Basics of Programming

L10: Recursion

Apr 2020

Dr. Ram P Rustagi
Professor, CSE Dept
KRP, KSGI
rprustagi@ksit.edu.in

Resources and Acknowledgements

- Python for everybody
  - https://www.py4e.com
- A first course in programming
  - https://introcs.cs.princeton.edu/python/20functions/
- netacad.com: Python Essentials:

https://780671818.netacad.com/courses/1004579/modules/items/66720226

Exercises - 1

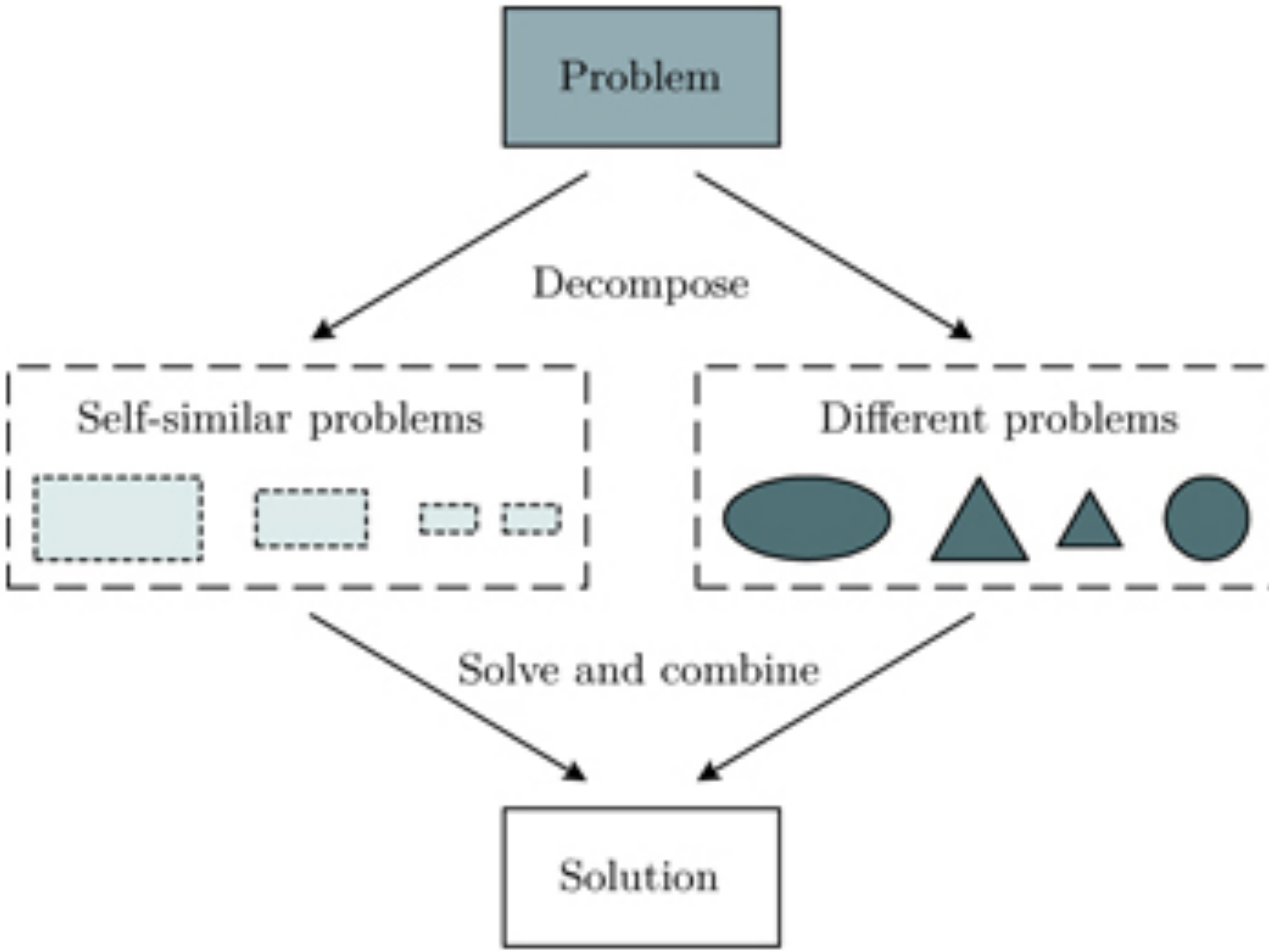
- For the following algorithms (problems), identify
  - Natural input size metric
  - Basic operation
  - Count of basic operation (average case)

- P 01: Computing sum of `n` numbers
- P 02: Computing `factorial(n)`
- P 03: Finding largest element of `n` numbers

Basics of Recursion

- Look at Nature entities
  - Trees
  - Natural patterns
- Matryoshka dolls
  - https://en.wikipedia.org/wiki/Matryoshka_doll

Sub-Problems



src: Intro to Recursive programming

Reduction to a Simpler Problem

- Typically works where induction methods are used to show correctness proof
- Recursive functions take this appproach
- Example: Compute `n2 = f(n)`

  $n^2 = (n-1)^2 +$ Linear function of n.

  $= n-1)^2 + 2(n-1)+1$

  $=(n-1)^2 +2n-1 = f(n-1)+ (2n-1)$
- Example: factorial `n! = f(n)`

  `n! = n*(n-1)! = n*f(n-1)`
- Example: GCD computation

  `gcd(y,x)= gcd(x,r)` where `r=y%x`

Divide and Conquer Approach

- Divide (break) the problem (size `n`) into similar sub problems
  - Sub problems should be of smaller size. Decrease by a const or some factor of original (e.g. `n/c`)
    - When small enough, solve by brute force
- Conquer (solve) the sub-problem
  - Use recursion to solve small problem
- Combine (Merge) the solution of sub-parts
- The cost is
  - cost of breaking
  - cost of solving subproblem
  - cost of combining

Recursion

- A broad concept present in many fields
  - Comp Sc, Bio-informatics, mathematics etc.
  - Nature examples
    - Trees, rivers, Chinese dolls, art patterns
  - Helps in devleoping simple, succinct, elegant algorithms to solve computational problems
- Recursion process
  - Repeat itself till comes to a small problem
    - Small problem is solved by some means
- Understanding recursion
  - A process of defining concepts using the def^{n} itself
  - Fibbonacci numbers:

$$S_n = S_{n-1} + S_{n-2}$$

Recursive Sum

```
def sum(n):
   if n==1:
      return a[1]
   else
      return a[n]+sum(n-1)
________________
a[]=[11,8,17,5,9]
sum(5)
 =9+sum(4)
 =9+(5+sum(3))
 =9+(5+(17+sum(2)))
 =9+(5+(17+(8+11)))=9+(5+(17+(19)))
 =9+(5+(36))=9+(41) = 50
```

Recursive Square Computation

```
def square(n):
   if n==1:
      return 1
   else
      return 2*n-1+square(n-1)
________________
square(5)
 =9+square(4)
 =9+(7+square(3))
 =9+(7+(5+square(2)))
 =9+(7+(5+(3+square(1)))
 =9+(7+(5+(3+1))) = 9+(7+(5+4))
 = 9+(7+9)=9+(16) = 25
```

Recursive Permutation

```
def perm(n,r): = n!/(n-r)!
   if n==r:
      return n
   else
      return n * perm(n-1,r)
___________________
perm(5,3)
 =5*perm(4,3)
 =5*4*perm(3,3)
 =5*4*3
 =60
```

Finding Maximum Element

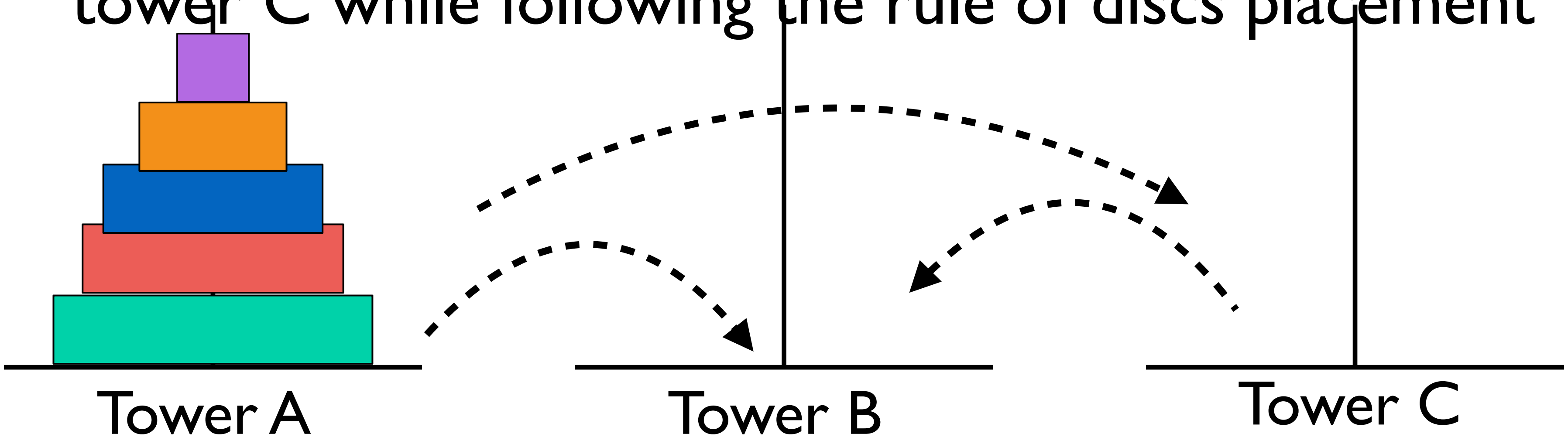
- Recursive approach to find max
- **Prog:** `FindMax(A[1:n])`

```
// Input: array A
// Output: The value of largest element
if len(A)==1
  return A[1]
else
  max = FindMax(A[1:n-1]
  if max > A[n]
    return max
  else
    return A[n]
```

- **Efficiency:** `O(n)`

Tower of Hanoi

- Task: Tranfer `n` discs from tower A to tower B using tower C while following the rule of discs placement

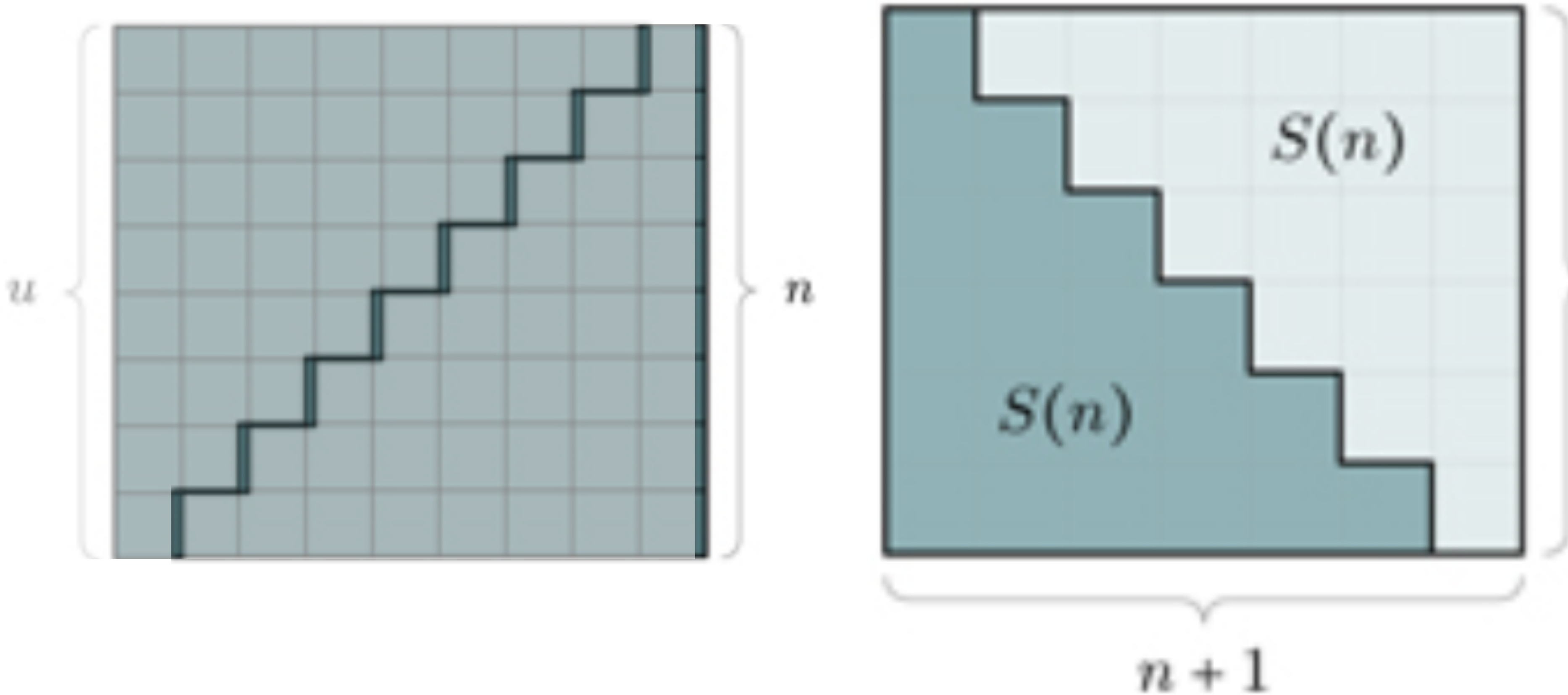


- Efficiency: Basic operations: Move `(n-1),1,(n-1)`

$$
\begin{aligned}
T(n) &= T(n-1)+1+T(n-1)\\
&= 1+2*T(n-1)\\
&= 1+2(1+2*T(n-2))\\
&= 2^0+2^1+2^2+\ldots+2^{n-1} = 2^n - 1\\
&= \Theta(2^n)
\end{aligned}
$$

Recursive Problem Solving

- Problem: Find sum S(n) of first N positive integers
- Non-recursive approach



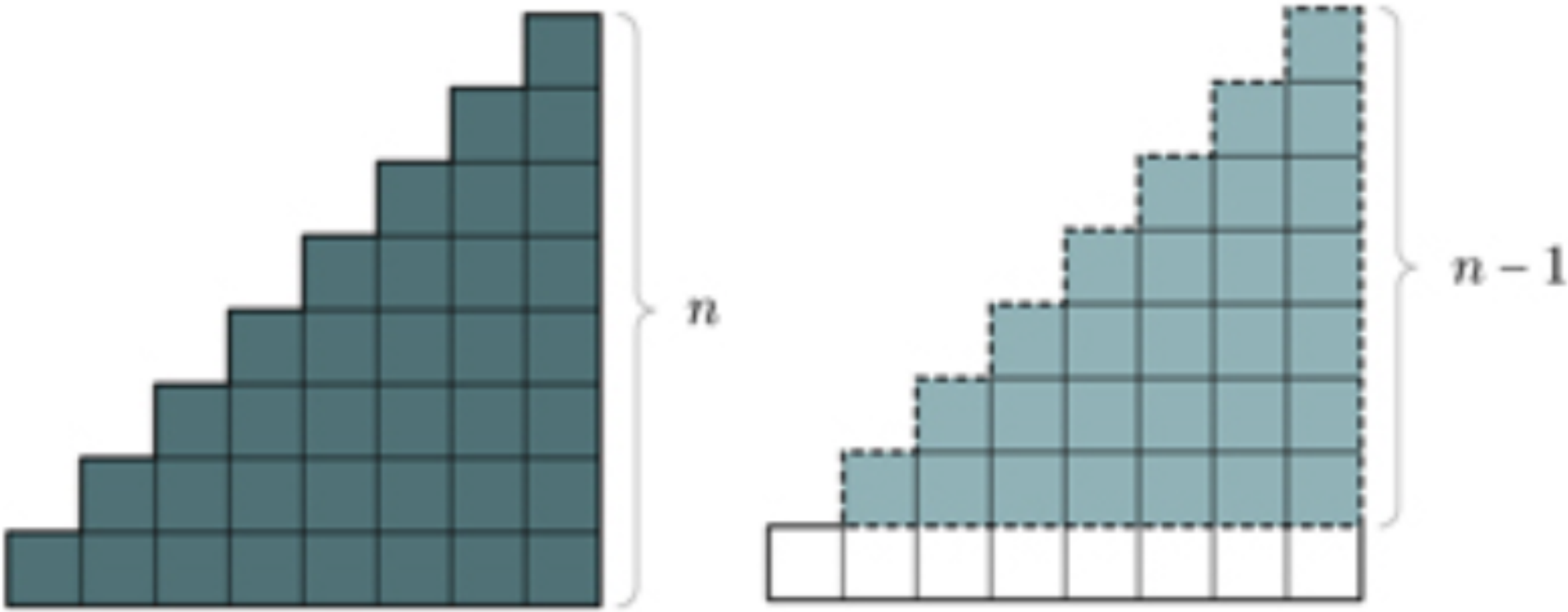
Non-recursive Approach: `2S(n)=n(n+1)`

`—>S(n)  =  n(n+1)/2`

src: Intro to Recursive programming: Rubio Sanchez (author)

Recursive Problem Solving

- Problem: Find sum S(n) of first N positive integers



Approach 1: `S(n)=S(n-1)+n`

src: Intro to Recursive programming: Rubio Sanchez (author)

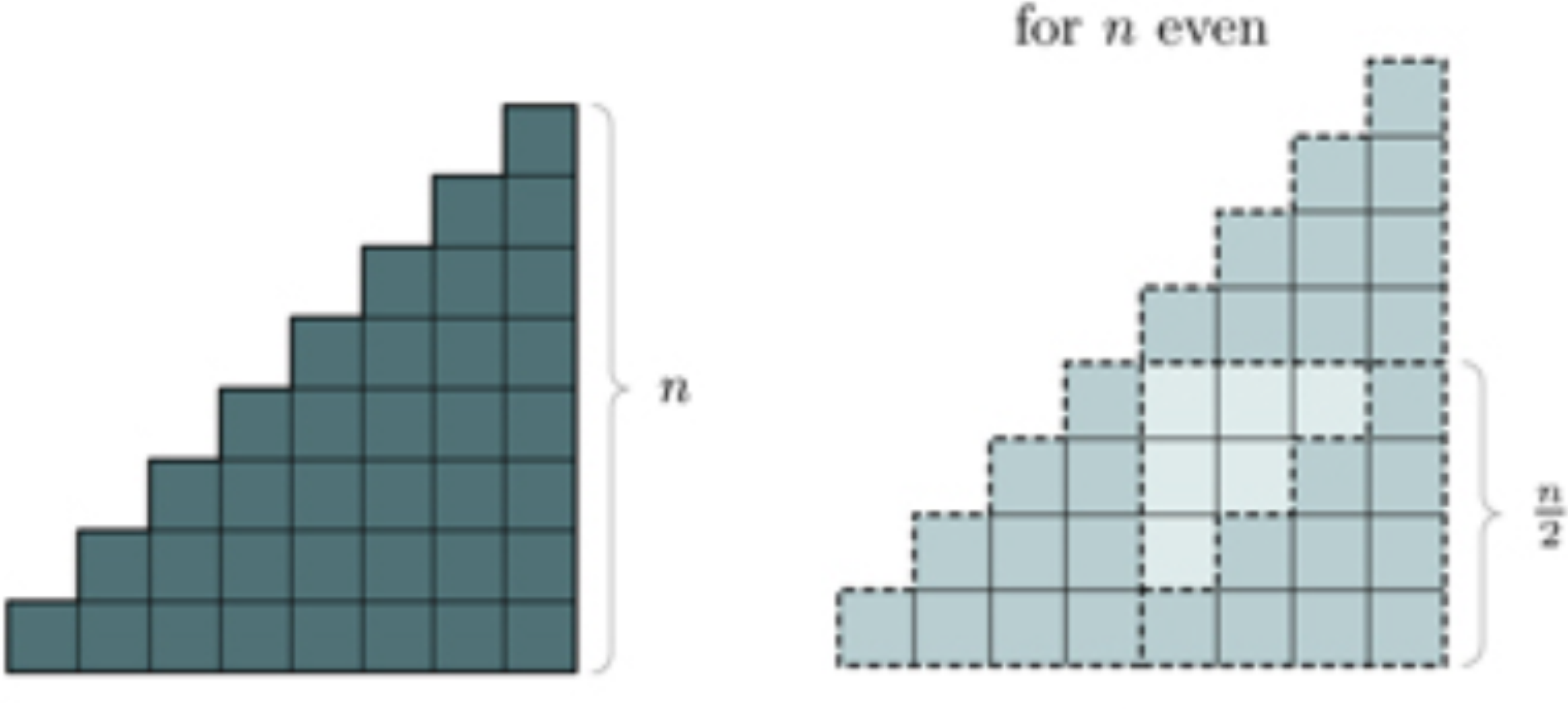
Recursive Problem Solving

- Problem: Find sum `S(n)` of first `N` positive integers
- Demo:
  - Get few students on stage.
  - Assign them the numbers (in order of appearance)
    - S_1: N^{th} student passes the subproblem (`n-1`) to next
    - S_2: When gets the answer, adds its number
    - S_3: Display the result
    - Step S_1 and S_2 keeps repeating till `n = 1`

src: Intro to Recursive programming: Rubio Sanchez (author)

Recursive Problem Solving

- Problem: Find sum S(n) of first N positive integers



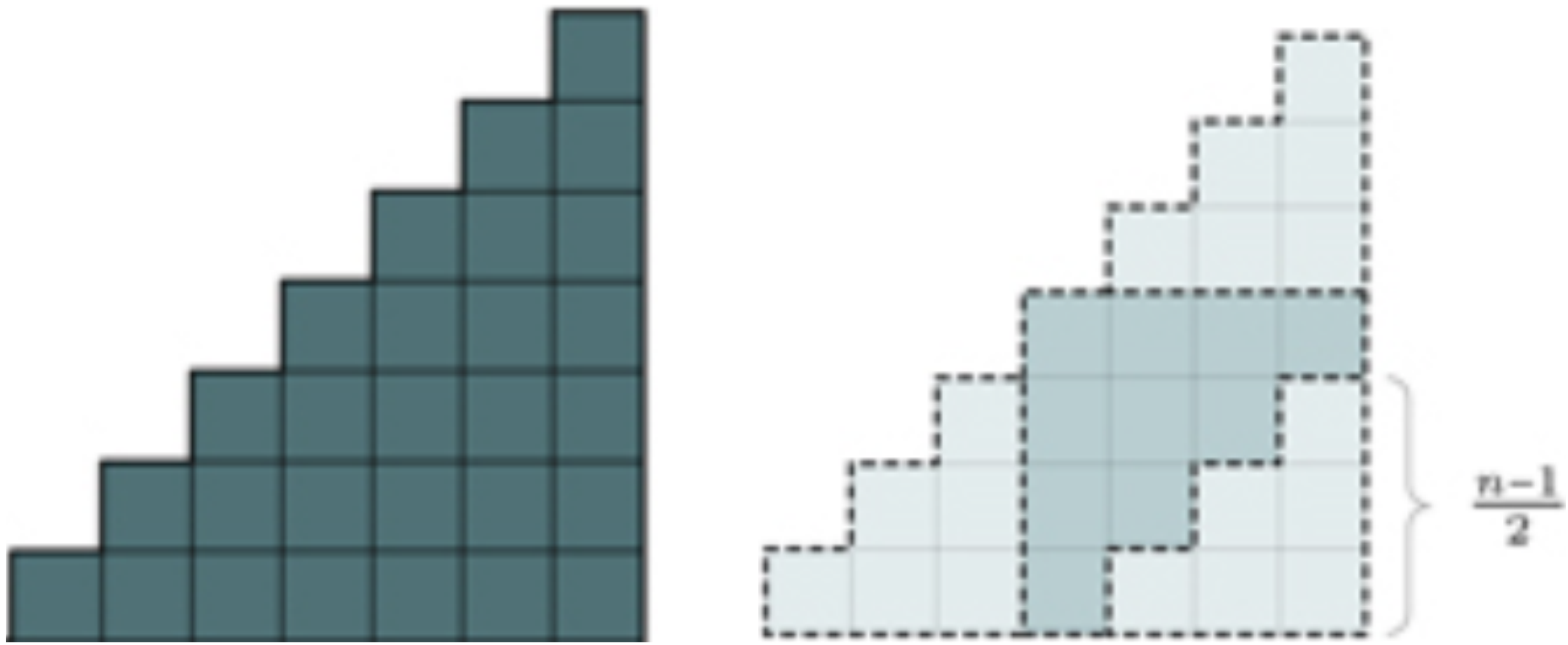
Approach 2a: When n is even

$$S(n)=3S(n/2)+S(n/2\ -1)$$

src: Intro to Recursive programming: Rubio Sanchez (author)

Recursive Problem Solving

- Problem: Find sum S(n) of first N positive integers



Approach 2b: When n is odd

$$S(n)=3S((n-1)/2)+S((n+1)/2)$$

src: Intro to Recursive programming: Rubio Sanchez (author)

Recursive Problem Solving

- Problem: Find sum S(n) of first N positive integers
- Recursive function

$$
S(n) = \begin{cases} 1 \text{ if } n = 1 \\ 3 \text{ if } n = 2 \\ 3S(\frac{n}{2}) + S(\frac{n}{2} - 1), & n > 2 \text{ and } n \text{ is even} \\ 3S(\frac{n-1}{2}) + S(\frac{n+1}{2}), & n > 2 \text{ and } n \text{ is odd} \end{cases}
$$

- Problem: This definition needs two base cases
    - For n=1, and n=2

src: Intro to Recursive programming: Rubio Sanchez (author)

Writing Recursive Code

- Python program for recursive approach

```
def sum(n):
  if (n==1):
    return 1
  elif (n==2):
    return 3
  elif (n%2 == 0): #even
    return 3*sum(n//2)+sum((n//2)-1))
  else:
    return 3*sum((n-1)//2)+sum((n+1)//2)
```

- Invocation of Python program for approach 2

```
sum(n)  #e.g. sum(20)
```

Summary

- Recursive approach
  - Break the given problem into smaller problems
  - Find the smallest size problem that you can solve
  - Merge the solution from smaller problem solve bigger problems

Exercise 01

- Compute ${}^{n}C_{r}$:

  $${}^{n}C_{r} = {}^{n-1}C_{r-1} + {}^{n-1}C_{r}$$

  – smallest problems that can be solved

  $${}^{n}C_{1} = 1; \; {}^{n}C_{n}=1$$

Exercise 02

- Recursive prog to find max/min of N numbers
  - Two approaches
    - First: decrease by 1,
    - 2^{nd}: divide into 2 equal (half) parts

Exercise 03

- Recursive prog to find Fibonacci Number F(n)
  – $F(n) = F(n-1) + F(n-2)$

Exercise 04

- Recursive prog to Computer `power(n,k)=n`^{k}
  - First approach decrease by 1,

    `P(n,k)=n*P(n,k-1)`

    `P(n,1) = n`
  - 2nd approach: divide into 2 equal (half) parts
    - if k is even

      `P(n,k)=P(n,k/2)*P(n,k/2)`
    - if `k` is odd

      `P(n,k)=n*P(n,(k-1)/2)*P(n,(k-1)/2)`
  - 3rd approach: divide into 2 equal (half) parts

    `P(n,k)=x*x;  x=P(n,k/2)`, k is even

    `P(n,k)=n*x*x;  x=P(n,(k-1)/2)`, k is odd

Questions