CS 395 – Analysis of Algorithms

Chapter 2 – Fundamentals of the Analysis of Algorithm Efficiency

Read 2.3 & 2.4

- **Mathematical analysis of recursive algorithms**
  - **Example 4: Solve Fibonacci using the characteristic equation**
  - **Algorithm Visualization**
  - **Weekly Assignment 6**

Linear homogeneous recurrence with constant coefficients

Linear refers to the fact that X(n−1), and(n−2) appear in separate terms and to the first power.

Homogeneous refers to the fact that the total degree of each term is the same (thus there is no constant term)

- **General 2nd order linear homogeneous recurrence with constant coefficients:**

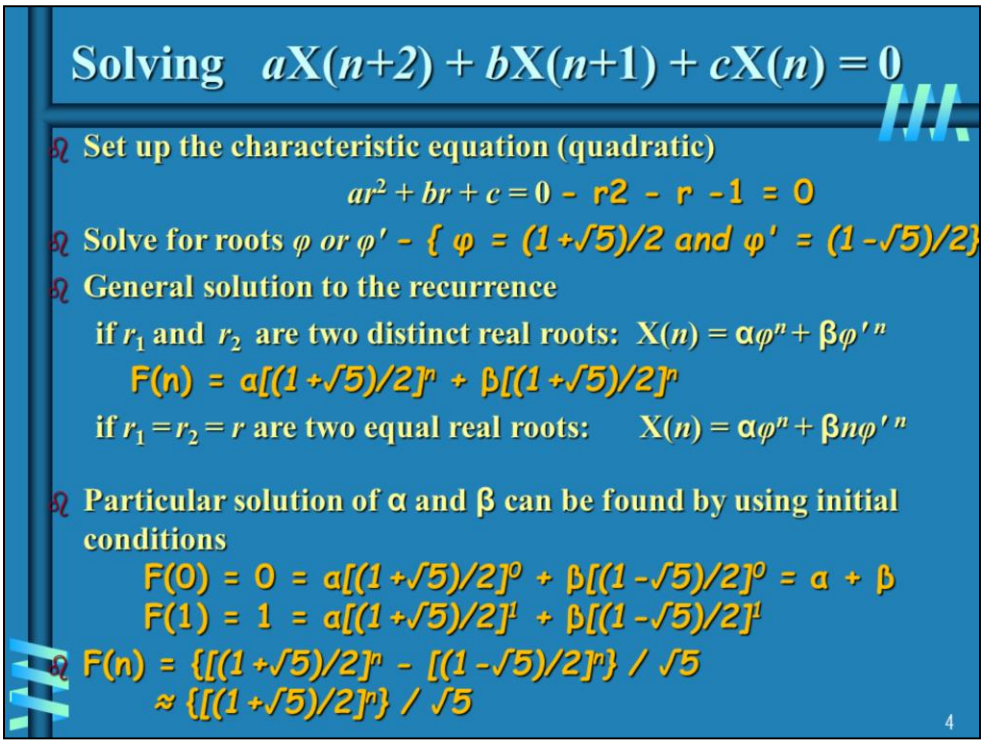
$$aX(n) + bX(n\text{-}1) + cX(n\text{-}2) = 0$$

Constant Coefficients refers to the fact that a, b and c are fixed real numbers that do not depend on n.

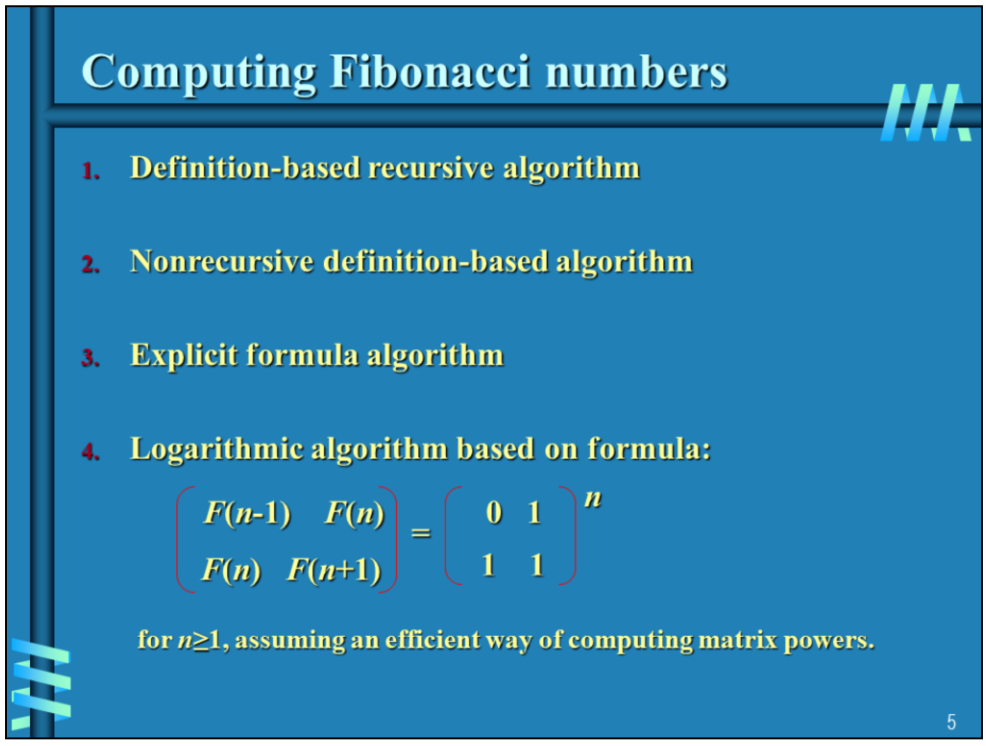
If you have a linear homogeneous recurrence with constant coefficients you can use the procedure on the next slides to solve for F(n)

Fibonacci as Linear homogeneous recurrence with constant coefficients

- $F(n) = F(n-1) + F(n-2)$
  - $F(n+2) = F(n+1) + F(n)$
  - $F(n+2) - F(n+1) - F(n) = 0$
- Characteristic equation: $r^2 - r - 1 = 0$



See excel help



Option 4: Here is a good article: https://kukuruku.co/post/the-nth-fibonacci-number-in-olog-n/ (See next –hidden - slide)

Recursive Algorithm

ALGORITHM $F(n)$

//Computes the nth Fibonacci number recursively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
if $n \leq 1$ **return** n
else return $F(n-1) + F(n-2)$

Size: n
Basic operation: addition
No difference between worst and best case
Recurrence relation:

$A(n) = A(n-1) + A(n-2) + 1$ for $n > 1$
$A(0) = A(1) = 0$

Inhomogeneous recurrence

Non-homogeneous!

Solving the recurrence

- In general solution to the inhomogeneous problem is equal to the sum of solution to homogenous problem plus solution only to the inhomogeneous part. The undetermined coefficients of the solution for the homogenous problem are used to satisfy the initial conditions.

In this case, $A(n) = B(n) + I(n)$ where

$A(n)$ is solution to complete inhomogeneous problem

$B(n)$ is solution to homogeneous problem

$I(n)$ solution to only the inhomogeneous part of the problem

Solving the recurrence

- We guess at $I(n)$ and then determine the new initial conditions for the homogenous problem for $B(n)$
- For this problem the correct guess is $I(n) = 1$
- Substitute $A(n) = B(n) - 1$ into the recursion and get

  $B(n) - B(n-1) - B(n-2) = 0$ with $B(0) = B(1) = 1$
- The same as the relation for $F(n)$ with different initial conditions
- We do not really need the exact solution; We can conclude

  $A(n) = B(n) - 1 = F(n+1) - 1 \in \Theta(\varphi^n)$, *exponential*

Iterative Algorithm

ALGORITHM *Fib*(n)

//Computes the nth Fibonacci number iteratively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
$F[0] \leftarrow 0$; $F[1] \leftarrow 1$
for $i \leftarrow 2$ **to** n **do**
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$

Size: *n*
Basic operation: addition
No difference between worst and best case
A*(*n* - 1), *linear algorithm

Algorithm Visualization

1. http://www.algomation.com/
2. https://visualgo.net/

Just for fun

- You are a parking attendant at a garage with n stalls:

- A motor bike takes 1 slot. A car takes two slots.
- The lot fills up every day.
- What is the longest number of days you can go without having the same configuration of cars/motorcycles repeated? (I.e. how many unique configurations are there?) Give a formula in terms of n.
- What is F(6)?

W06: Restricted Tower of Hanoi

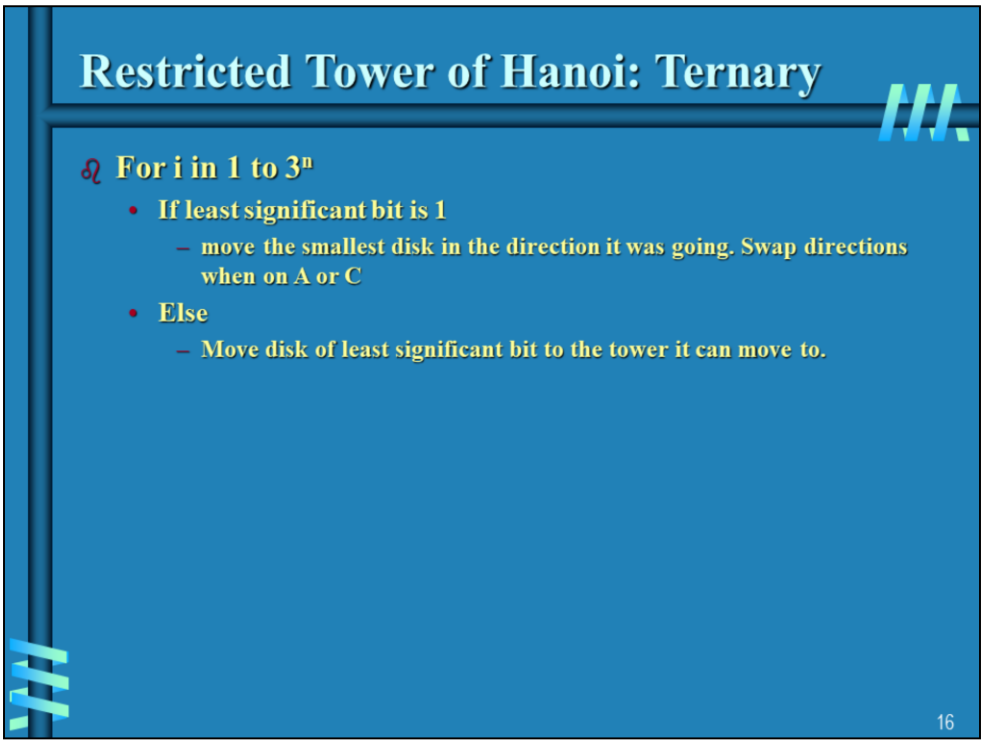
- Like the Tower of Hanoi, but you cannot move directly from A to C or from C to A

Restricted Tower of Hanoi: Recursive

- Move the 'n-1' discs from Tower A to Tower C – T(n)
- Move 'n'th disc to Tower B – 1
- Move 'n-1' discs from Tower C to Tower A – T(n)
- Move 'n'th disc from Tower B to Tower C – 1
- Move 'n-1' discs from Tower A to Tower C – T(n)
- T(n) + 1 + T(n) + 1 + T(n)

Lest solve:

- T(n) = T(n-1) + 1 + T(n-1) + 1 + T(n-1)
- T(n) = 3 * T(n-1) + 2
- T(n) + 1 = 3 * [T(n-1) + 1]
  - Let U(n) = T(n) + 1
- U(n) = 3*U(n-1)
- U(n) = 3^n (for n > 0)
  - Sub back in U(n) = T(n) + 1
- T(n) = 3^n - 1 (for n > 0)



Run on putty with:

./hanoiChecker 5 drbc.txt R