WASHINGTON STATE UNIVERSITY

CPTS 223 Advanced Data Structure C/C++

Math Review: Proof & Recursion

WSU

Proofs

- What do we want to prove?
  - Properties of a data structure always hold for all operations
  - Algorithm's running time / memory will never exceed some threshold
  - Algorithm will always be correct
  - Algorithm will always terminate
- Proof techniques
  - Proof by induction
  - Proof by counterexample
  - Proof by contradiction

WSU

Proof by induction

- Goal: Prove some hypothesis is true
- Three-step process
  - Base case: Show hypothesis is true for some initial conditions ($k=1$)
  - Inductive hypothesis: Assume hypothesis is true for all cases $\leq k$
  - Inductive step: Show hypothesis is true for next larger value (typically $k+1$)

WSU

Inductive proof: example

- Prove arithmetic series
  - $\sum_{i=0}^{N} i = N(N+1)/2$
- Base case: show it is true for $N = 0$
  - $\sum_{i=0}^{N} i = 0 \Leftrightarrow \frac{N(N+1)}{2} = 0$
- Inductive hypothesis: assume it is true for all $N \leq k$: $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$
- Inductive step: generalize to $k+1$ to see whether $\sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$

WSU

Inductive proof: example

- Prove arithmetic series
  - $\sum_{i=0}^{N} i = N(N+1)/2$
- Base case: show it is true for $N = 0$
  - $\sum_{i=0}^{N} i = 0 \Leftrightarrow \frac{N(N+1)}{2} = 0$
- Inductive hypothesis: assume it is true for all $N \leq k$: $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$
- Inductive step: generalize to $k+1$ to see whether $\sum_{i=0}^{k+1} i = \boxed{\frac{(k+1)(k+2)}{2}}$
  - $\sum_{i=0}^{k+1} i = \sum_{i=0}^{k} i + (k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)\left(\frac{k}{2}+1\right) = \boxed{\frac{(k+1)(k+2)}{2}}$

Inductive proof: example

- More examples:
  - Prove the geometric series
    - $\sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$
  - Prove that the number of nodes N in a complete binary tree of depth D is $2^{D+1} - 1$

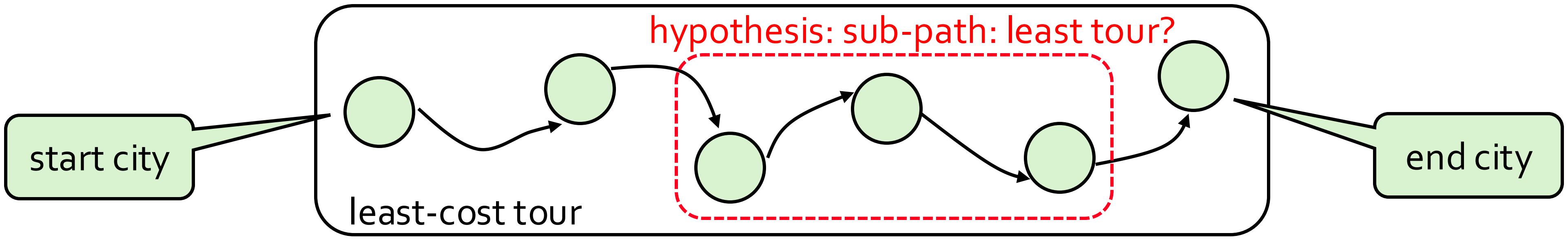
WSU

Proof by counterexample

- Prove hypothesis is not true by giving an example that does not hold
  - Example: how to prove $2N > N^2$?
    - Proof by letting $N = 2$ (or 3 or 4)

Counterexample: example

- Given N cities and costs for traveling between each pair of cities, a "least-cost tour" is one which visits every city exactly once with the least cost
- Hypothesis: Any sub-path within any least-cost tour will also be a least cost tour for those cities included in the sub-path.
- Is this hypothesis true?

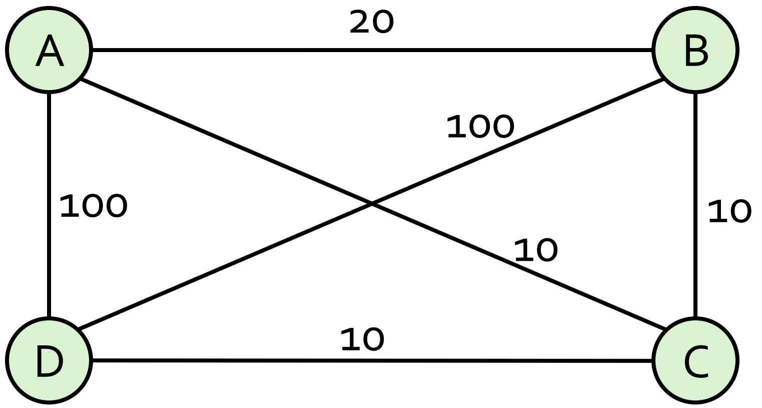


Counterexample: example

- Counterexample
  - Cost (A→B→C→D) = 40: least cost tour for {A, B, C, D}
  - What is the least cost tour for {A, B, C}?
    - Cost (A→B→C) = 30
    - Cost (A→C→B) = 20

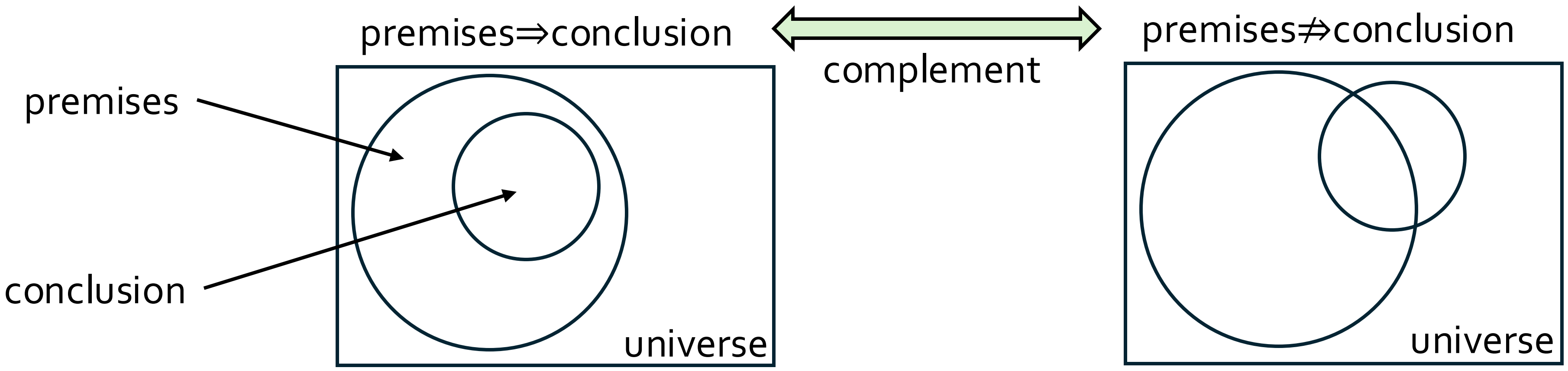
not the least cost tour

the least cost tour

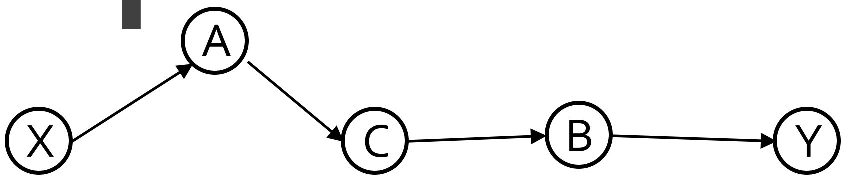


Proof by contradiction

- Start by assuming that the hypothesis is false
- Show this assumption could lead to a contradiction (i.e., some known property is violated)
- Therefore, hypothesis must be true



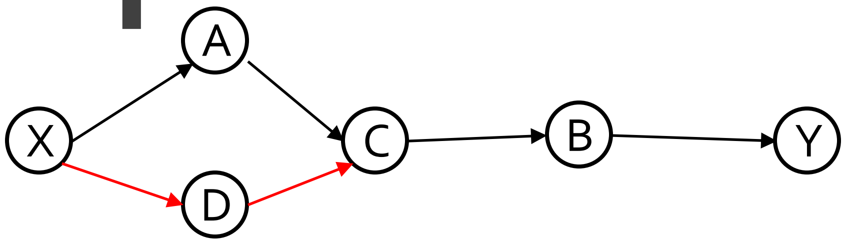
Contradiction: example



Hypothesis: P=X→A→C→B→Y is least-cost

- Single pair shortest path problem
  - Given N cities and costs for traveling between each pair of cities, find the least-cost path to go from city X to city Y
- Hypothesis: A least-cost path from X to Y contains least-cost paths from X to every city on the path
  - E.g., if X→A → C → B → Y is a least-cost path from X to Y, then
    - X → A → C → B must be a least-cost path from X to B
    - X → A → C must be a least-cost path from X to C
    - X → A must be a least-cost path from X to A
- Conclusion: Least cost paths should contain smaller least cost paths starting at the source

Contradiction: example



Hypothesis: P=X→A→C→B→Y is least-cost

P'=X→A→D→B→Y

- Let P be a least-cost path from X to Y
- Now, assume that the hypothesis is false:
  - ==> there exists C along X->Y path, such that, there is a better path from X to C than the one in P
  - ==> So we could replace the subpath from X to C in P with this less-cost path, to create a new path P' from X to Y
  - ==> Thus we now have a better path from X to Y
  - i.e., cost(P') < cost(P)
  - ==> But this violates the fact that P is a least-cost path from X to Y (hence a contradiction!)
- Therefore, the original hypothesis must be true

WSU

Recurrence V.S. recursion

- A recursive function or a recursive formula is defined in terms of itself
- Example:

$$n! = \begin{cases} 1 \text{ if } n = 0 \\ \\ n * (n-1)! \text{ if } n > 0 \end{cases}$$

Mathematical recurrence

```
Factorial (n)
{
if (n = 0)
      then return 1;
else
      return (n *
Factorial (n-1));
}
```

Recursion (code)

Basic rules of recursion

- Base cases
  - Must always have some base cases, which can be solved without recursion
- Making progress
  - Recursive calls must always make progress toward a base case
- Design rule
  - Assume all recursive calls work
- Compound interest rule
  - Try not to duplicate work by solving the same instance of a problem in separate recursive calls

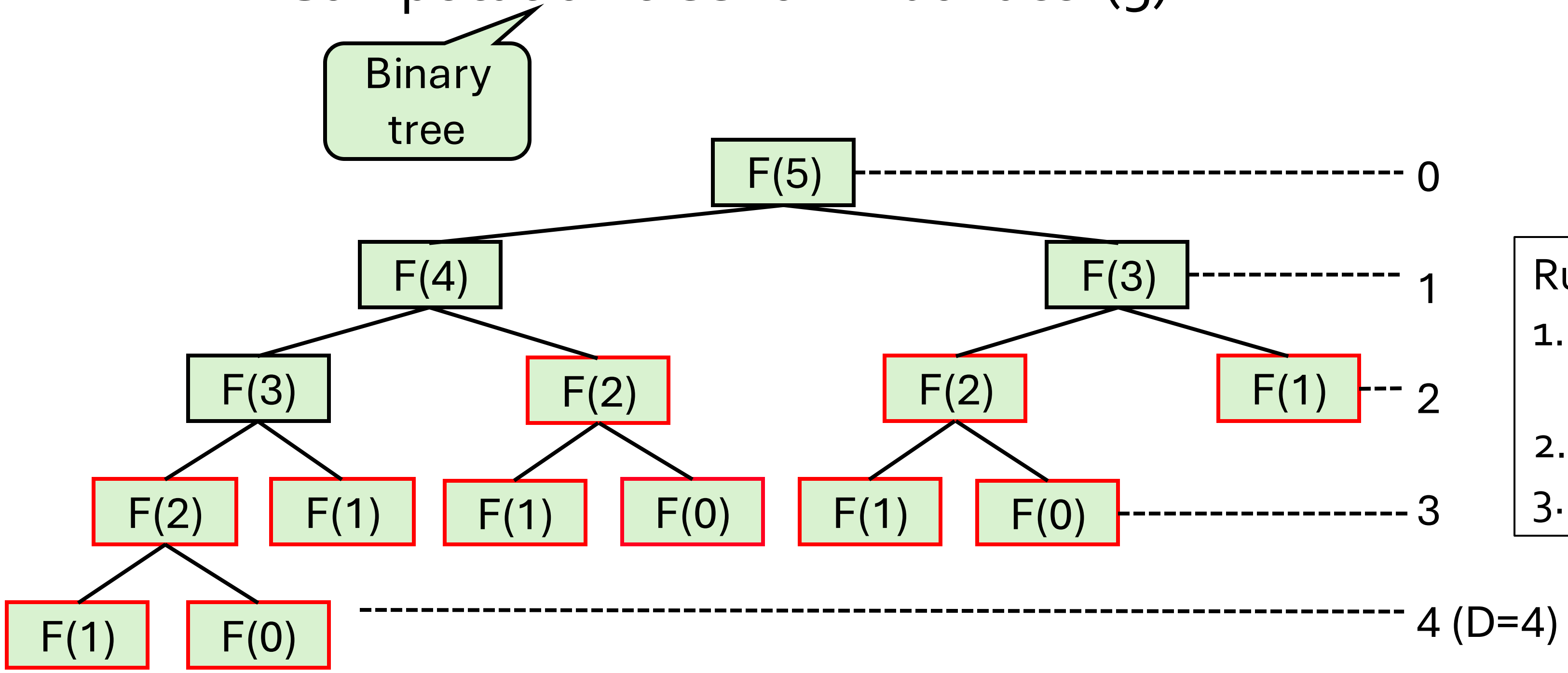
Recursion: example

- Fibonacci numbers
  - $F(0) = 1$
  - $F(1) = 1$
  - $F(n) = F(n-1) + F(n-2)$

```
Fibonacci (n)
{
   if (n ≤ 1)
      then return 1
   else return (Fibonacci (n-1) + Fibonacci (n-2))
}
```

Recursion: example

- Computation tree for: Fibonacci (5)



Runtime for the recursive code

1. proportional to the size of the tree (and that is a lot wasteful)
2. (#nodes for Depth D) ≤ 2^(D+1) – 1
3. n=D+1

Running time for Fibonacci(n)?

- Show that the running time $T(n)$ of Fibonacci(n) is exponential in n, Depth $D = n - 1$
- This only gives an upper bound for $T(2^n)$
  - We also need to prove that T(n) is at least exponential
- Use mathematical induction
  - A tighter upper bound: $T(n) < (5/3)^n$for n>=1

WSU

Solving recurrences

- Example:

```
Algo1(A,1,n)
    // A is an integer array of size n
    if(n<2) return;
    x = floor(n/2);
    Algo1(A,1,x);
    Algo1(A,x+1,n);
```

- How much time does Algo1 take?
  - Express time as a function of n (input size)
- Let T(n) be the time taken by Algo1 on an input size n
- Then: $T(n) = 1 + T(n/2) + T(n/2)$

  $= 2T(n/2) + 1$

Solving recurrences

- Recurrence:
  - T(n) = 2T(n/2) + 1
  - T(1) = 1
- Solution (via geometric sequence)

T(n) = 2T(n/2) + 1
= 4T(n/4) + 2+ 1
= 8 T(n/8) + (4+2+1)
= …… (K steps)
= $2^K\ T(n/2^K) + \sum_{k=0}^{K-1} 2^k$
= n T(1) + n-1
= 2n-1 [Closed-form]

Expand K times
such that $n/2^K = 1$

Lower bound for Fibonacci(n)?

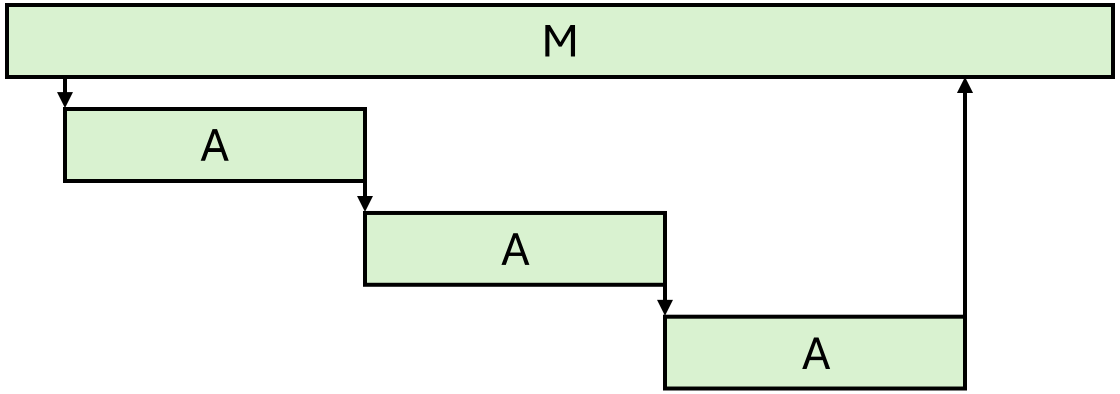
- T(n) = T(n-1) + T(n-2)
- Assume T(n-1) ~ T(n-2), approximation
  = 2T(n-2)
  = 2*(2T(n-4))
  = 4T(n-4)
  = 8T(n-6)
  = 2k * T(n - 2k)
- For termination, n = 2k, k = n/2
  - T(n) = 2^(n/2) * T(0)
    T(n) = O(2^(n/2))

WSU

Recurrence V.S. recursion

- Recurrence vs. Recursion
  - A recurrence need not always be implemented using recursion
  - How?

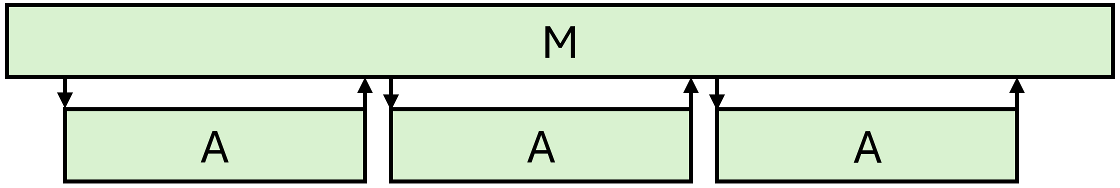
Tail recursion → iteration



tail recursive code

```
A(n){
   …
   A(n-1)
}
```

The result of A(n-1) is NOT needed/used in A(n)



while or for loop

```
A(n){
   for (i=n; i>=0; i--)
   {……}
}
```

WSU

Example: print 1 to n

```
void print(int n)
{
    if (n < 0) return;
    cout << " " << n;
    print(n-1);
}
```

```
void print(int n)
{
    for (int i = n; i > 0; i--) {
        cout << " " << n;
    }
}
```

Same time complexity, but this one is better!

Iterative code is usually more efficient than tail recursive code:
Recursion requires more overhead due to function calls (Stack frame)

Then why still using recursion?
Recursion: readable code, easy to understand, well suit divide-and-conquer algorithms

Fibonacci number using loop?

WSU

Recursive function calls

```
M()
{
  A()
  A()
  B()
  C()
}
```

```
A()
{
  ......
}
```

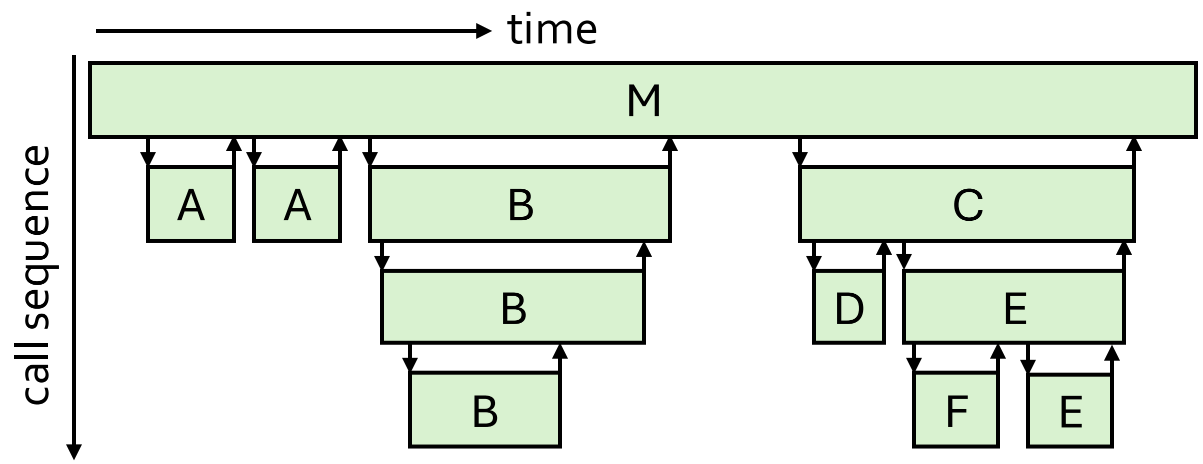
```
B()
{
  B()
}
```

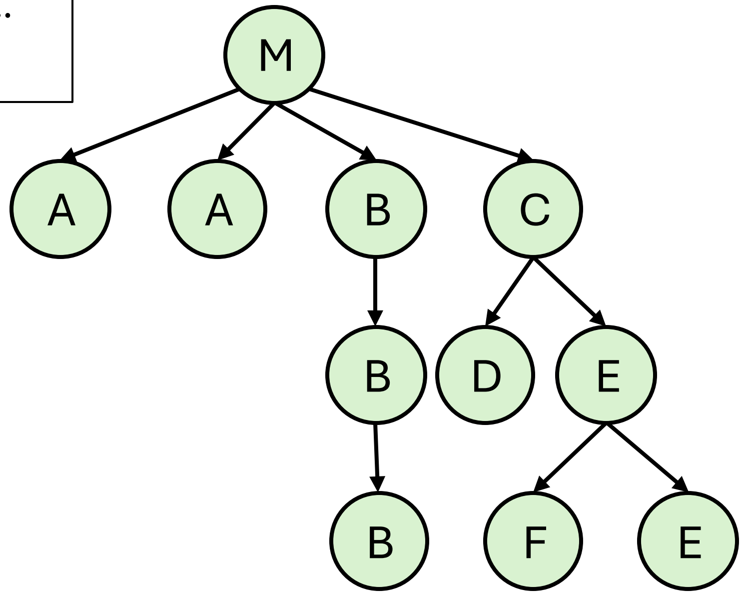
```
C()
{
  D()
  E()
}
```

```
D()
{
  ......
}
```

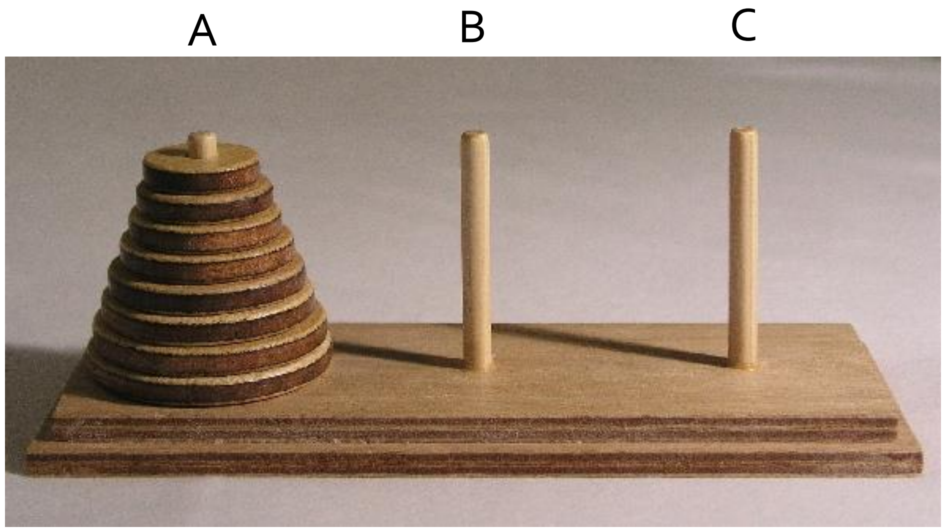
```
E()
{
  F()
  E()
}
```

```
F()
{
  ......
}
```





Recursion: example



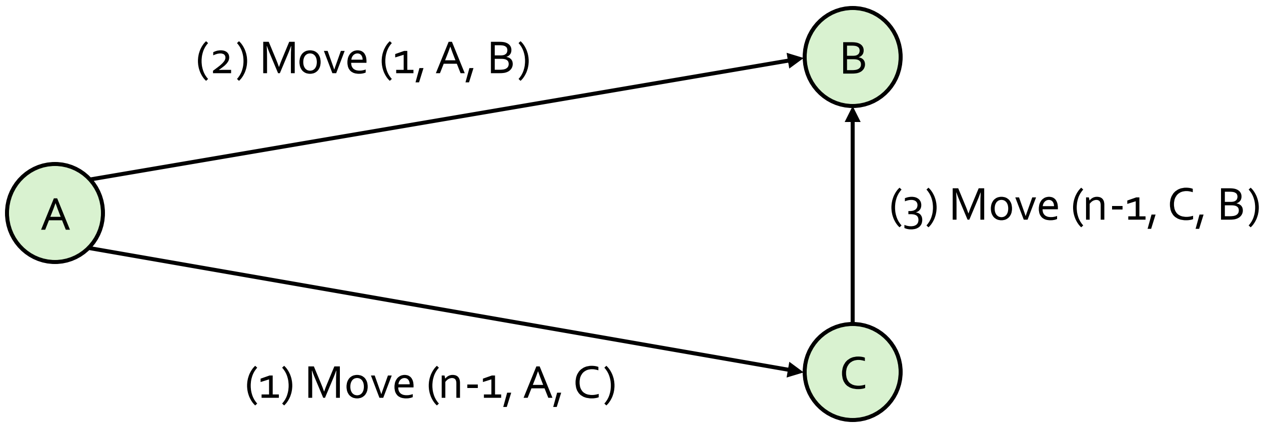
- Tower of Hanoi
  - Goal:
    - Move all disks from peg A to peg B using peg C
  - Rules:
    - Move one disk at a time
    - Larger disks cannot be placed above smaller disks
- Question: What is the minimum number of moves necessary to solve the problem?

Image credit: https://en.wikipedia.org/wiki/Tower_of_Hanoi#/media/File:Tower_of_Hanoi.jpeg

WSU

Recursion: example

- A Recursive Algorithm: Find the base case -> N=1
  - Move the top n-1 disks, "recursively", from A to C (using B)
  - Move nth disk (i.e., largest & bottom-most in A) from A to B
  - Move all the n-1 disks, "recursively", from C to B (using A)



WSU

Recursion: example

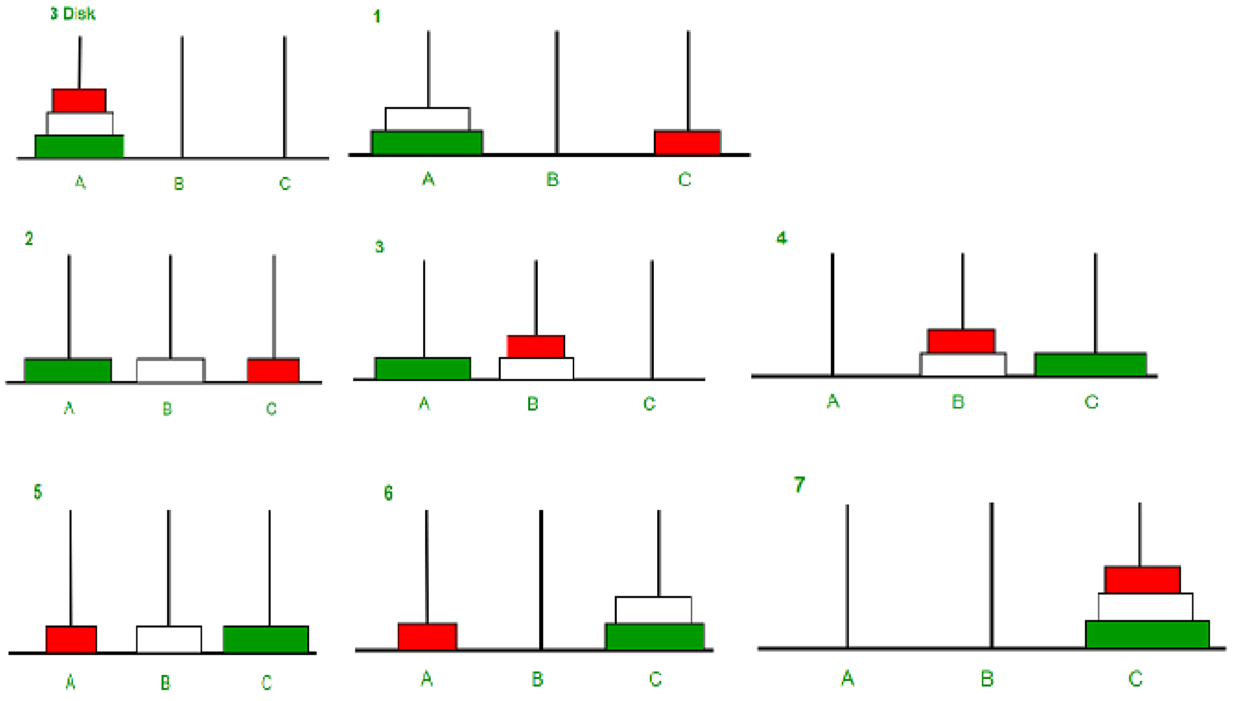


Image credit: https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/

Recursion: example

- Pseudocode
  - Move (n, A, B, C)
  - PRE: n disks on A; B and C unaffected
  - POST: n disks on B; A and C unaffected
  - BEGIN
    - IF n=0 THEN RETURN
    - Move (n-1, A,C,B)
    - Move nth disk from A to B directly
    - Move (n-1,C,B,A)
  - END

Recursion: example

- Define: T(n)= minimum required number of moves
- Analysis: starts with T(1)=1
  - T(n) = 2T(n-1)+1
- Solving this: T(n)=2^n-1 (how?)
- In the original Tower of Hanoi problem, n=8 & so T(n)=255
- For Tower of Brahma, n=64
  - 2^{64}-1 moves made by a priest in a temple
  - Assuming each move takes 1 second, this would take 5,000,000,000 centuries to complete
- Online game: https://www.mathsisfun.com/games/towerofhanoi.html

WSU

Summary

- Floors, ceilings, exponents, logarithms, series, and modular arithmetic
- Proofs by mathematical induction, counterexample and contradiction
- Recursion and solving recurrences
- Tools to help us analyze the performance of our data structures and algorithms