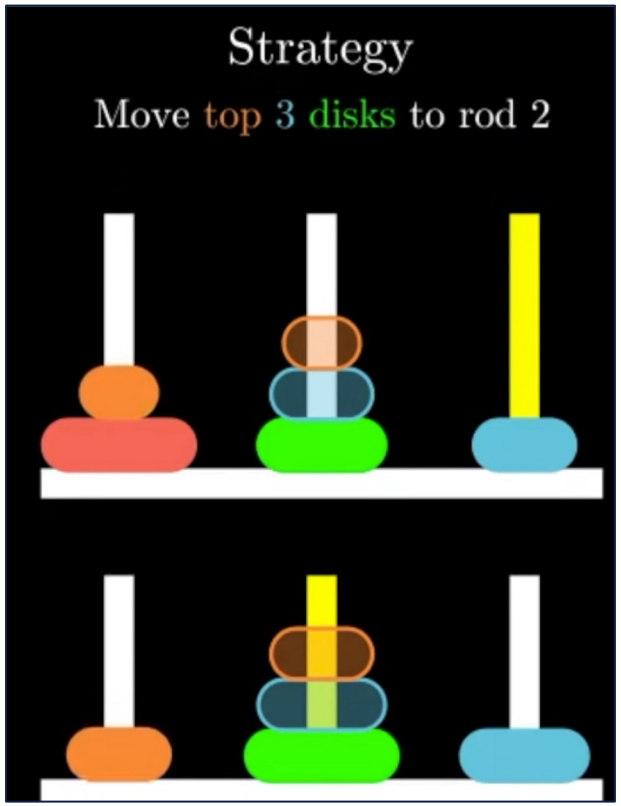
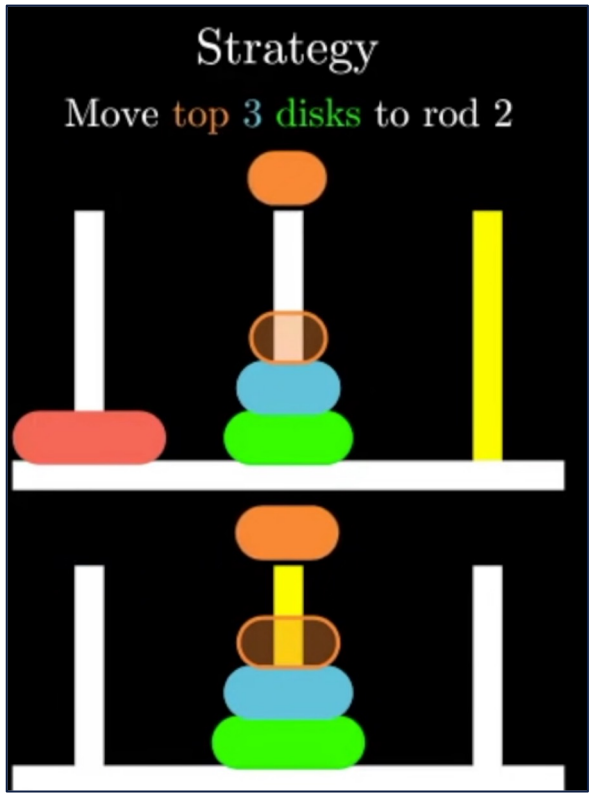


Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi







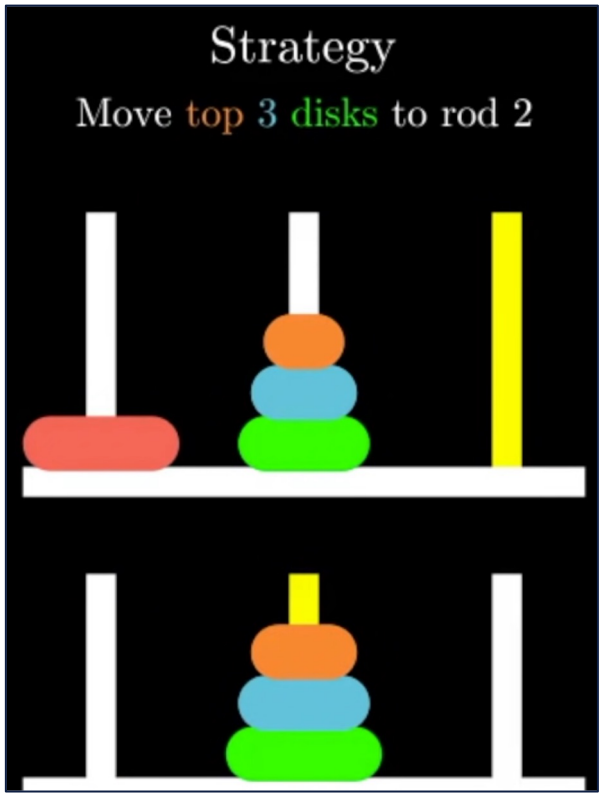
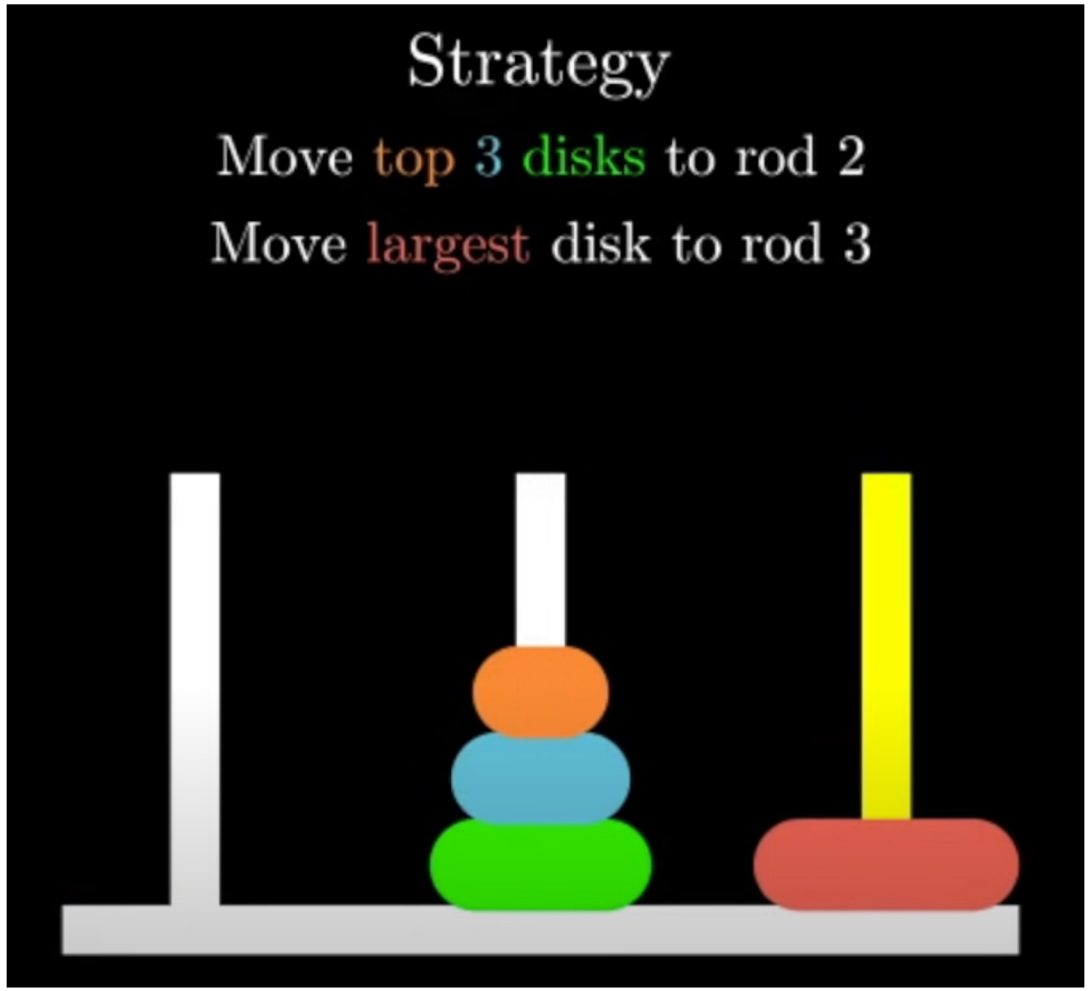


Image credit - Reducible on YouTube

Divide and Conquer

Towers of Hanoi



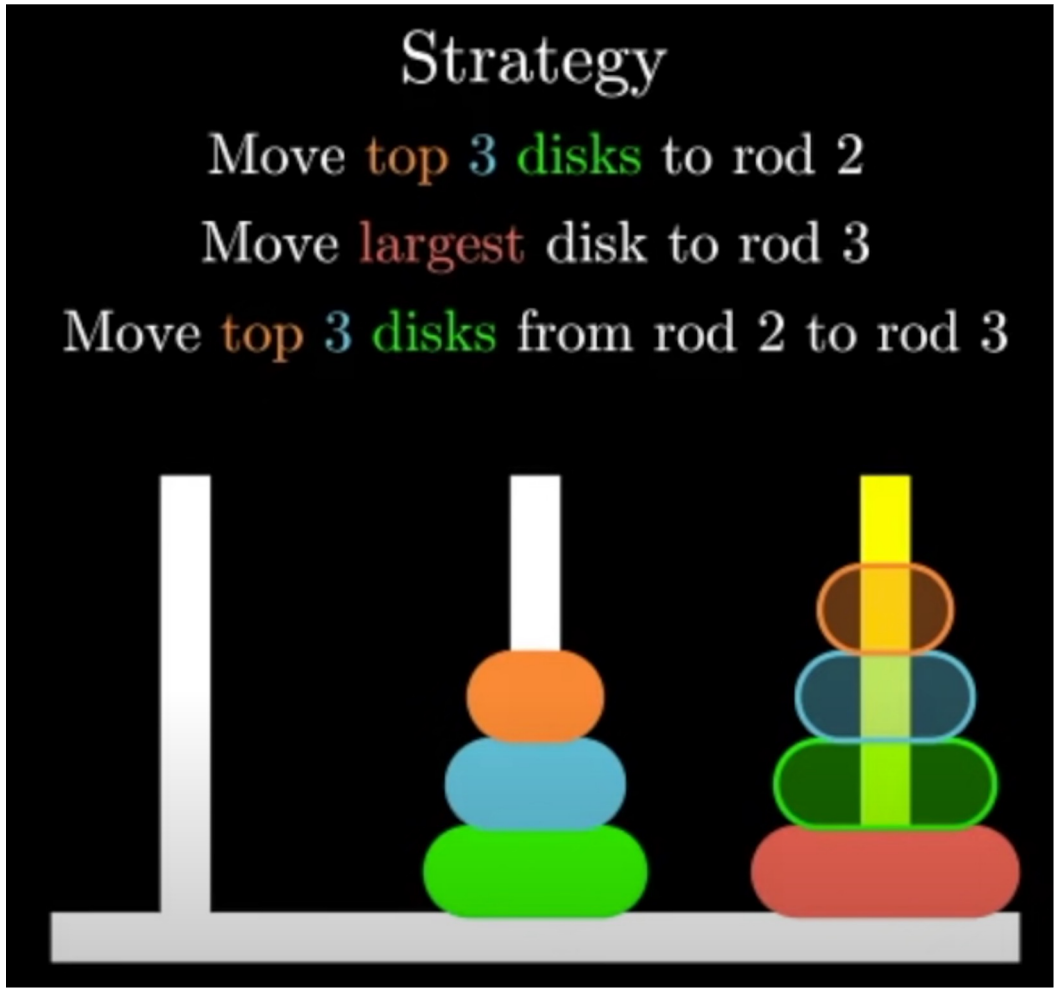
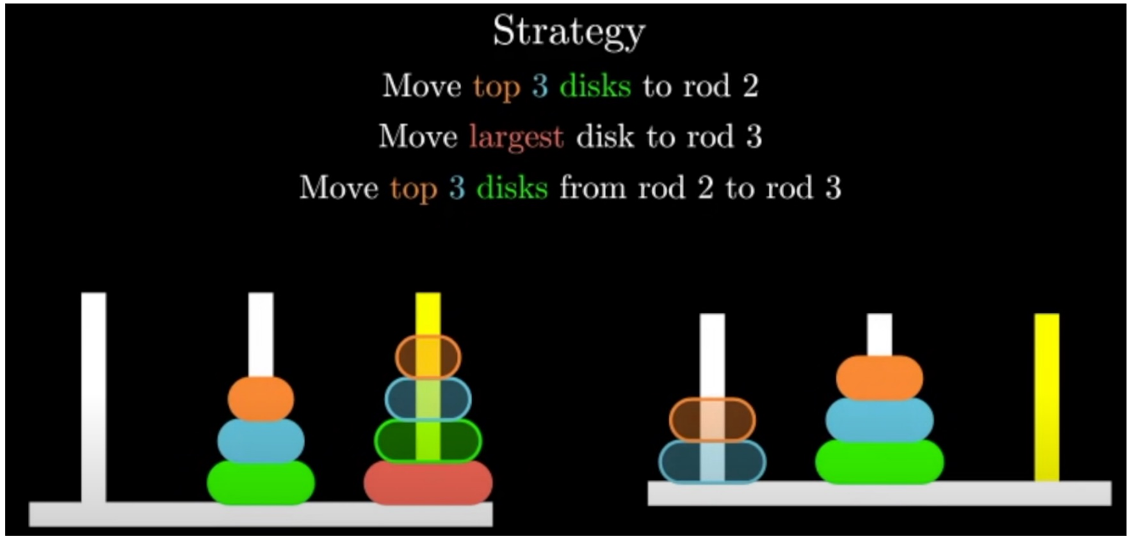
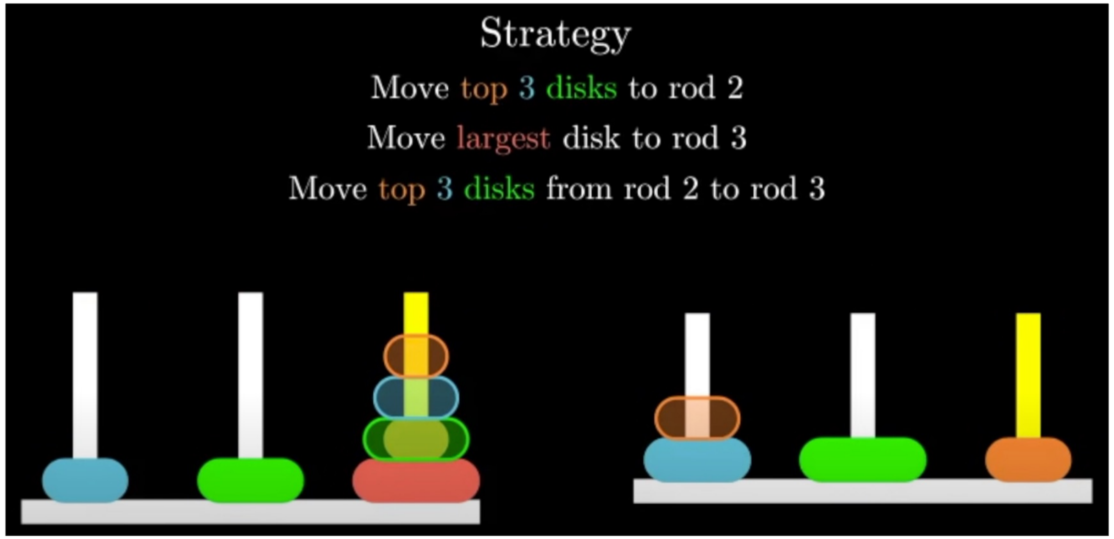
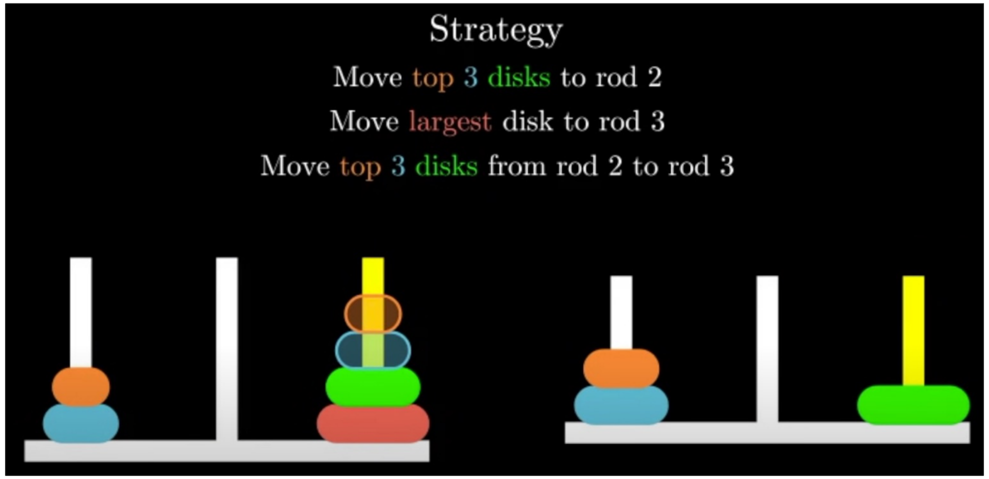


Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi







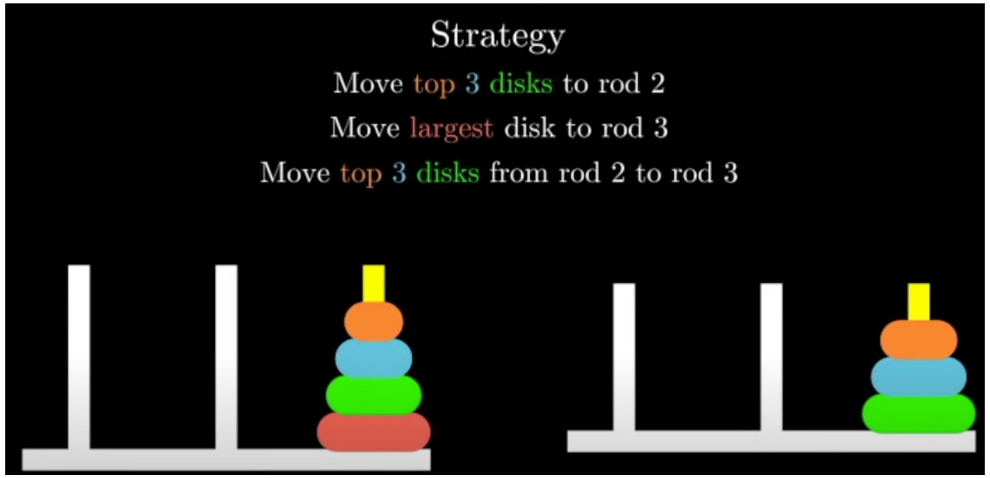


Image credit - Reducible on YouTube

Divide and Conquer
Towers of Hanoi

- We first moved the top 3 discs from rod 1 to rod 2
- This involved a couple of intermediate steps -
  - We first moved the top 2 discs from rod 1 to rod 3
  - We moved the largest rod (largest in the subtask) from rod 1 to rod 2
  - Then we move the top 2 disks back from rod 3 to rod 2
- We are then free to move the largest disk from rod 1 to the destination rod

We use similar subtasks to move the top 3 disks from the middle rod to the destination rod on top of the largest rod.