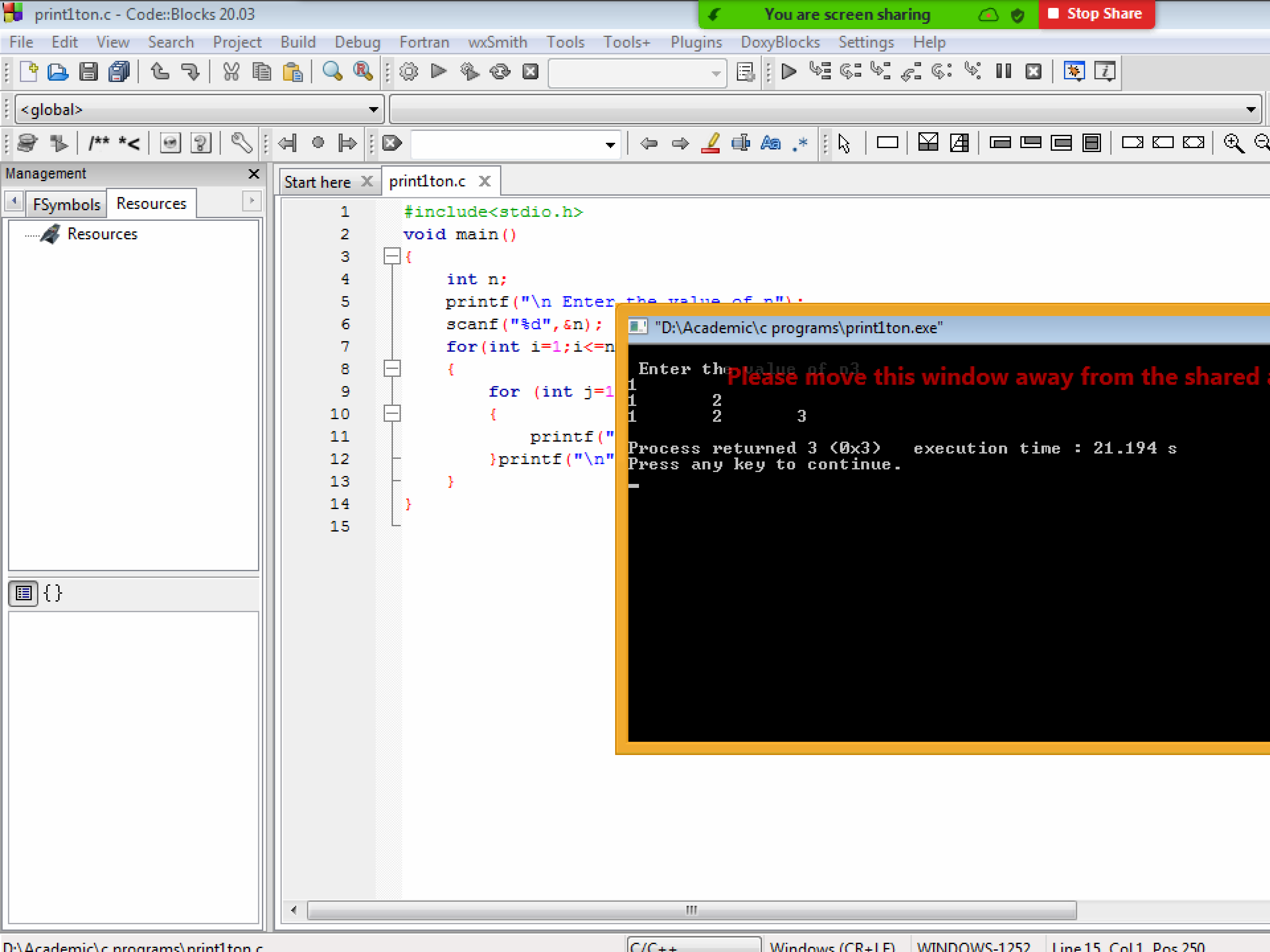

print1ton.c - Code::Blocks 20.03
You are screen sharing
Stop Share
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
FSymbols Resources
Resources
Start here
print1ton.c
#include<stdio.h>
void main()
{
int n;
printf("\n Enter the value of n");
scanf("%d",&n);
for(int i=1;i<=n
{
for (int j=1
{
printf("
}printf("\n"
}
}
"D:\Academic\c programs\print1ton.exe"
Enter the value of n3
Please move this window away from the shared a
1
1 2
1 2 3
Process returned 3 (0x3) execution time : 21.194 s
Press any key to continue.
D:\Academic\c programs\print1ton.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 15, Col 1, Pos 250

Fundamentals of the Analysis of Algorithm Efficiency

- Analysis Framework
- Asymptotic Notations and its properties
- Mathematical analysis of Non - Recursive algorithms
- Mathematical analysis of Recursive algorithms

Mathematical analysis of Recursive algorithms

General plan for Analyzing the time efficiency of Recursive algorithm

1. Decide on a parameter (or parameters) indicating an **input's size.**
2. Identify the algorithm's **basic operation.**
3. Check whether the **number of times the basic operation** is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.
4. Set up a **recurrence relation**, with an appropriate initial condition, for the number of times the basic operation is executed.
5. Solve the recurrence or, at least, ascertain the **order of growth** of its solution.

Mathematical analysis of Recursive algorithms

- Recursive Function – function that calls itself
- Example 1: Factorial of a given number

$n! = 1 \ldots\ldots (n-1) \,.\, n = (n-1)! * n$ for $n \geq 1$

$F(n) = F(n-1) \,.\, n$ for $n > 0$,

```
ALGORITHM F(n)
    //Computes n! recursively
    //Input: A nonnegative integer n
    //Output: The value of n!
    if n = 0 return 1
    else return F(n − 1) ∗ n
```

Example 1: Factorial of a given number

- *F(n)* = *F(n* − 1*)* . *n* for *n* > 0
- No.of multiplications ***(Recurrence relation)***

$$M(n) = \underset{\substack{\text{to compute} \\ F(n-1)}}{M(n-1)} + \underset{\substack{\text{to multiply} \\ F(n-1) \text{ by } n}}{1} \quad \text{for } n > 0.$$

- ***Initial condition – sequence***

if *n*=0 return 1

n=0 → no multiplications are done

$$M(0) = 0.$$

the calls stop when $n = 0$ → $M(0)$

no multiplications when $n = 0$ → 0

Example 1: Factorial of a given number

- F(n)=F(n-1) . N
- F(0) = 1
- $M(n) = M(n-1) + 1$

$= [M(n-2) +1] + 1 = M(n-2) +2$

$= [M(n-3)+ 2] + 1 = M(n-3) + 3$

$M(n) = M(n-i) + i$

If $i=n$,

$M(n) = M(n-n)+n$

$= M(0) + n$

$= n$

Example 2: Towers of Hanoi

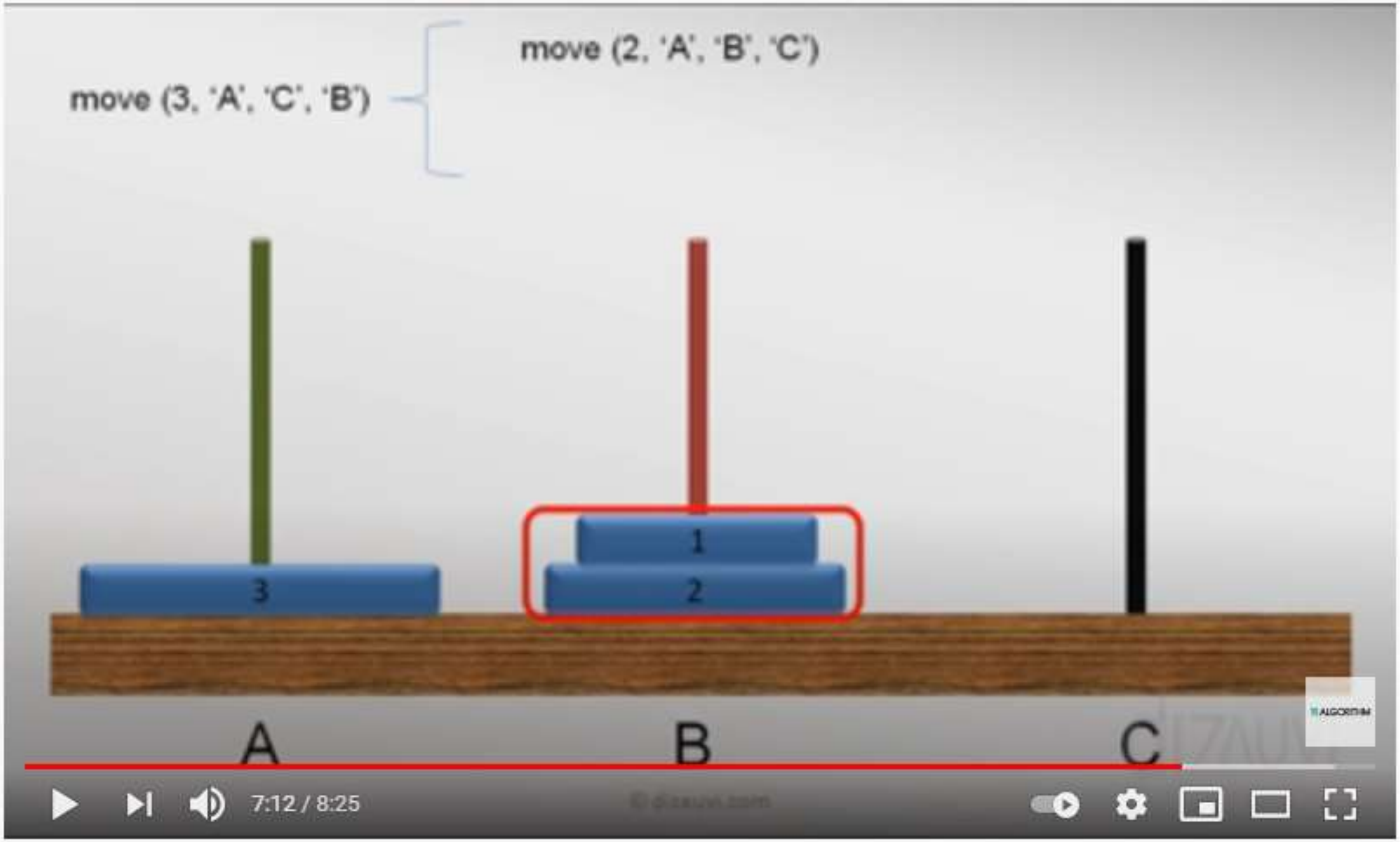
Problem statement : Given n disks of different sizes and 3 rods. Initially all the disks are in the 1^{st} rod, largest on the bottom and smallest on the top.

The goal is to move all the disks to 3^{rd} rod with the help of 2^{nd} rod if essential.

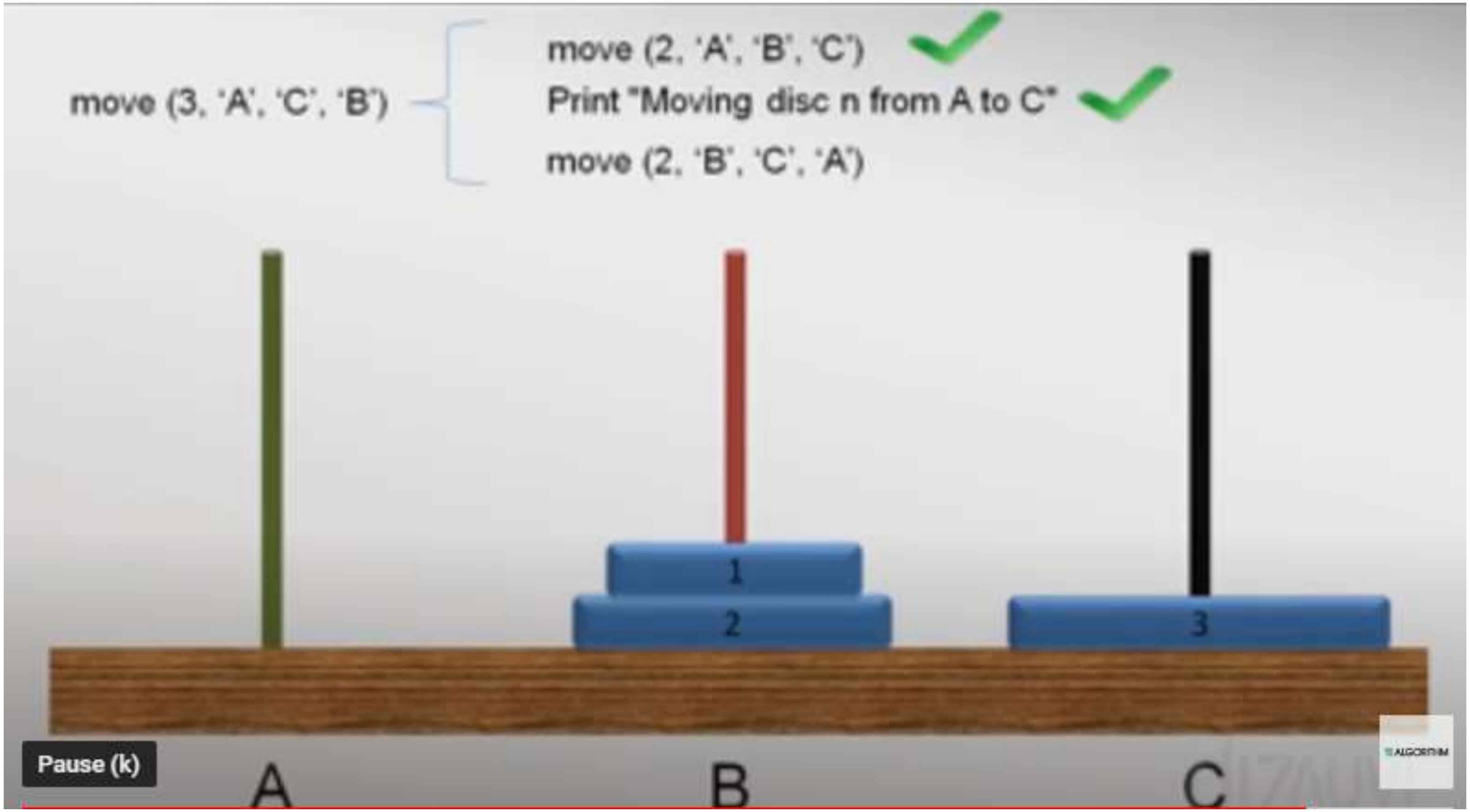
Condition 1: Move one disk at a time

Condition 2: place smaller disk on larger disk

Setting up the Recurrence Relation



Setting up the Recurrence Relation



Example 2: Towers of Hanoi

- Initial condition M(1) = 1

(if there are only one disk we can move to 3rd rod with one move)

- M(n) = M(n-1) + 1 + M(n-1) for n>1. ***Backward Substitution***
- $M(n) = 2M(n-1) + 1$ sub. $M(n-1) = 2M(n-2) + 1$

  $= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1$ sub. $M(n-2) = 2M(n-3) + 1$

  $= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1.$
- $2^4M(n-4) + 2^3 + 2^2 + 2 + 1$
- $M(n) = 2^iM(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2 + 1 = \mathbf{2^iM(n-i) + 2^i - 1.}$
- *$[2^4 = 16]$ $[2^3+2^2+2^1+1=8+4+2+1 = 15]$*
- **Initial condition is n=1, so i= upper bound – lower bound → i=n-1**
- $M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$

  $= 2^{n-1}M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$

Analysis of problems discussed

Problem	Size of the problem	Basic operation	Count of basic operation	Efficiency class
Greatest element in list	n	Comparison inside loop A[i]>maxval	O(n)	Worst /Best
Matrix Muliplication	Order of matrix	Multiplication	$O(n^3)$	Worst
Element Uniqueness Problem	n	Comparison inside for loop	$O(n^2)$	Worst
No. of bits in a decimal number	n	Comparison	$O(\log_2 n)$	Worst/Best/Avg
Factorial of a given number	n	Multiplication	O(n)	Worst
Towers of hanoi	n	Movements	$O(2^n-1)$	Worst