What the heck is Big-O notation ?

Why is my code running slow ? ☹

Donald Knuth Says:

Topics Covered

1. Why should we care about Big-O ?
2. The Three Musketeers(O, Ω, Θ)
3. Code Analysis: Adding an Array of Numbers
4. Code Analysis: Searching for a Target in a sorted Array
5. Big-O Algebra and Solving recurrences
6. Code Analysis: Fibonacci Sequence
7. Keep Thinking: How many moves ?

1. Why should we care about Big-O ?

1. Three Broad Reasons

- Optimal use of resources: Space(ram, 16 GB) and time(processor speed, i7, 2.9Ghz)
- Analyzing different solutions and choosing the optimal one
- Code that can Scale(Will my code execute in acceptable time limit if the input size is >=1 billion ? Or will it hit the hardware limit ?)

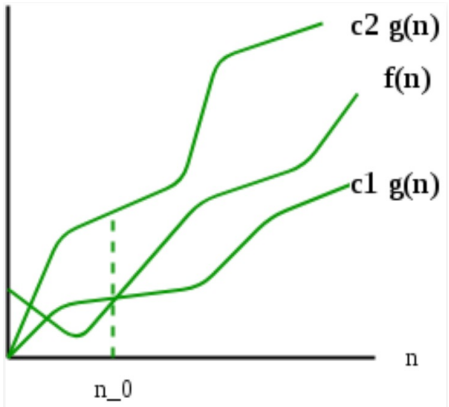
2. The Three Musketeers (O, Ω, Θ)

2. The Three Musketeers(O, Ω, Θ)

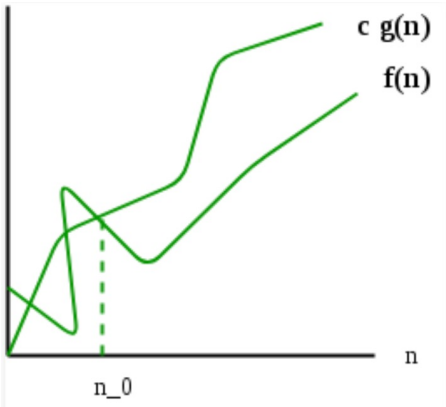
Big-O: Upper (space and time)Bound of an algorithm(Worst Case)

Big-Ω: Lower (space and time) Bound of an algorithm(Best Case, Bogus)

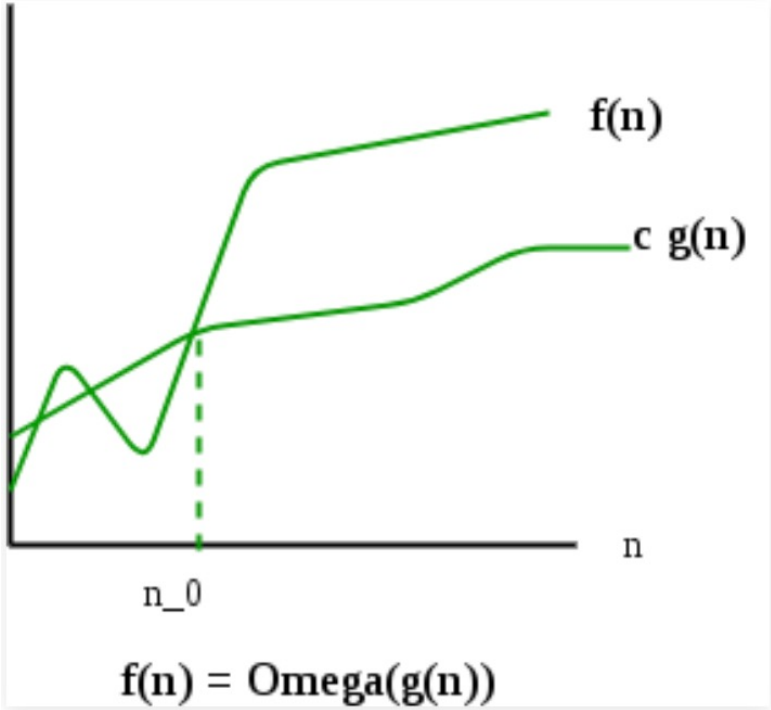
Big-Θ: Upper + Lower (space and time) Bound of an algorithm



f(n) = theta(g(n))

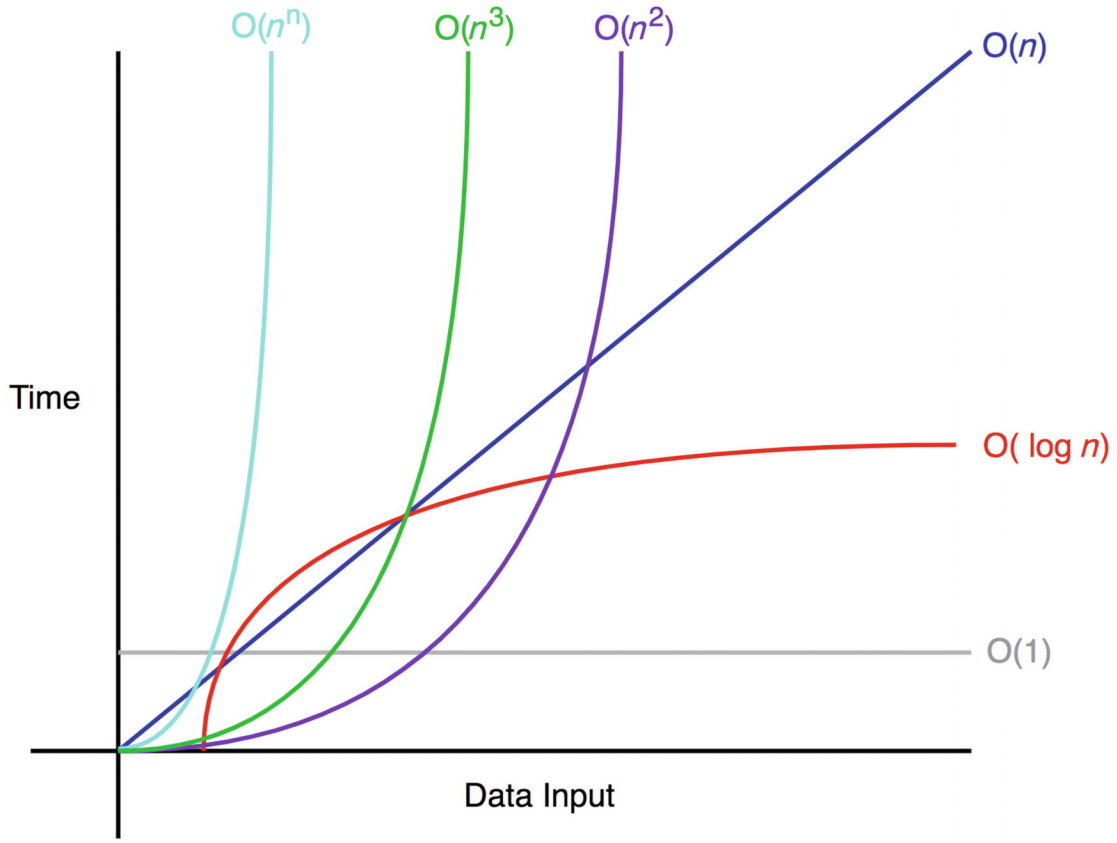


f(n) = O(g(n))



f(n) = Omega(g(n))

2. The First Musketeer(Big-O)



3. Code Analysis: Adding an [Array] of Numbers

3. Code Analysis: Adding an Array of Numbers

```swift
// Swift
func findSum() -> Int8 {
    let numArray: [Int8] = [1,2,3,4,5,6,7,8,9,10]
    var sum: Int8 = 0 // 1 Byte
    for num in numArray {
        sum = sum + num
    }
    return sum
}
```

```kotlin
// Kotlin
fun findSum(): Int {
    val numList = listOf<Int>(1,2,3,4,5,6,7,8,9,10)
    var sum: Int = 0 // Int = 4 bytes
    for(num in numList) {
        sum = sum + num
    }
    return sum
}
```

```javascript
// Javascript
function findSum() {
  let numArray = [1,2,3,4,5,6,7,8,9,10]
  let size = 10
  var sum = 0 // number = 8 Bytes
  for(i = 0; i<size; i++) {
    sum = sum + numArray[i]
  }
  return sum
}
```

3. Worst Case Space-Time Analysis

Space Complexity = 10x 1 byte(Int8) + 1 byte(Int8) = N+c bytes = **O(N)**

When N is large we ignore the constants, space varies linearly with the input array size N

Time Complexity= C1 + C2.N (A linear function of Input size, N) = **O(N)**

We Ignore the constants when N is large. Time taken increases linearly with input array size N

C1 = Constant, Time taken to execute is independent of input Array size. It takes the same amount of time always for a given processor = **O(1)**

4. Code Analysis: Search a Target in a sorted Array

4. Code Analysis: Search a Target in a Given Array

```swift
//Swift
func searchTarget() -> Int {
    let numArray: [Int] = [2,5,8,12,16,23,38,56,72,91]
    let size: Int = 10 // 8 Bytes
    let target: Int = 92
    for i in 0..<size {
        if numArray[i] == target {
            return i
        }
    }
    return -1
}
```

```kotlin
// Kotlin
fun searchTarget(): Int {
    val numList = listOf<Int>(2,5,8,12,16,23,38,56,72,91)
    val size: Int = 10 // Int = 4 Bytes
    val target: Int = 92
    for(i in 0..size-1) {
        if(numList[i] == target) {
            return i
        }
    }
    return -1
}
```

```javascript
// Javascript
function searchTarget() {
  let numArray = [2,5,8,12,16,23,38,56,72,91]
  let size = 10
  let target = 92 // number = 8 Bytes
  for(i = 0; i<size; i++) {
    if(numArray[i] == target) {
      return i
    }
  }
  return -1
}
```

4. Worst Case Space-Time Analysis(1)

Space Complexity = 10x 8 bytes(Int) + c = N+ c bytes = **O(N)**

When N is large we ignore the constants, space varies linearly with the input array size N

Time Complexity(Best case, found at index 0**)** = c = **Ω(1)** = independent of input size

Time Complexity(Worst case, not found**)** = c1 + c2.N = **O(N)** = Time varies linearly with input size

4. Can we Optimize the search?

4. Using Binary Search

```swift
//Swift
func searchTarget(start: Int,
                  end: Int,
                  numArr: [Int],
                  target: Int) -> Int {
    if start > end {
        return -1
    }
    let mid: Int = start + (end-start)/2
    if numArr[mid] == target {
        return mid
    } else if numArr[mid] < target {
        return searchTarget(start: mid+1,
                            end: end,
                            numArr: numArr,
                            target: target)
    } else {
        return searchTarget(start: start,
                            end: mid-1,
                            numArr: numArr,
                            target: target)
    }
}
```

```kotlin
// Kotlin
fun searchTarget(start: Int,
                 end: Int,
                 numArr: List<Int>,
                 target: Int): Int {
    if(start > end) {
        return -1
    }
    val mid = start + (end-start)/2
    if(numArr[mid] == target) {
        return mid
    } else if(numArr[mid] < target) {
        return searchTarget(mid+1,
                            end,
                            numArr,
                            target)
    } else {
        return searchTarget(start,
                            mid-1,
                            numArr,
                            target)
    }
}
```

```javascript
// Javascript
function searchTarget(start,
                      end,
                      numArray,
                      target) {
  if(start > end) return -1
  let mid = start + (end-start)//2
  if(numArray[mid] == target)
    return mid
  else if(numArray[mid] < target)
    return searchTarget(mid+1,
                        end,
                        numArray,
                        target)
  else return searchTarget(start,
                           mid-1,
                           numArray,
                           target)
}
```

4. Lets see an Example

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

23 > 16
take 2nd half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

23 > 56
take 1st half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91

Found 23,
Return 5

0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

4. Worst Case Space-Time Analysis(2)

[0-2, 1-5, 2-8, 3-12, 4-16, 5-23, 6-38, 7-56, 8-72, 9-91]

→searchTarget(0, 9, input_Array, 21), mid = 4

→searchTarget(5, 9, input_Array, 21), mid = 7

→searchTarget(8, 9, input_Array, 21), mid = 8

→searchTarget(9, 9, input_Array, 21), mid = 9

→searchTarget(9, 9, input_Array, 21), terminate at base case

The problem size reduces by half with every function call**, N, N/2, N/4, N/16** and stops when N = 2^K, **K = count of function call** = LogN

Time Complexity = Time taken to execute one function X LogN = c.LogN = **O(LogN)**

4. Worst Case Space-Time Analysis(2)

Space Complexity = stack space = LogN.(1 byte + 1 byte + N.1Byte + 1 Byte) = **O(NLogN)**(Array passed by value)

Space Complexity = O(LogN)(Array passed by reference)

Can we further optimize this code ? May be we break the problem into 3 parts, ternary search, then T = O(LogN, base 3)

4. Can we get rid of Stack Space ?

4.Converting Recursion to Iteration

```swift
//Swift
func searchTarget(numArr: [Int],
                  target: Int,
                  size: Int) -> Int {
    var start = 0
    var end = size-1
    while start <= end {
        let mid = start + (end-start)/2
        if numArr[mid] == target {
            return mid
        } else if numArr[mid] < target {
            start = mid + 1
        } else {
            end = mid - 1
        }
    }
    return -1
}
```

```kotlin
// Kotlin
fun searchTarget(numArr: List<Int>,
                 target: Int,
                 size: Int): Int {
    var start = 0
    var end = size-1
    while(start <= end) {
        val mid = start + (end-start)/2
        if(numArr[mid] == target) {
            return mid
        } else if(numArr[mid] < target) {
            start = mid + 1
        } else {
            end = mid - 1
        }
    }
    return -1
}
```

```javascript
// Javascript
function searchTarget(numArr,
                      target,
                      size) {
  var start = 0
  var end = size - 1
  while(start <= end) {
    let mid = start + (end-start)//2
    if(numArr[mid] == target) {
      return mid
    } else if(numArr[mid] < target) {
      start = mid + 1
    } else end = mid-1
  }
  return -1
}
```

4. Worst Case Space-Time Analysis(3)

Space Complexity = stack space = 8 byte + 8 byte + N.8Byte + 8 Byte = **O(N)**(Array passed by value)

Space Complexity = O(1)(Array passed by reference)

Time Complexity = **O(LogN)**

5. Big- O Algebra and Recurrences

5. Big- O Algebra

```swift
//Swift
let arr1 = [1,2,3,4,5,6,7,8,9,10]
let arr2 = [11,12,13,14,15,16,17,18,19,20]
var sum = 0
for num in arr1 {
    sum += num
}
for num in arr2 {
    sum += num
}
```

```kotlin
// Kotlin
val arr1 = listOf<Int>(1,2,3,4,5,6,7,8,9,10)
val arr2 = listOf<Int>(11,12,13,14,15,16,17,18,19,20)
var sum = 0
for(num in arr1) {
    sum = sum + num
}
for(num in arr2) {
    sum = sum + num
}
```

```javascript
// Javascript
let arr1 = [1,2,3,4,5,6,7,8,9,10]
let arr2 = [11,12,13,14,15,16,17,18,19,20]
let size1 = 10
let size2 = 10
var sum = 0
for(i = 0; i<size1; i++) {
  sum += arr1[i]
}
for(i = 0; i<size2; i++) {
  sum += arr2[i]
}
```

Time Complexity, $T(N) = O(N1) + O(N2) = O(N1 + N2)$

Space Complexity $= O(N1) + O(N2) = O(N1 + N2)$

5. Big- O Algebra

```swift
//Swift
let arr1 = [1,2,3,4,5,6,7,8,9,10]
let arr2 = [11,12,13,14,15,16,17,18,19,20]
var sum = 0
for num1 in arr1 {
    for num2 in arr2 {
        sum += num1 + num2
    }
}
```

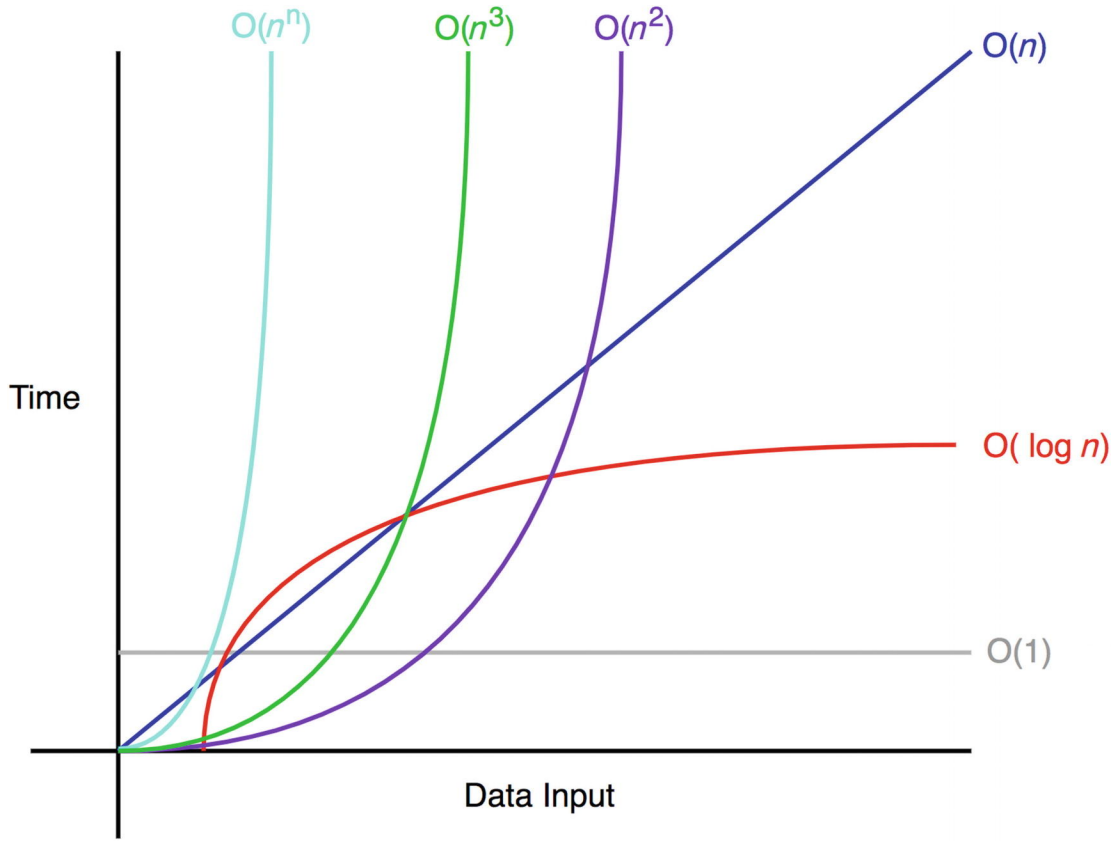
```kotlin
// Kotlin
val arr1 = listOf<Int>(1,2,3,4,5,6,7,8,9,10)
val arr2 = listOf<Int>(11,12,13,14,15,16,17,18,19,20)
var sum = 0
for(num1 in arr1) {
    for(num2 in arr2) {
        sum += num1 + num2
    }
}
```

```javascript
// Javascript
let arr1 = [1,2,3,4,5,6,7,8,9,10]
let arr2 = [11,12,13,14,15,16,17,18,19,20]
let size1 = 10
let size2 = 10
var sum = 0
for(i = 0; i<size1; i++) {
  for(j = 0; j<size2; j++) {
      sum += arr1[i] + arr2[j]
    }
}
```

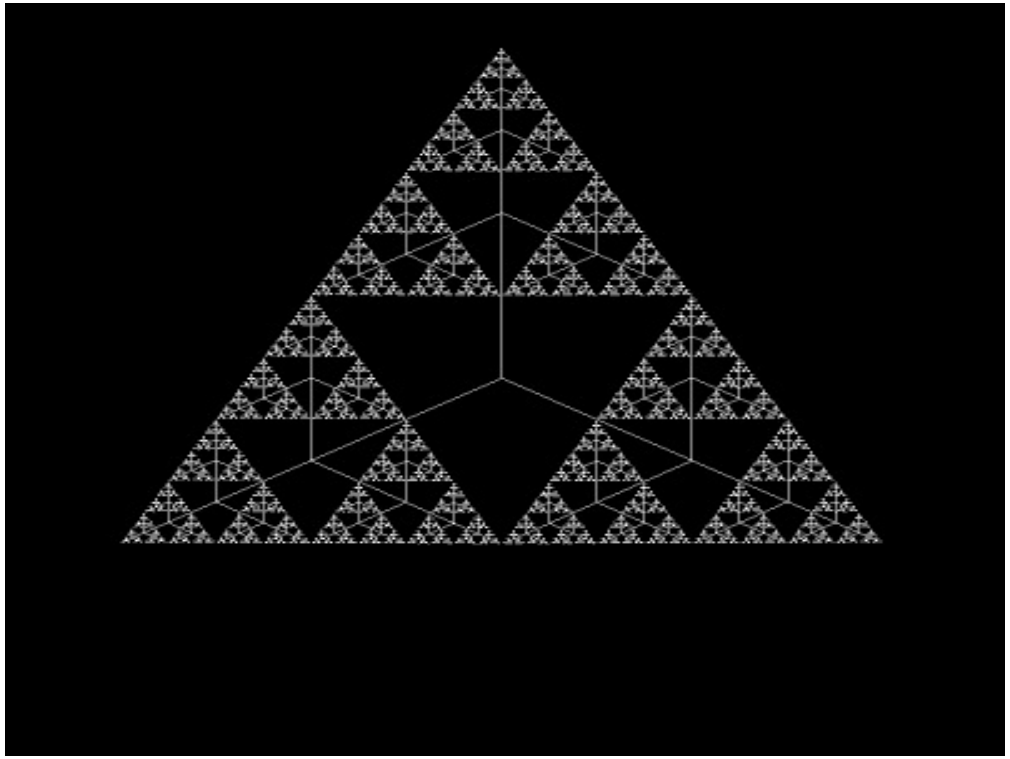
Time Complexity, T(N) = O(N1)*O(N2) = O(N1^N2)

Space Complexity = O(N1) + O(N2) = O(N1 + N2)

5. What is O(N^2) + O(logN) = ?



5. How do we solve a recurrence ?



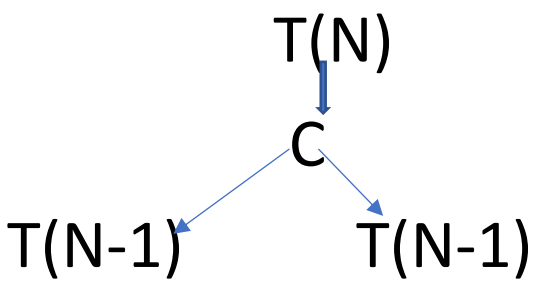
5. Three Techniques

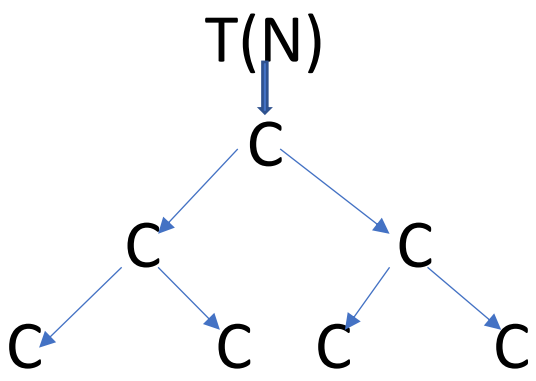
- Recurrence Tree Method
- Substitution Method
- Master Theorem

5. Recurrence Tree

Recurrence Relation: T(N) = T(N-1) + T(N-2) + C

T(N) <= 2T(N-1) + C (since T(N-1) > T(N-2))





→ 2^0 x C

→ 2^1 x C

→ 2^2 x C

T(N) <= c + 2c + 4c + 16c + 32c …2^n c <= (2^(N-1)-1).c <= 2^N

T(N) = O(2^N)

5. Using Master Theorem

```
T(n) = aT(n/b) + f(n) where a >= 1 and b > 1
```

There are following three cases:

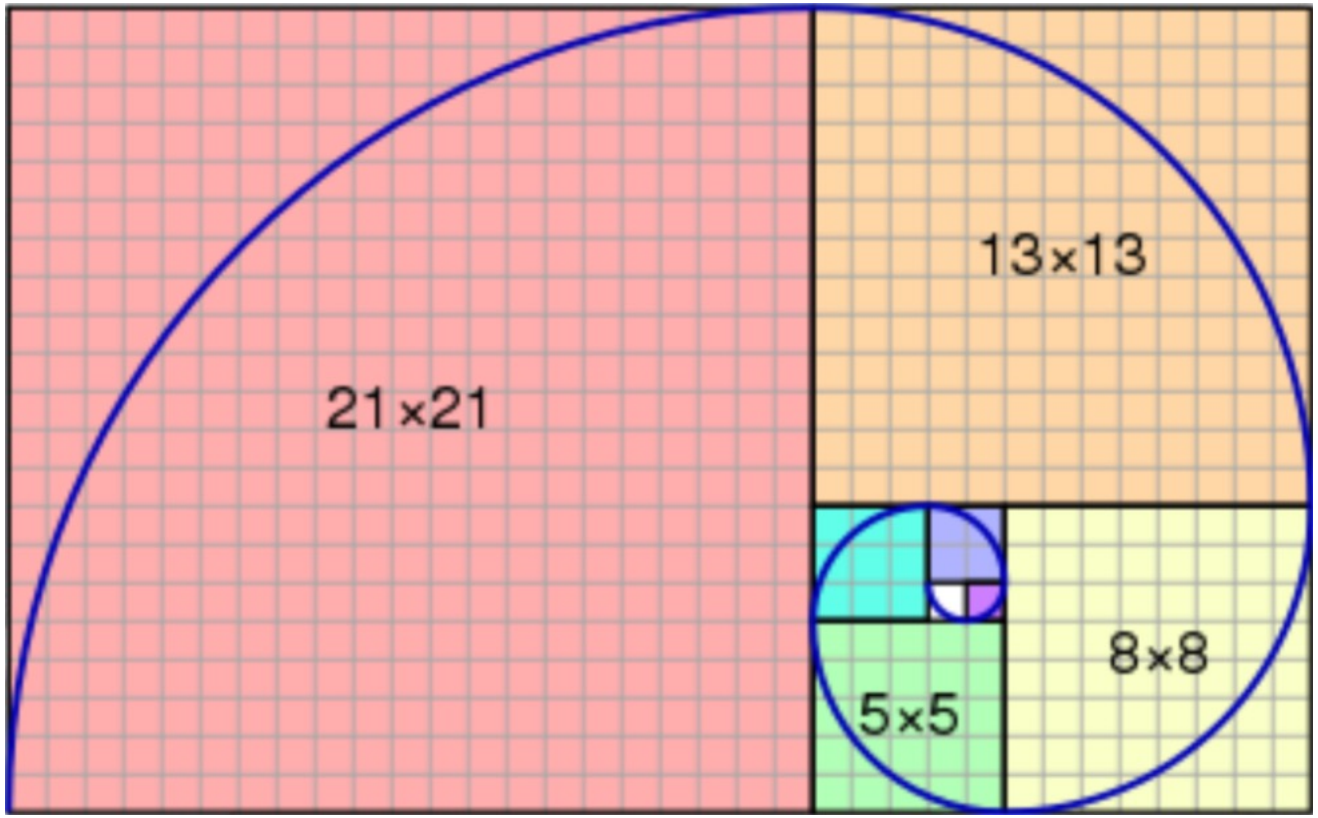
1. If $f(n) = \Theta(n^c)$ where $c < \text{Log}_b a$ then $T(n) = \Theta(n^{\text{Log}_b a})$

2. If $f(n) = \Theta(n^c)$ where $c = \text{Log}_b a$ then $T(n) = \Theta(n^c \text{Log } n)$

3. If $f(n) = \Theta(n^c)$ where $c > \text{Log}_b a$ then $T(n) = \Theta(f(n))$

6. Code Analysis: Fibonacci Sequence

6. Code Analysis: Fibonacci Sequence



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233

```swift
//Swift
func fib(n: Int) -> Int {
    if n == 0 {
        return 0
    } else if n <= 2 {
        return 1
    }
    return fib(n: n-1) + fib(n: n-2)
}
```

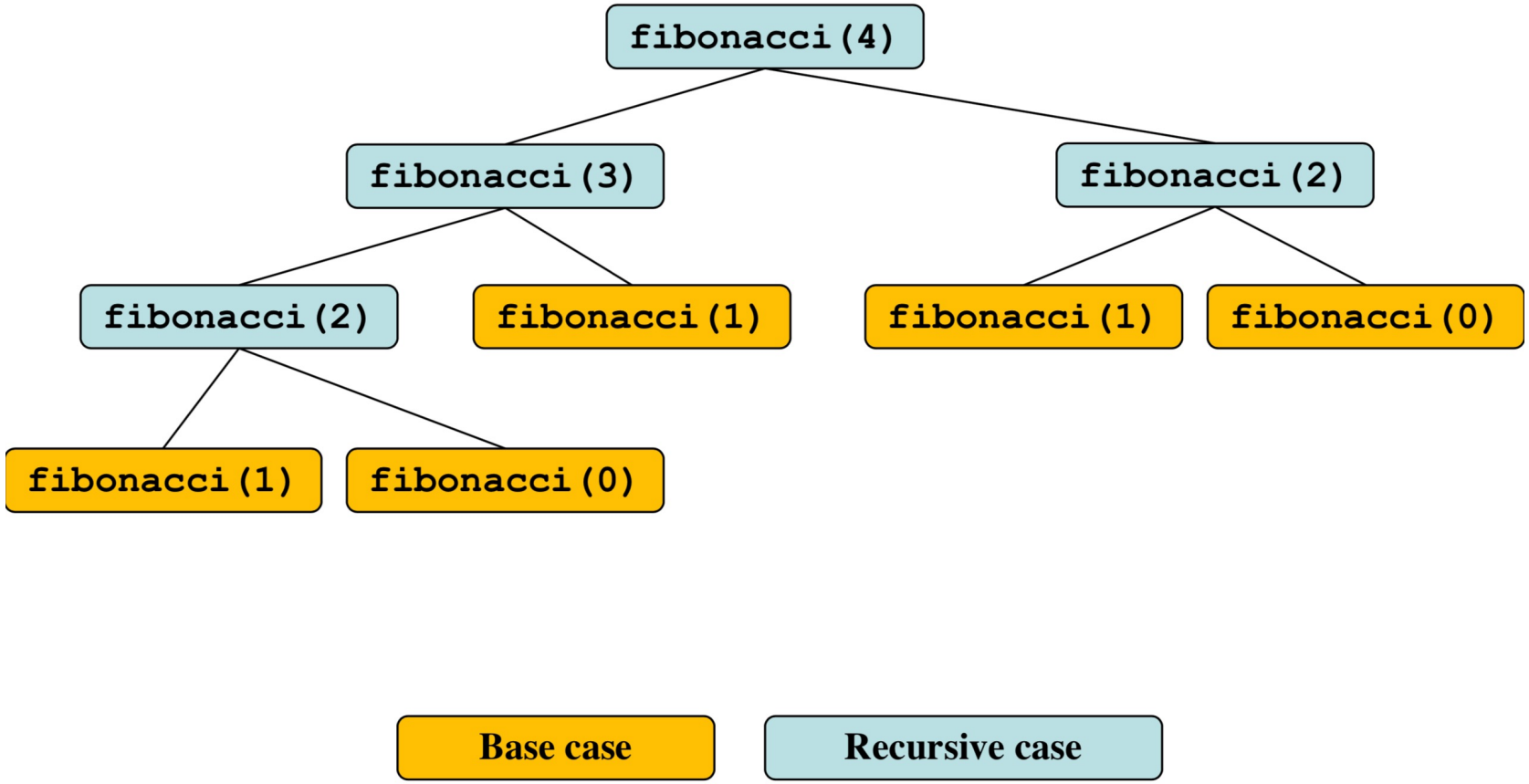
```kotlin
// Kotlin
fun fib(n: Int): Int {
    if(n == 0) {
        return 0
    } else if(n <= 2) {
        return 1
    }
    return fib(n-1) + fib(n-2)
}
```

```javascript
// Javascript
function fib(n) {
  if(n == 0) return 0
  else if(n<= 2) return 1
  return fib(n-1) + fib(n-2)
}
```

Time Complexity, T(N) = O(2^N)

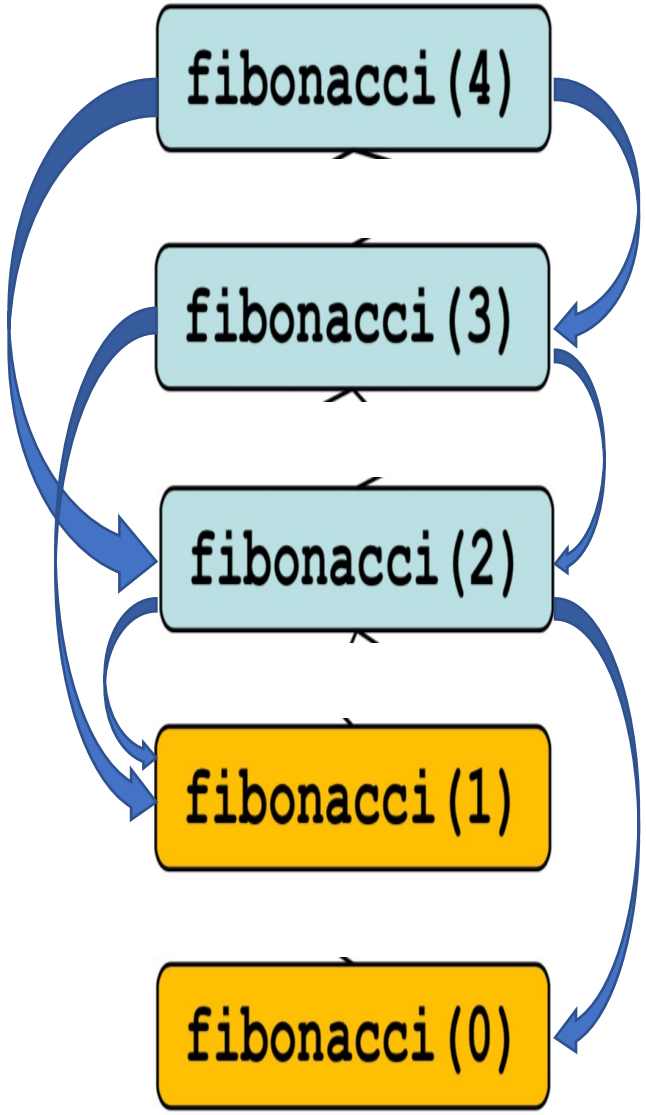
Space Complexity = O(2^N)

6. T(N) = T(N-1) + T(N-2) => O(2^N)



6. Lets Optimize it!!

6. Solve a subproblem once!!



6. Solve a subproblem once!!

```swift
//Swift
var cache = Array(repeating: -1, count: 20)
func fib(n: Int) -> Int {
    if n == 0 {
        return 0
    } else if n <= 2 {
        return 1
    }
    if cache[n] != -1 {
        return cache[n]
    } else {
        cache[n] = fib(n: n-1) + fib(n: n-2)
        return cache[n]
    }
}
```

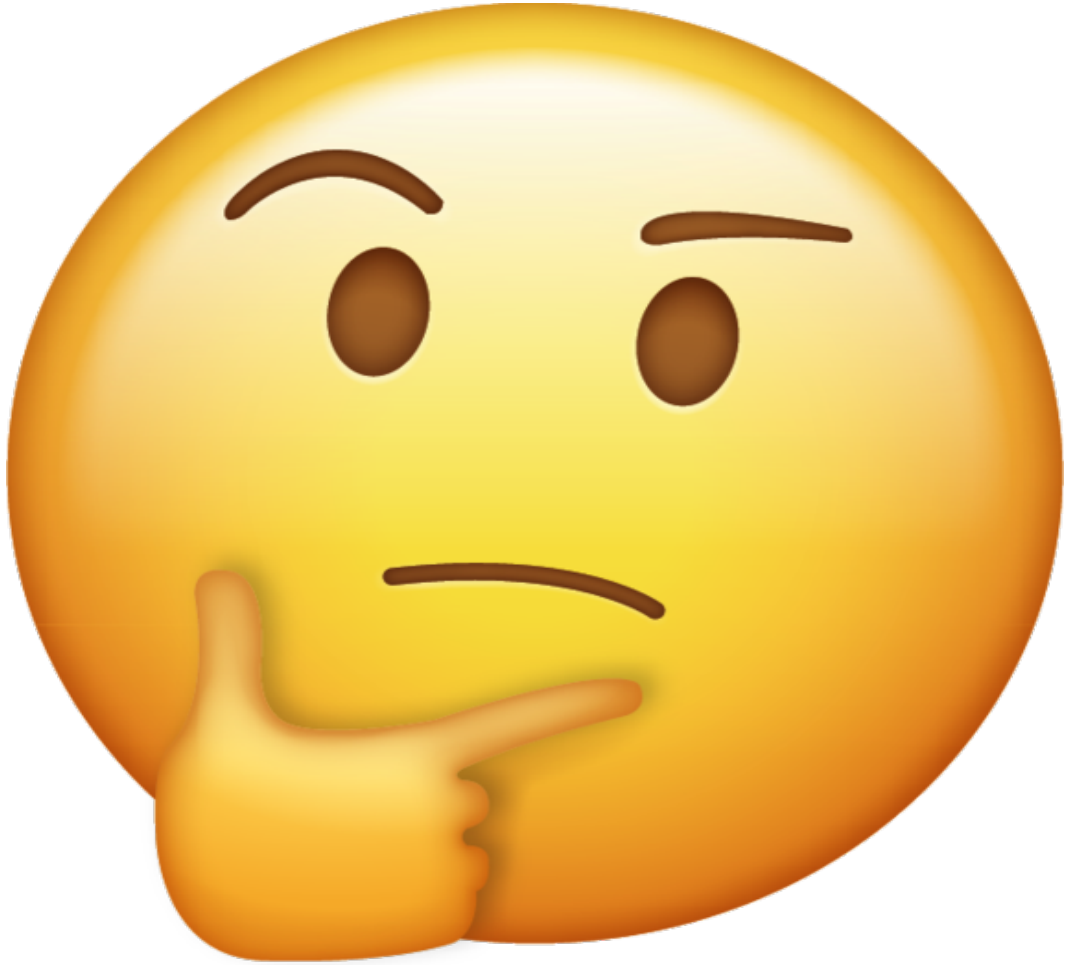
```kotlin
// Kotlin
val cache = Array(20, {_ -> -1})
fun fib(n: Int): Int {
    if(n == 0) {
        return 0
    } else if(n <= 2) {
        return 1
    }
    if(cache[n] != -1) {
        return cache[n]
    } else {
        cache[n] = fib(n-1) + fib(n-2)
        return cache[n]
    }
}
```

```javascript
// Javascript
let cache = new Array(20)
function fib(n) {
  if(n == 0) return 0
  else if(n<= 2) return 1
  if(cache[n] != undefined) return cache[n]
  else {
    cache[n] = fib(n-1) + fib(n-2)
    return cache[n]
  }
}
```

Time Complexity, T(N) = O(N)

Space Complexity = O(N)[cache] + O(N)[stack space]

6. Can we Optimize it further ?



6. Lets get rid of stack space!!

```swift
//Swift
func fib(n: Int) -> Int {
    if n == 0 {
        return 0
    } else if n <= 2 {
        return 1
    }
    var cache = Array(repeating: 0, count: n+1)
    cache[1] = 1
    cache[2] = 1
    for i in 3...n {
        cache[i] = cache[i-1] + cache[i-2]
    }
    return cache[n]
}
```

```kotlin
// Kotlin
fun fib(n: Int): Int {
    if(n == 0) {
        return 0
    } else if(n <= 2) {
        return 1
    }
    val cache = Array(n+1, {_ -> 0})
    cache[1] = 1
    cache[2] = 1
    for(i in 3..n) {
        cache[i] = cache[i-1] + cache[i-2]
    }
    return cache[n]
}
```

```javascript
//Javascript
function fib(n) {
  if(n == 0) return 0
  else if(n<= 2) return 1
  let cache = new Array(n+1)
  cache[1] = 1
  cache[2] = 1
  for(i = 3; i<=n; i++) {
    cache[i] = cache[i-1] + cache[i-2]
  }
  return cache[n]
}
```

Time Complexity, T(N) = O(N)

Space Complexity = O(N)[cache]

6. Any further optimizations possible?

6. Lets get rid of cache!!

```swift
//Swift
func fib(n: Int) -> Int {
    if n == 0 {
        return 0
    } else if n <= 2 {
        return 1
    }
    var prev = 1, value = 1
    for _ in 3...n {
        let newPrev = value
        value = value + prev
        prev = newPrev
    }
    return value
}
```

```kotlin
// Kotlin
fun fib(n: Int): Int {
    if(n == 0) {
        return 0
    } else if(n <= 2) {
        return 1
    }
    var prev = 1; var value = 1
    for(i in 3..n) {
        val newPrev = value
        value = value + prev
        prev = newPrev
    }
    return value
}
```

```javascript
//Javascript
function fib(n) {
  if(n == 0) return 0
  else if(n<= 2) return 1
  var prev = 1, value = 1
  for(i = 3; i<=n; i++) {
    let newPrev = value
    value = value + prev
    prev = newPrev
  }
  return value
}
```

Time Complexity, T(N) = O(N)

Space Complexity = O(1)

6. Last Optimization?

6. Lets use Golden Ratio!!

```swift
//Swift
func fib(n: Double) -> Double {
    // Golden Ratio
    let phi = (1+sqrt(5))/2
    return round(pow(phi, n)/sqrt(5))
}
```

```kotlin
// Kotlin
fun fib(n: Double): Double {
    val phi: Double = (1 + sqrt(5.0))/2
    return round(phi.pow(n)/sqrt(5.0))
}
```

```javascript
//Javascript
function fib(n) {
  let phi = (1+Math.sqrt(5.0))/2
  return Math.round(Math.pow(phi, n)/Math.sqrt(5.0))
}
```

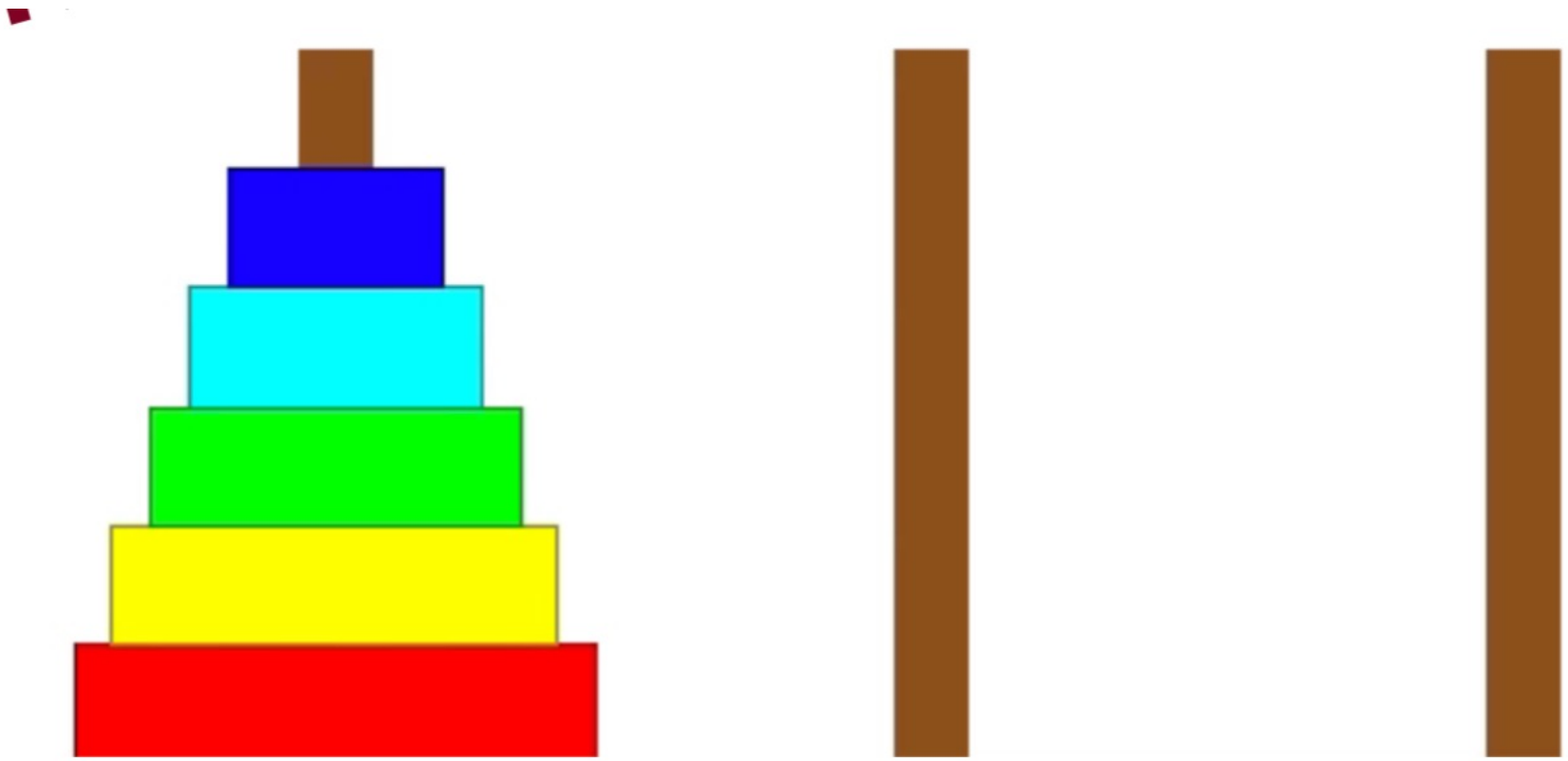
Fib(n) = {Φ^n} /√5, Φ = (√5 + 1)/2 = Golden Ratio

Time Complexity, T(N) = O(1) because the code is single line ?

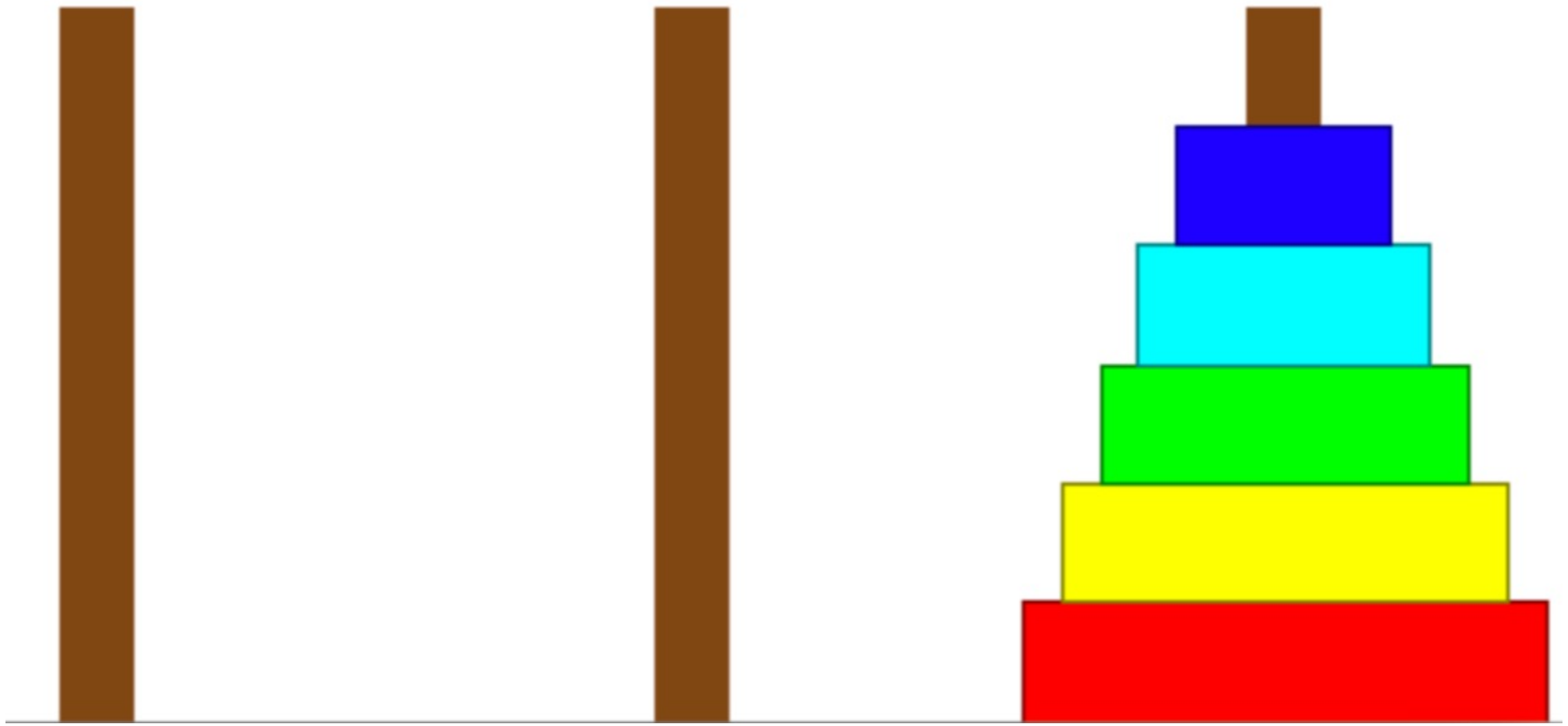
Space Complexity = O(1)

7. Keep Thinking: How many Moves?

7. Keep Thinking: How many Moves?



7. Keep Thinking: How many Moves?



References

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/
- https://www.geeksforgeeks.org/fundamentals-of-algorithms/
- Introduction to Algorithms by Thomas H. Cormen
- https://codeforces.com/blog/entry/23054
- https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)
- https://en.wikipedia.org/wiki/Golden_ratio