

RECURSIVE FORMULATIONS TO ALGORITHM DESIGN



Partha P Chakrabarti
Indian Institute of Technology Kharagpur

Overview

- Algorithms and Programs
 - Pseudo-Code
 - Algorithms + Data Structures = Programs
 - Initial Solutions + Analysis + Solution Refinement + Data Structures = Final Algorithm
 - Use of Recursive Definitions as Initial Solutions
 - Recurrence Equations for Proofs and Analysis
 - Solution Refinement through Recursion Transformation and Traversal
 - Data Structures for saving past computation for future use
- Time } complexity
Space }

Sample Problems:

1. Finding the Largest
2. Largest and Smallest
3. Largest and Second Largest
4. Fibonacci Numbers
5. Searching for an element in an ordered / unordered List
6. Sorting
7. Pattern Matching
8. Permutations and Combinations
9. Layout and Routing
10. Shortest Paths

First Problem: Largest of a Set of Numbers

Sequential Comparison

$L = \{x_1, x_2, \dots, x_n\}$ x is an integer

$\max(L)$

$\{L = \{x_1, x_2, \dots, x_n\}\}$

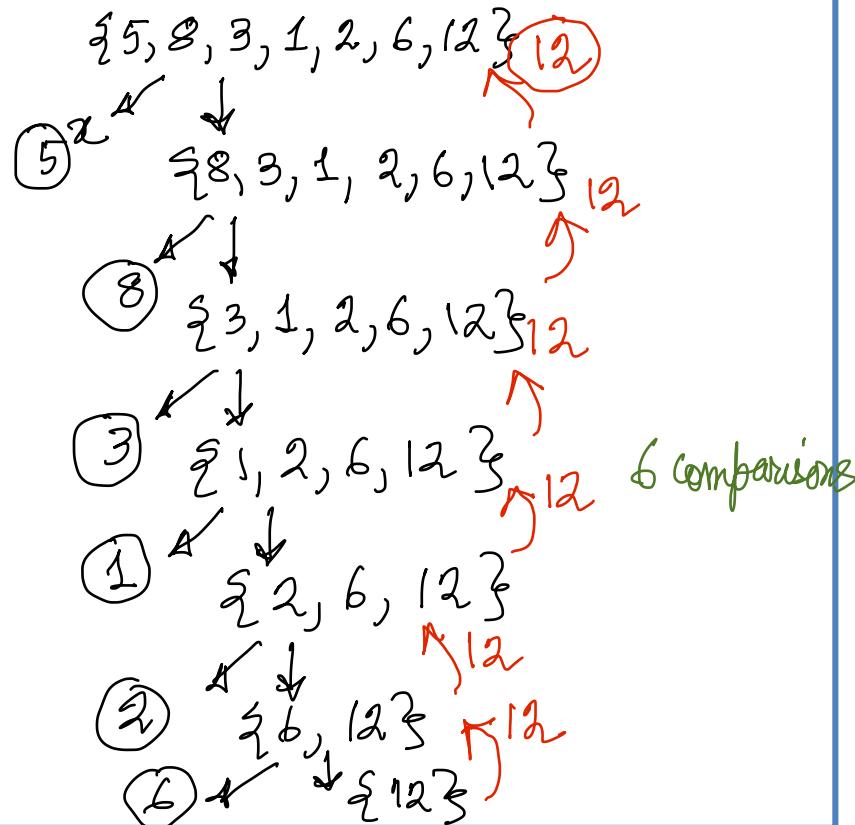
if $|L| = 1$ return (x_1)

$L' = L - \{x_1\}$

$y = \max(L')$

if $(x_1 \geq y)$ return (x_1)
else (y)

$$\begin{aligned} T(n) &= T(n-1) + 1, n > 1 \\ &= 0, \text{ if } n = 1 \\ &= (n-1) \end{aligned}$$



Finding Largest: Recursive Formulation

$\text{max2}(L)$

$\{ L = \{x_1, x_2, \dots, x_n\} \}$

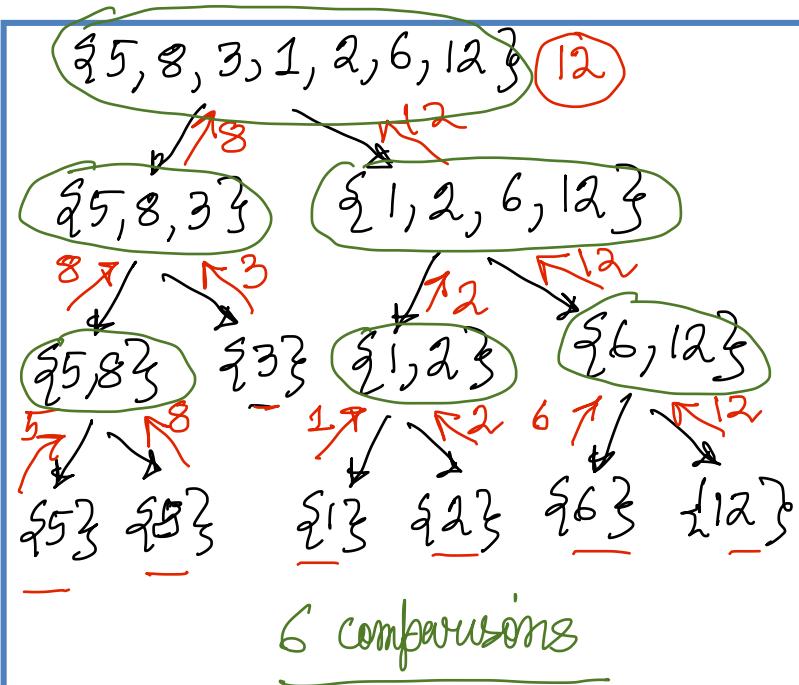
If $|L| = 1$ return (x_1) ✓
split L into 2 non-empty sets L_1, L_2

$\rightarrow y_1 = \text{max2}(L_1)$ } ✓

$\rightarrow y_2 = \text{max2}(L_2)$ }
If $(y_1 \geq y_2)$ return (y_1)
else return (y_2)

} correctness proof by INDUCTION

$$\begin{aligned} T(n) &= T(k) + T(n-k) \quad n > 1 \\ &= 0, \text{ if } n=1 \\ &= (n-1) \end{aligned}$$



Largest: Analysis

Proof of Correctness

By induction:

Base condition $n = 1$

Inductive condition

correctly for all $n < n_0$

We prove inductively
it is true for $n_0 + 1$

True for all $n \geq 1$

Complexity Analysis

$$T(n) = T(k) + T(n-k) + 1$$

for $n > 1$

$$= 0, n = 1$$

$$T(x) = x - 1$$

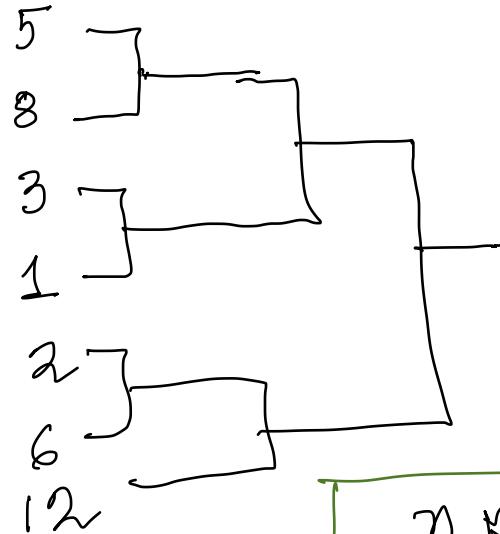
$$T(n) = (k-1) + (n-k-1)$$

+ 1

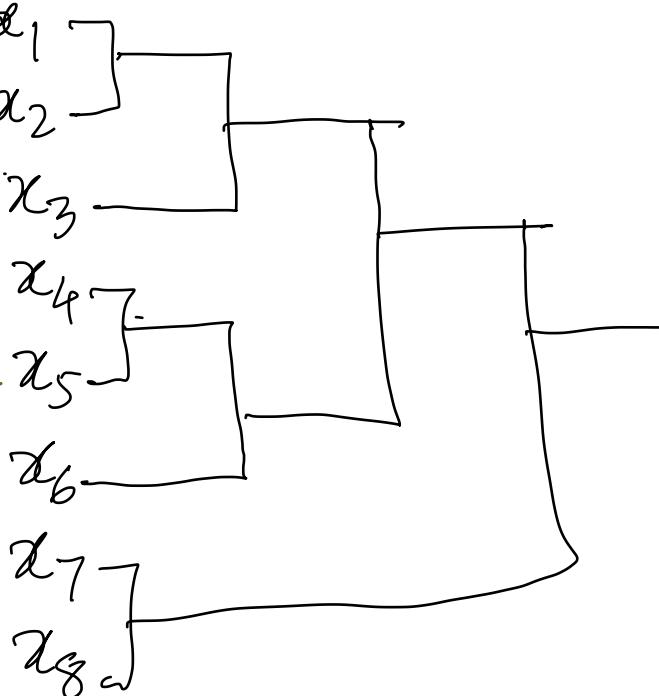
$$= \underline{n-1}$$

Comparison Tournament

5, 8, 3, 1, 2, 6, 12



n players
 $(n-1)$ matches



2nd Problem: Largest and Smallest

Sequential Comparison

maxmin (L)

{ let $L = \{x_1, x_2, \dots, x_n\}$

if $|L| = 1$ return $\langle x_1, x_1 \rangle$

$L' = L - \{x_1\}$

$\langle y_1, y_2 \rangle = \text{maxmin}(L')$

if $(x_1 > y_1)$ then $m_1 = x_1$
 else $m_1 = y_1$

if $(x_1 \leq y_2)$ then $m_2 = x_1$
 else $m_2 = y_2$

return $\langle m_1, m_2 \rangle$

$$T(n) = 0, \text{ if } n=1$$

$$= T(n-1) + 2, n > 1$$

$$= 2(n-1)$$

$\{5, 8, 3, 1, 2, 6, 12\} \quad \langle 12, 12 \rangle$

5 ↓ ↗ $\langle 12, 1 \rangle$

$\{8, 3, 1, 2, 6, 12\}$

8 ↓ ↗ $\langle 12, 12 \rangle$

$\{3, 1, 2, 6, 12\}$

3 ↓ ↗ $\langle 12, 1 \rangle$

$\{1, 2, 6, 12\}$

1 ↓ ↗ $\langle 12, 2 \rangle$

$\{2, 6, 12\}$

2 ↓ ↗ $\langle 12, 6 \rangle$

$\{6, 12\} \quad \langle 12, 12 \rangle$

6 ↓ ↗ $\langle 12 \rangle$

Largest and Smallest: Recursive Definition

$\text{maxmin2}(L)$

{ Let $L = \{x_1, x_2, \dots, x_n\}$

if $|L| = 1$, return (x_1, x_1)

if $|L| = 2$, if $(x_1 > x_2)$

return (x_1, x_2)

else return (x_2, x_1)

split L into 2 non-empty sets L_1, L_2

$(y_1, y_2) = \text{maxmin2}(L_1)$

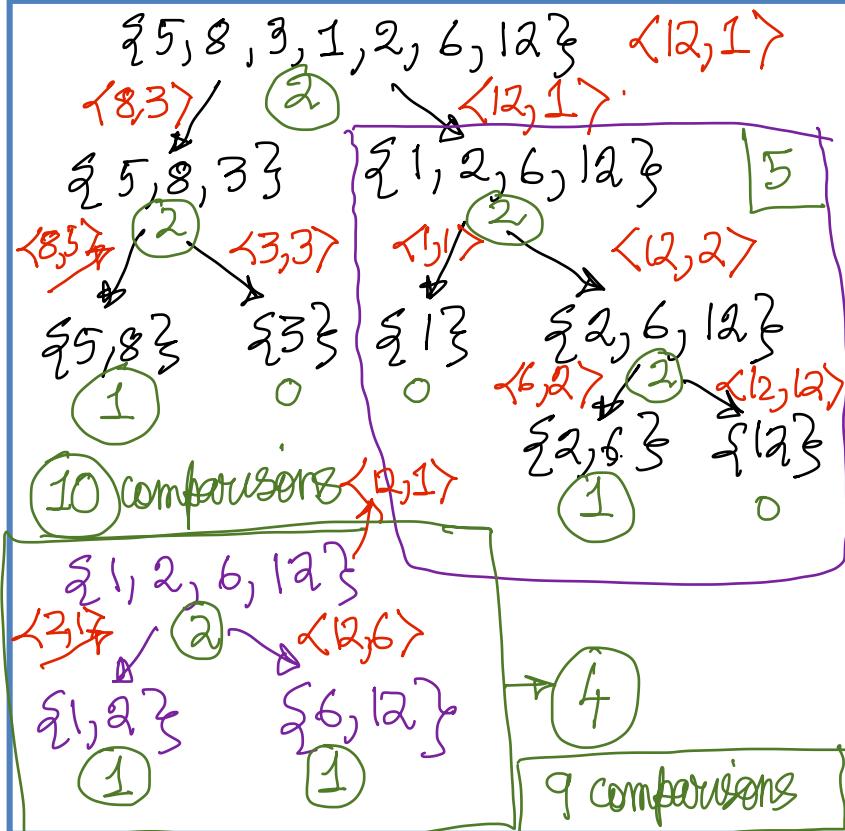
$(z_1, z_2) = \text{maxmin2}(L_2)$

if $(y_1 > z_1)$ $m_1 = y_1$ else $m_1 = z_1$

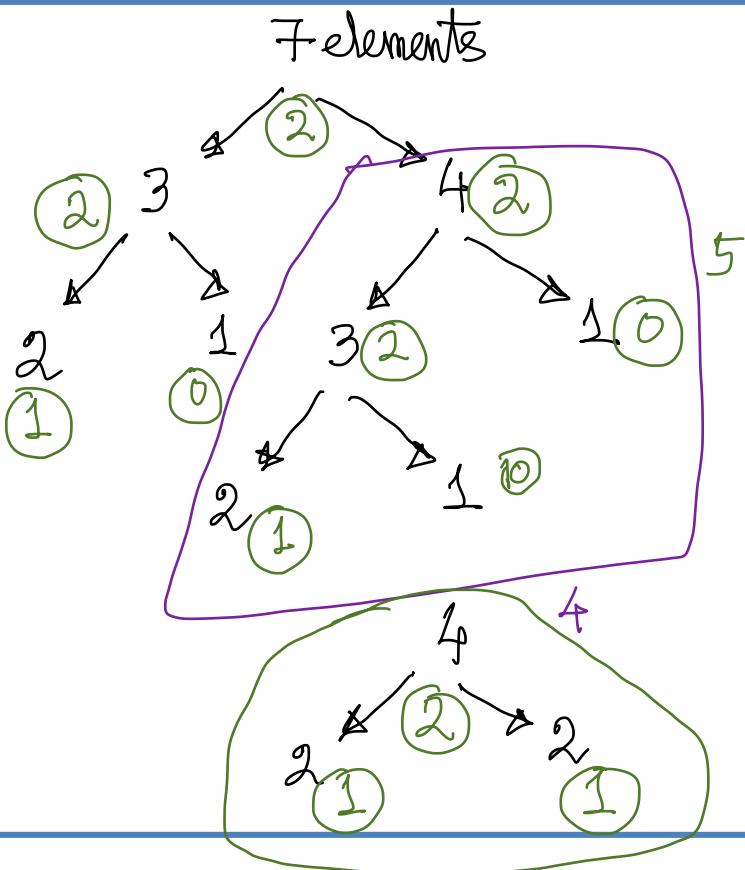
if $(y_2 \leq z_2)$ $m_2 = y_2$ else $m_2 = z_2$

return (m_1, m_2)

}



Largest & Smallest: Choice of Split



$$\begin{aligned} n &\xrightarrow{\text{2}} \xrightarrow{\text{(n-2)}} \\ T(n) &= 0, \text{ if } n = 1 \\ &= 1, \text{ if } n = 2 \\ &= T(k) + T(n-k) + 2, \text{ if } n > 2 \end{aligned}$$

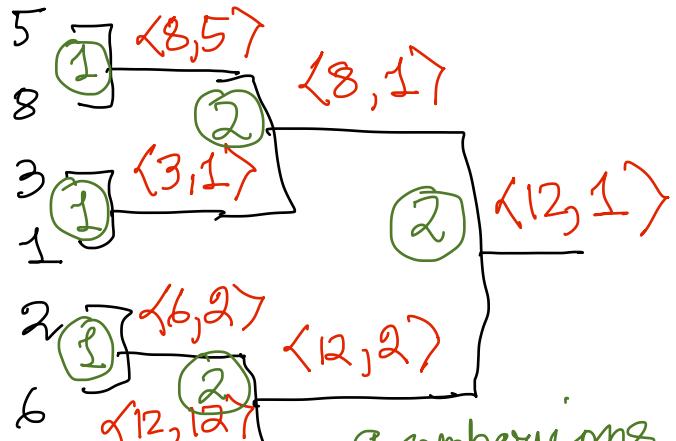
if we choose $k = 1$

$$\begin{aligned} T(1) + T(n-1) + 2 \\ = 2n - 3 \quad (\text{base case of } T(2) = 1) \end{aligned}$$

choose $k = 2$

$$\begin{aligned} &= T(2) + T(n-2) + 2 \\ &= T(2) + T(2) + T(n-4) + 2 + 2 \\ &= T(3n/2 - 2) \end{aligned}$$

Tournament Based Analysis & Design



9 comparisons

$$\left(\frac{3}{2}n - \frac{1}{2} \right)$$
$$\frac{3.7}{2} - \frac{1}{2}$$
$$= 51 - 2$$

optimal split ✓

$|L| = \text{even}$ then split into even, even parts



$|L| = \text{odd}$ then split into even, odd

6 elements

$\rightarrow 3,3$

$\rightarrow 4,2$

$$\left\lceil \frac{\frac{3}{2}n - 2}{2} \right\rceil$$

3rd Problem: Largest and Second Largest

Sequential Comparison

$\text{max1max2}(L)$

{ Let $L = \{x_1, x_2, \dots, x_n\}$

If $|L| = 1$ return (x_1, NULL)

$L' = L - \{x_1\}$

$(y_1, y_2) = \text{max1max2}(L')$

If $(x_1 \geq y_1)$ { $m_1 = x_1$
 $m_2 = y_1$ }

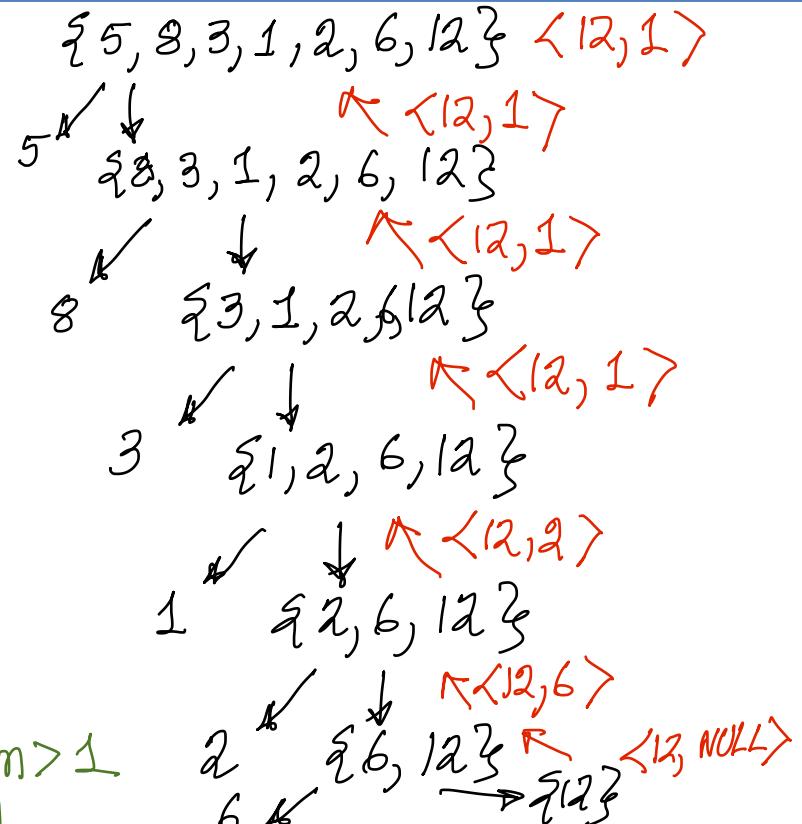
else { $m_1 = y_1$

if $(x_1 \geq y_2)$ $m_2 = x_1$
else $m_2 = y_2$

} return (m_1, m_2)

$$\begin{aligned} T(n) &= T(n-1) + 2, \\ &\quad n > 1 \\ &= 0, \quad n = 1 \end{aligned}$$

$$T(n) = 2(n-1) = 2n-2$$



Largest and 2nd Largest: Recursive Formulation

max1 max2 B(L)

{ Let L = {x₁, x₂, ..., x_n} }

if |L| = 1 return (<x₁, NULL>)

if |L| = 2 { if (x₁ ≥ x₂) { m₁ = x₁; }

 m₂ = x₂ }

else { m₁ = x₂, m₂ = x₁ }

~~return~~

return (<m₁, m₂>)

}

if |L| > 2

split L into 2 non-empty sets L₁, L₂

<y₁, y₂> = max1 max2 B(L₁)

<z₁, z₂> = max1 max2 B(L₂)

$$T(n) = \boxed{\frac{3n}{2} - 2}$$

if (y₁ ≥ z₁) { m₁ = y₁,

 if (z₁ ≥ y₂) m₂ = z₁

 else m₂ = y₂

}

else { m₁ = z₁

 if (y₁ ≥ z₂) { m₂ = y₁ }

 else { m₂ = z₂ }

}

return (<m₁, m₂>)

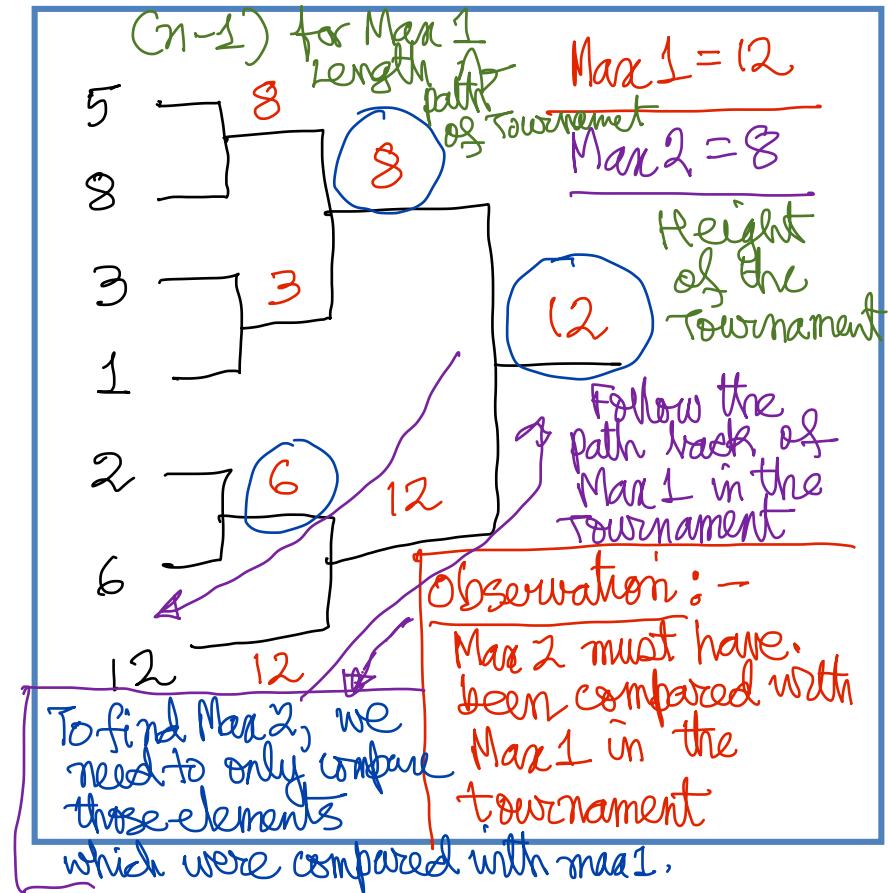
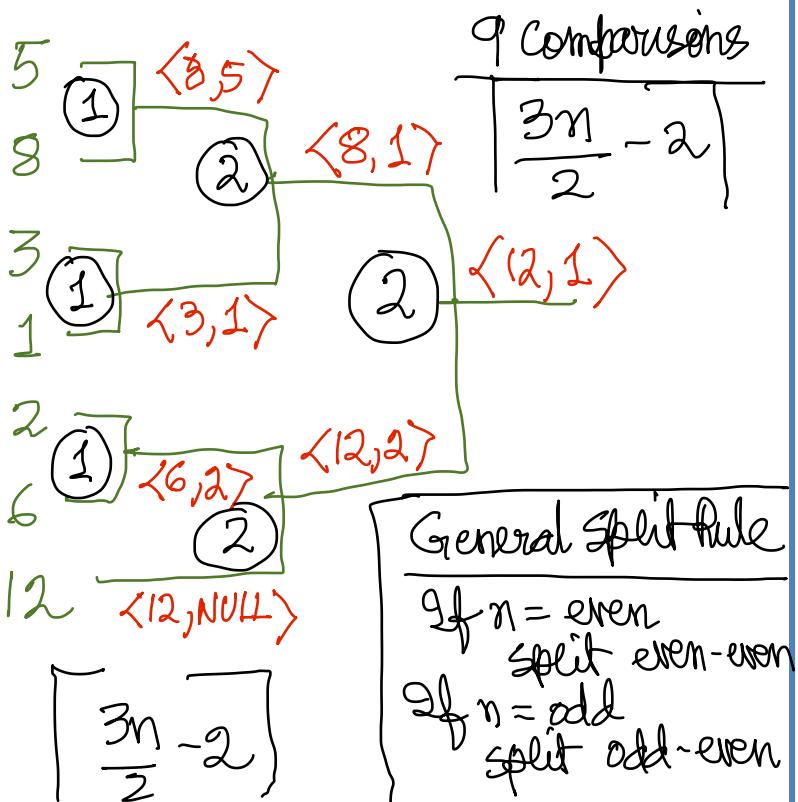
} $T(n) = T(k) + T(n-k) + 2, n > 2$

= 1, n = 2

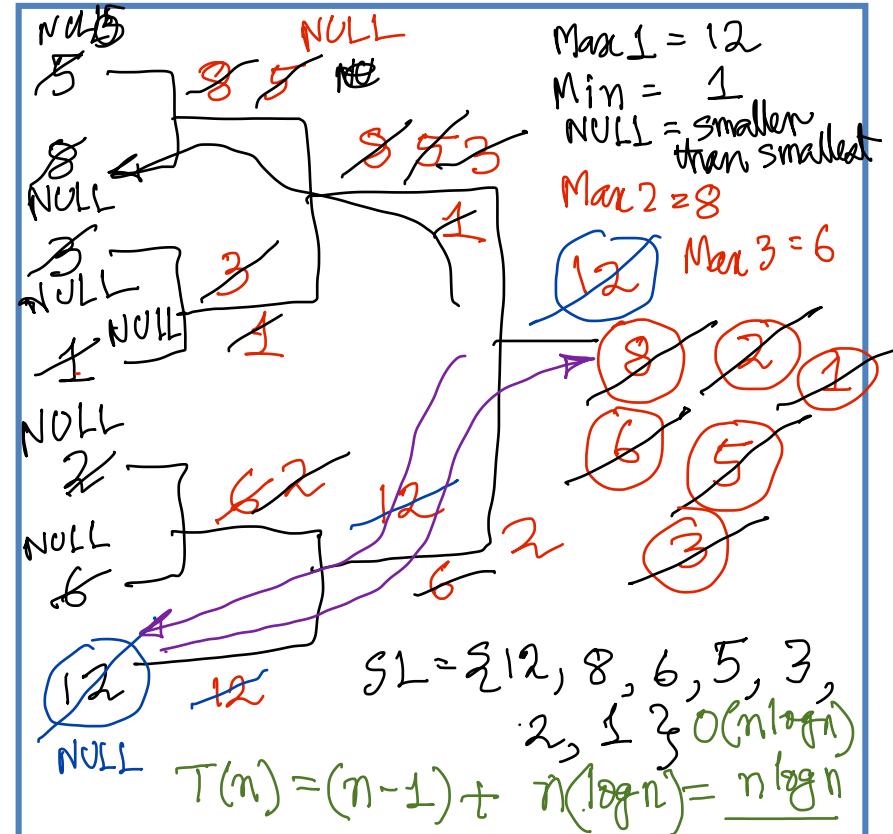
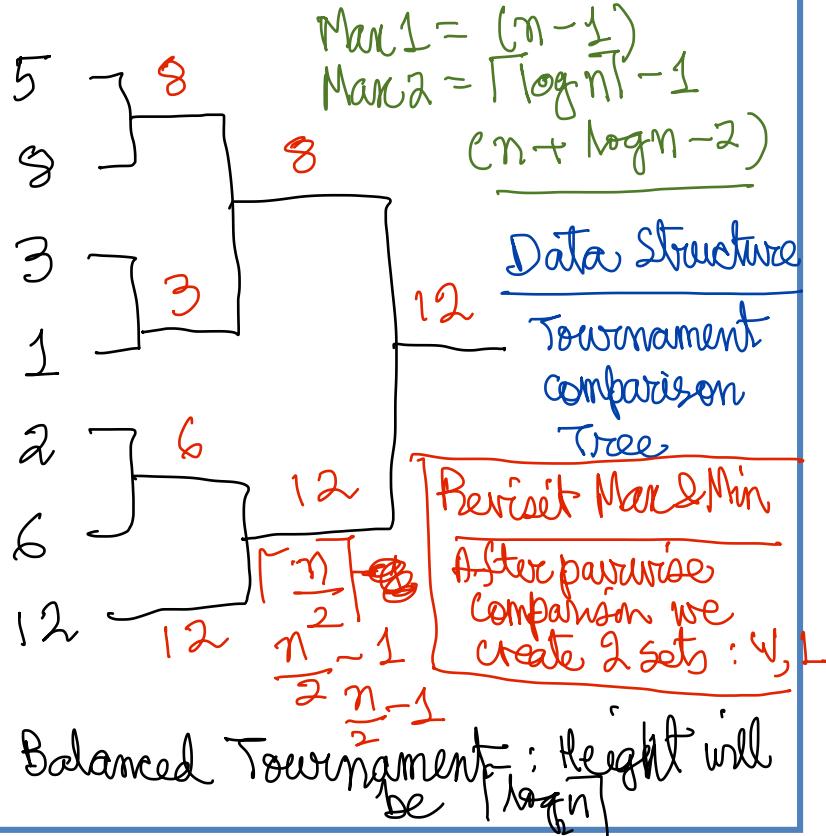
= 0, n = 1

Optimal split will occur for k=2

Largest and Next: Tournament



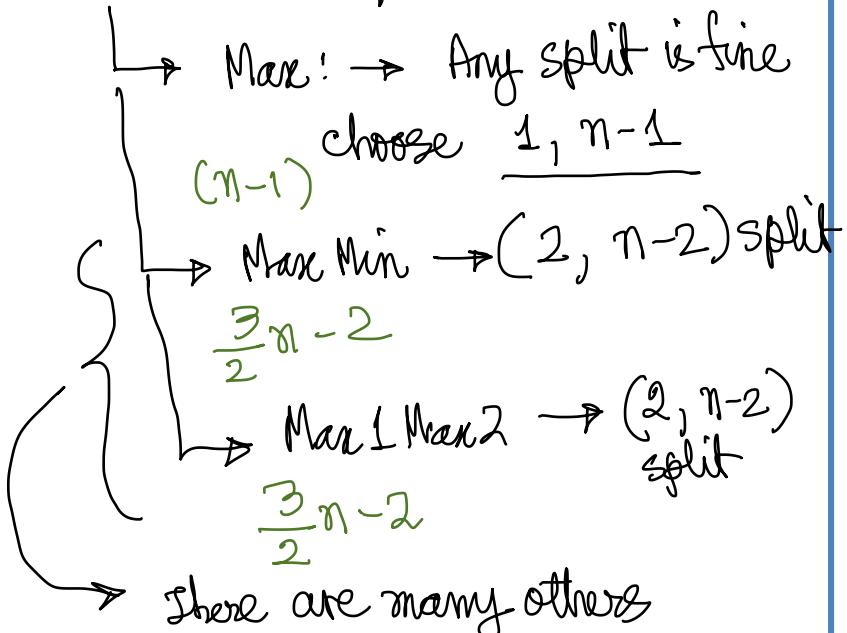
Tournaments & Sorting



Final Algorithms & Data Structuring

Max , MaxMin, Max1 Max2

Recursive Definition



Refinement of the recursion structure

Max : \rightarrow No further refinement

Max Min : \rightarrow Tournament structure for all the splits of optimal manner yields the same complexity

New Idea \downarrow Pairwise compare $\frac{3}{2}n-2$ lists of winners & losers and form 2 lists which

Max1 Max2 : Balanced Tournament

split which minimizes the height of the tournament allows a more efficient algorithm $((n-1) + \lceil \log n \rceil)$ SORTING

Algorithms + Data Structures = Programs

Max:

Use a temporary variable

"max"

Is there a better algorithm possible?

PROVE: Finding the largest by comparison using only a comparison operation cannot be done in less than ~~$(n-1)$~~ $(n-1)$ comparisons for n numbers

Max Min:

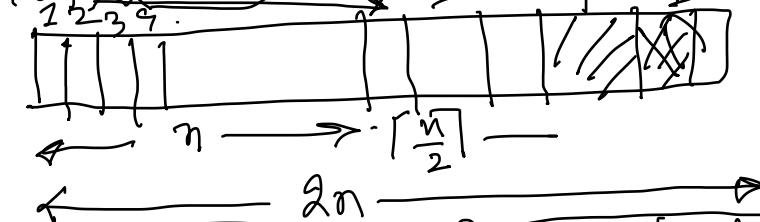
max, min

HEAP
Data Structure

Max1 Max2

Implement the Tournament Data

Structure



NPTEL Lectures: — Programming and Data Structures — by P. P. Chakrabarti

→ 3 lectures on Introduction to Data Structures 1, 2, 3

3 lectures on Recursive Definitions

Algorithm Design by Recursion Transformation

- ❑ Algorithms and Programs
- ❑ Pseudo-Code
- ❑ Algorithms + Data Structures = Programs
- ❑ Initial Solutions + Analysis + Solution Refinement + Data Structures = Final Algorithm
- ❑ Use of Recursive Definitions as Initial Solutions
- ❑ Recurrence Equations for Proofs and Analysis
- ❑ Solution Refinement through Recursion Transformation and Traversal
- ❑ Data Structures for saving past computation for future use

1. Initial Solution
 - a. Recursive Definition – A set of Solutions ✓
 - b. Inductive Proof of Correctness ✓
 - c. Analysis Using Recurrence Relations ✓
2. Exploration of Possibilities
 - a. Decomposition or Unfolding of the Recursion Tree ✓
 - b. Examination of Structures formed ✓
 - c. Re-composition Properties ✓
3. Choice of Solution & Complexity Analysis
 - a. Balancing the Split, Choosing Paths ✓
 - b. Identical Sub-problems ↙
4. Data Structures & Complexity Analysis
 - a. Remembering Past Computation for Future ↘
 - b. Space Complexity ↙
5. Final Algorithm & Complexity Analysis
 - a. Traversal of the Recursion Tree
 - b. Pruning
6. Implementation
 - a. Available Memory, Time, Quality of Solution, etc

Piṅgala's Numbers (3rd Century BC)

- The Chandahśāstra presents the first known description of a binary numeral system in connection with the systematic enumeration of meters with fixed patterns of short and long syllables. The discussion of the combinatorics of meter corresponds to the binomial theorem. Halāyudha's commentary includes a presentation of the Pascal's triangle (called *meruprastāra*). Piṅgala's work also contains the Fibonacci numbers, called *mātrāmeru*. (later Bharata Muni (100 BC), Virahanka (700 AD), Hemachandra (1150 AD) – all before Fibonacci 1200 AD)
$$f(n) = f(n-1) + f(n-2)$$
- Use of zero is sometimes ascribed to Piṅgala due to his discussion of binary numbers, usually represented using 0 and 1 in modern discussion, but Piṅgala used light (*laghu*) and heavy (*guru*) rather than 0 and 1 to describe syllables. As Piṅgala's system ranks binary patterns starting at one (four short syllables—binary "0000"—is the first pattern), the nth pattern corresponds to the binary representation of n-1 (with increasing positional values). Piṅgala is thus credited with using binary numbers in the form of short and long syllables (the latter equal in length to two short syllables), a notation similar to Morse code.
- Piṅgala used the Saṁskṛta word śūnya explicitly to refer to zero.

Mātrāmeru or Fibonacci Numbers

$$f(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f(n-1) + f(n-2), & \text{if } n>1 \end{cases}$$

$$f(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

0, 1, 1, 2, 3, 5, 8, ...

feb(n)

{ if ($n \leq 0$) return (0)
if ($n=1$) return (1)

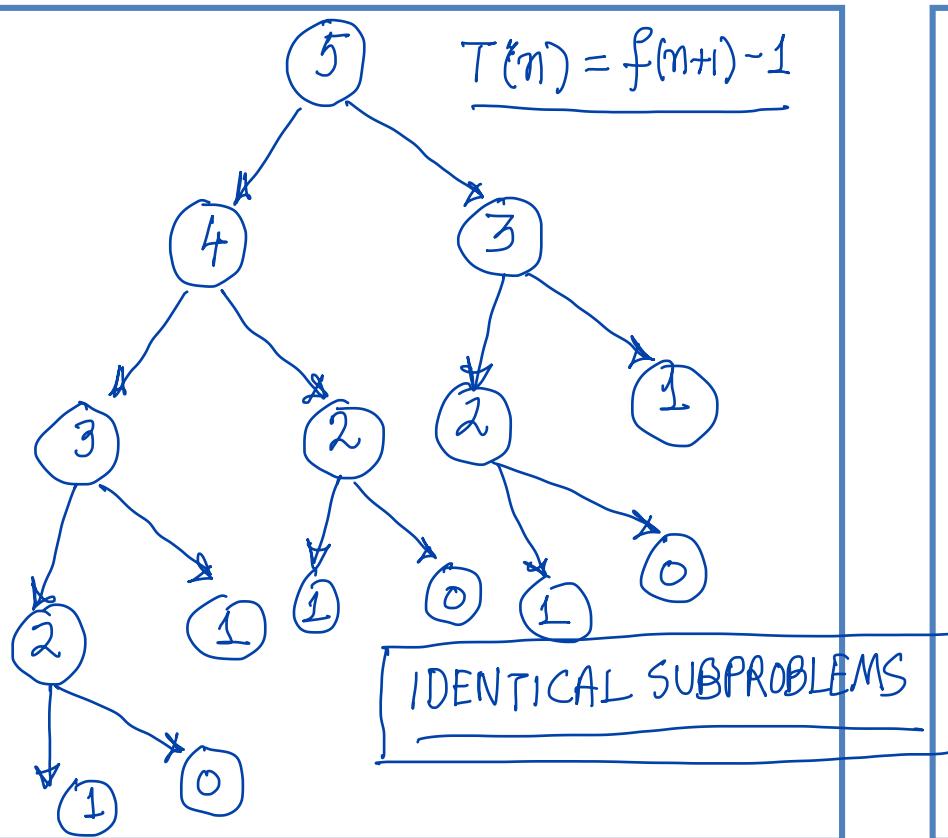
$m = feb(n-1) + feb(n-2)$
return (m) ↑

$T(n) = 0 \text{ if } n \leq 1$

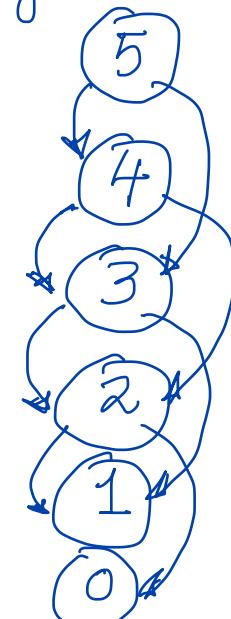
= $T(n-1) + T(n-2) + 1$

= $\boxed{feb(n+1) - 1}$

Analyzing the Recursion Structure



we would like to compute the value of $f(n)$ only once for every n and reuse the same



Data Storage

To store the required past computations

MEMOIZATION

$$T(n) = O(n)$$

Memoization

$FIB[]$, $FIB[0]=0$ $FIB[1]=1$

Top-down algorithm

$Done[]$, $Done[0]=1$, $Done[1]=1$
All others are 0.

$fib2(n)$

{ if ($Done[n]=1$) return ($FIB[n]$);

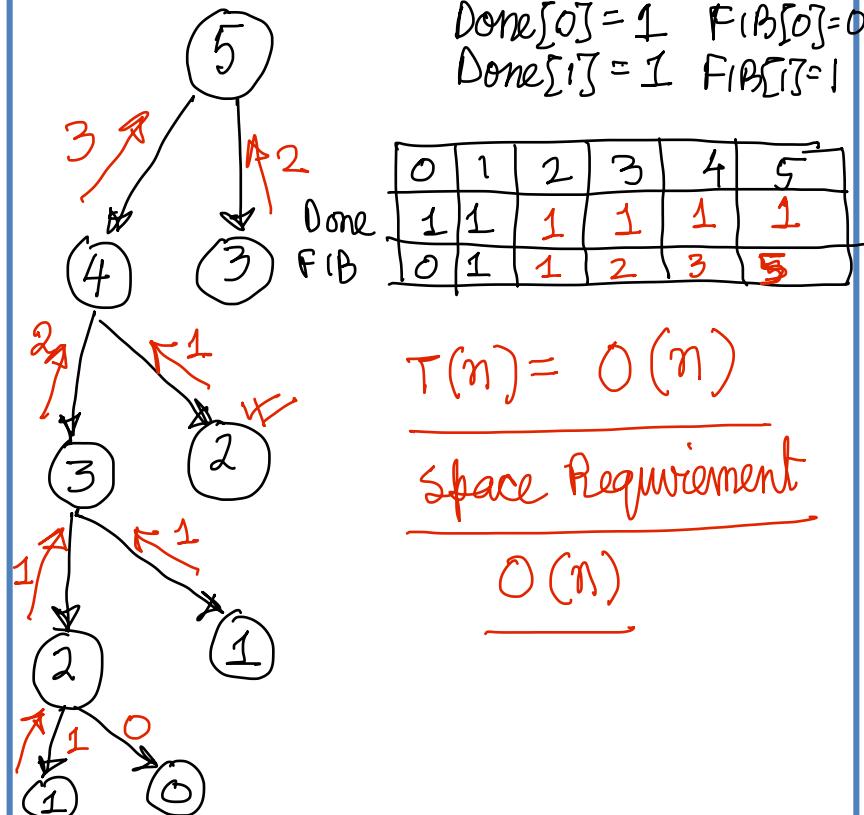
$$m = fib2(n-1) + fib2(n-2)$$

$Done[n] = 1$

$FIB[n] = m$

return (m)

}



Finalizing the Algorithm

$\text{fib3}(n)$

$$\begin{aligned} \text{FIB}[0] &= 0 \\ \text{FIB}[1] &= 1 \end{aligned}$$

{ for $i = 2$ to n do

$$\text{FIB}[i] = \text{FIB}[i-1] + \text{FIB}[i-2]$$

}

0	1	2	3	4	5	
0	1	1	2	3	5	...

BOTTOM-UP EVALUATION

$\text{fib4}(n)$

$$\left\{ \begin{array}{l} x_1 = 1 // \text{FIB}[1] \\ x_2 = 0 // \text{FIB}[0] \end{array} \right.$$

for $i = 2$ to n do

$$\left\{ \begin{array}{l} m = x_1 + x_2 \\ x_2 = x_1 \end{array} \right.$$

$$x_1 = m$$

{

return (m)

$\text{fib5}(n, -)$

{

Using TAIL RECURSION

Variations

$$1. \ f(n) = f(n-1) + f(n-157)$$

$$2. \ f(n) = f(n-1) + f\left(n - \frac{n}{2}\right)$$

$$3. \ f(n) = f(n+1) + f(n+3)$$

if n is even

$$f\left(\frac{n-1}{2}\right) \text{ if } n \text{ is odd}$$

$$4. \ f(n) = f(g(n)) + f(h(n))$$

↳ possibility of cyclic
dependencies

Evaluation of Fibonacci-like Recurrences

$$\begin{aligned}f(n) &= r(n) \text{ if } c(n) \text{ is true} \\&= f(g(n)) + f(h(n)) \\&\quad \text{if } c(n) \text{ is false}\end{aligned}$$

where $c(n)$ is the base condition. Also $c(n)$, $r(n)$, $g(n)$, $h(n)$ may be assumed to be deterministic non-recursive functions (some well-defined algorithm)

Write an algorithm to evaluate $f(n)$

1. Detect and flag cyclic dependencies
2. Avoid re-solving identical sub-problems
3. optimal memory usage

Evaluation Algorithm

Data Structure for
Dynamically storing
 $F[n]$ and $\text{Done}[n]$

eval-f(n)

{ if ($c(n) = \text{true}$) return ($r(n)$)

$x = g(n)$

$y = h(n)$

$z = f(x) + f(y)$

return (z)

$F[n]$ to store
value

}

$\text{Done}[i] \sim$

- 0 if it has
not yet been
called
- 1 if it has been
called.
- 2 if value is
computed

eval-f(n)

{ if ($c(n) = \text{true}$)

{ $\text{Done}[n] = 2$; $F[n] = r(n)$;
return ($F[n]$); }

if ($\text{Done}[n] = 1$) { print ("CYCLE")
exit }

if ($\text{Done}[n] = 2$) { return ($F[n]$); }

$\text{Done}[n] = 1$

$x = g(n)$; $y = h(n)$;

$z = \text{eval-f}(x) + \text{eval-f}(y)$.

$F[n] = z$

$\text{Done}[n] = 2$

return ($F[n]$)

We could also
memoize
 $g(n)$ & $h(n)$

Memoization Data Structure

Data Structures

define operations :-

find(n)
insert(n)

} ✓

Balanced

BST

create, get-val, assign-val

Coin Selection Problem

Given a set C of n coins having denomination values $\{c_1, c_2, \dots, c_n\}$ and a desired final value of V , find the minimum number of coins to be chosen from C to get an exact value of V from the sum of denominations of the chosen subset.

example: $C = \{8, 6, 5, 2, 1\}$, $V = 11$

$$S_1 = \{8, 3, 1\}, S_2 = \{6, 5\}$$

\uparrow minimum

coins (S, T, x, z, n)

S : set of coins selected till now

T : remaining set of coins from which we can select

x : value of set S

z : remaining value desired to be chosen from T

n : The number of coins selected

coins (NULL, C , 0, V , 0)

→ Base condition

→ Recursive condition

First Recursive Definition

$\langle P, d \rangle = \text{coins}(S, T, x, z, n)$
 { Let $S = \{s_1, s_2, \dots, s_n\}$
 $T = \{t_1, t_2, \dots, t_m\}$ }

BASE CONDITIONS

if ($z = 0$) return ($\langle S, n \rangle$)
if ($z < 0$) return ($\langle \text{NULL}, \infty \rangle$)
if ($T = \text{NULL}$) return ($\langle \text{NULL}, \infty \rangle$)

$P_{\min} = \text{NULL}$

$\min = \infty$

RECURSIVE CONDITION

for ($i = 1$ to m) do

{ $W = S + \{t_i\}$ }

$U = T - \{t_i\}$

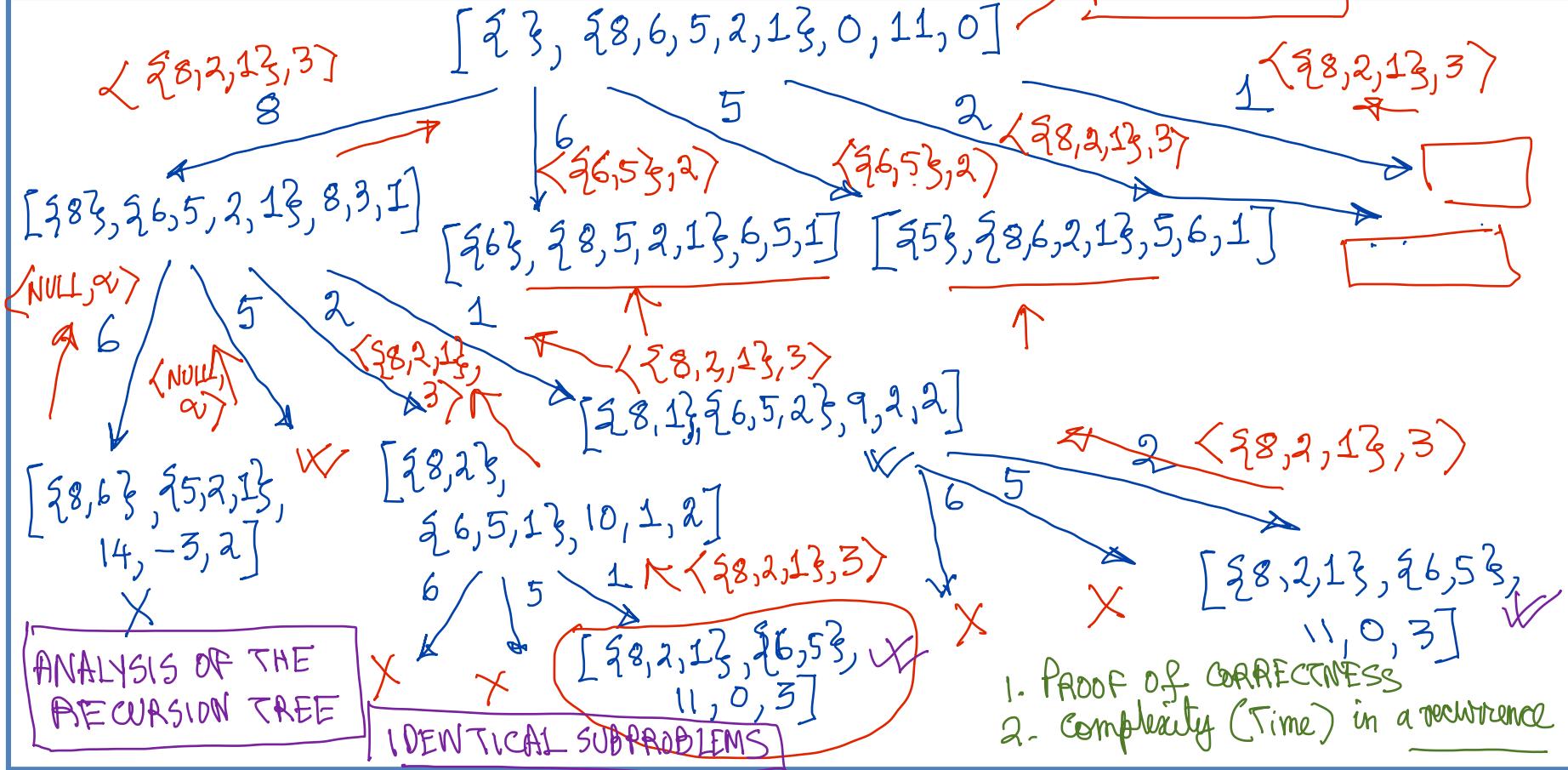
$\langle P', d' \rangle = \text{coins}(W, U, x + t_i, z - t_i, n + 1)$

if ($d' < \min$)

{ $\min = d'$
 $P_{\min} = P'$ }

}
return ($\langle P_{\min}, \min \rangle$)

Example



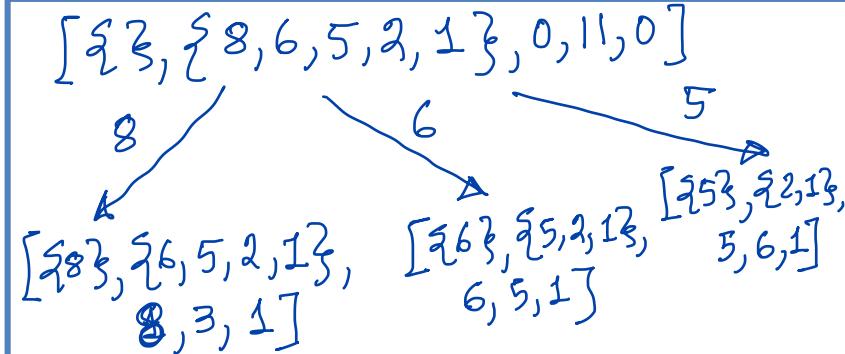
Improved Recursive Definition

Instead of

$$U = T - \{t_i\}$$

we do the following

$$U = T - \{t_1, t_2, \dots, \underline{t_i}\}$$



Identical subproblems that were generated earlier will not be generated now.

1. PROOF OF CORRECTNESS
2. TIME COMPLEXITY BASED ON Recurrence Relation
3. ANALYSIS OF RECURSION STRUCTURE

Alternative Recursive Definition

$$\langle P, d \rangle = \text{coins2}(S, T, x, z, n)$$

{ Let $S = \{s_1, s_2, \dots, s_n\}$

$T = \{t_1, t_2, \dots, t_m\}$

BASE CONDITIONS

If ($z=0$) return $\langle \langle S, n \rangle \rangle$

If ($z < 0$) return $\langle \text{NULL}, \alpha \rangle$

If ($T = \text{NULL}$) return $\langle \text{NULL}, \alpha \rangle$

$$\begin{aligned} p_{\min} &= \text{NULL} \\ \min &= \alpha \end{aligned}$$

Recursive Condition

(Inclusion - Exclusion Principle)

$$\langle P_1, d_1 \rangle = \text{coins2}(S+t_1, T-t_1, x+t_1, z-t_1, n+1)$$

$$\langle P_2, d_2 \rangle = \text{coins2}(S, T-t_1, x, z, n)$$

If ($d_1 \leq d_2$)

$$\{ p_{\min} = P_1 ; \min = d_1 \}$$

$$\text{else } \{ p_{\min} = P_2 ; \min = d_2 \}$$

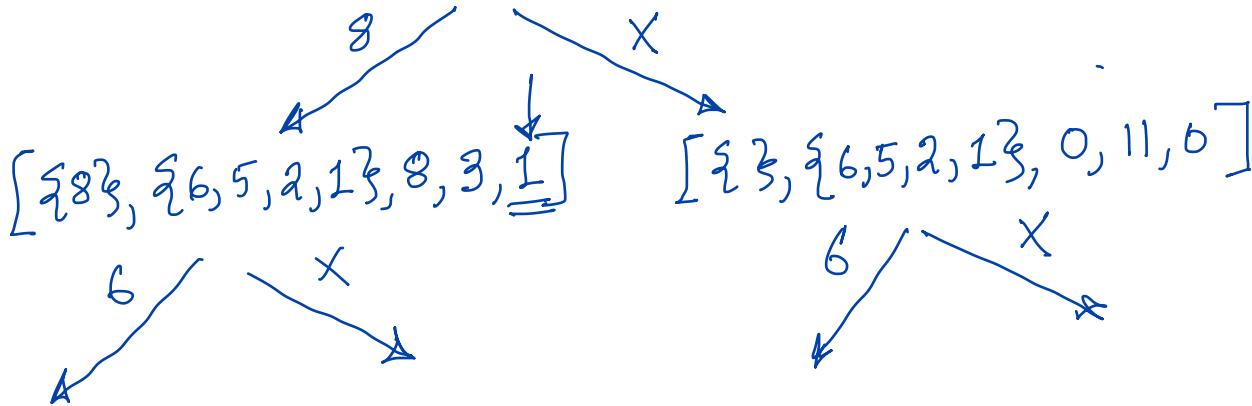
return $\langle p_{\min}, \min \rangle$

}

Example

$[23, 28, 6, 5, 2, 13, 0, 11, 0]$

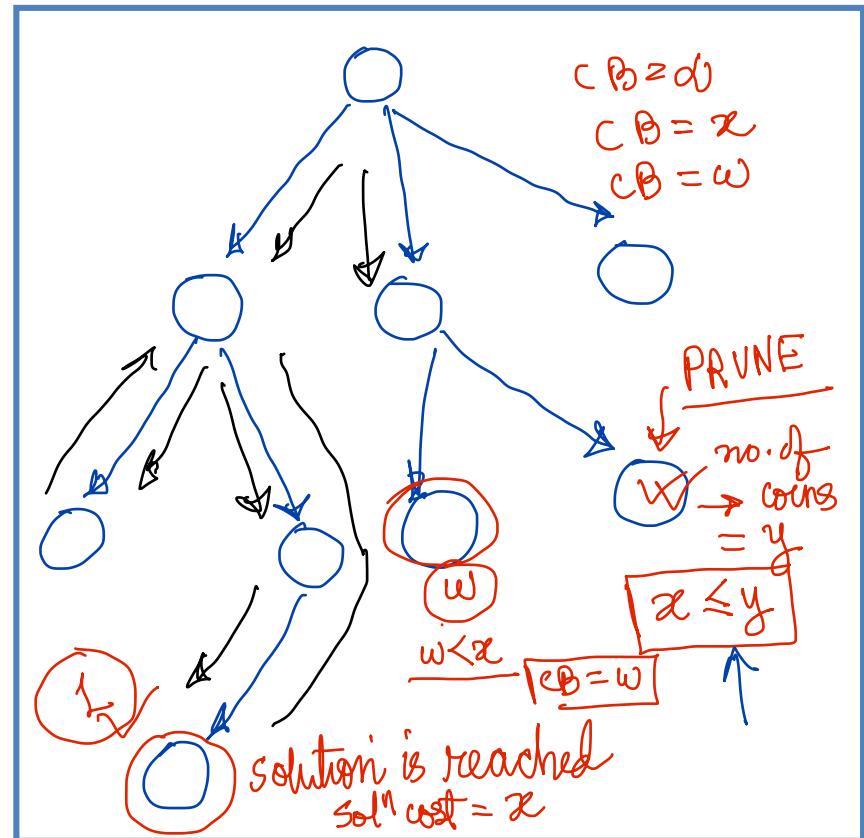
8:



6:

1. Inductive Proof
2. Time Recurrence
3. Identical Subproblems

Traversal and Potential Pruning



Maintain a global current best

$$CB = \alpha \text{ (initially)}$$

Recursion is evaluated in a depth-first manner

BASE CONDITIONS are revised
for pruning

$\text{g}_f(z=0) \left\{ \begin{array}{l} \text{if } (n < CB), CB = n \\ \text{[update the current best]} \\ \text{return}(s, n) \end{array} \right.$

g). $\oplus 2 < 0$) return (NULL , \emptyset).

if ($T = \text{NULL}$) return (LNULL, ∞)

~~If ($n \geq CB$) return ($\langle \text{NULL}, \infty \rangle$)~~

WORK IT OUT ON OUR EXAMPLE

Finalizing the Algorithm

2 options of Recursive Definitions

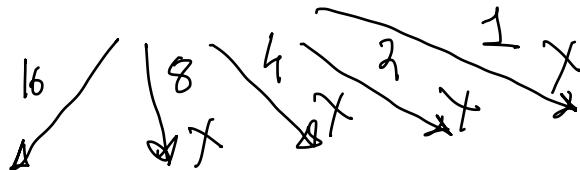
PRUNING (

→ IDENTICAL SUBPROBLEMS
ARISE

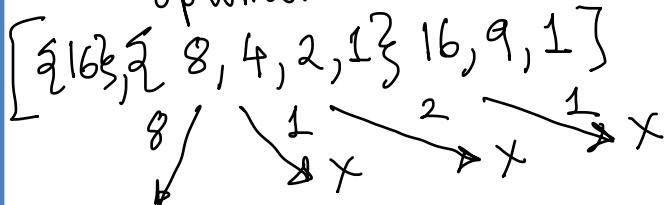
Special Case

$$C = \underbrace{\{16, 8, 4, 2, 1\}}_{\text{L}} \quad \{2^i\}$$

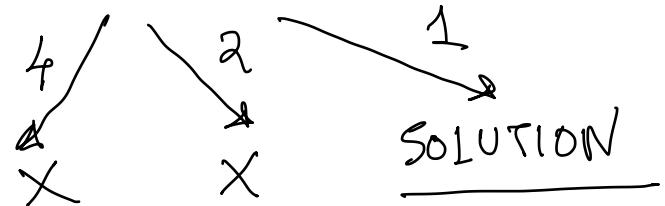
$$\rightarrow \text{gf } V = 25$$



choice for 16 will be part of
optimal solution



$$\left[\{16, 8\}, \{4, 2, 1\}, 24, 1, 2 \right]$$



We can make a SINGLE choice
from the various recursive
sub-problems.

$$\{100, 50, 25, 20, 10, 5, 3, 2, 1\}$$

Summary

1. Initial Solution
2. Analyse the recursion [D&C]
 - (a) Balancing the split [D&C]
 - (b) Identical Sub-problems (Memoization) [DP]
 - (c) choice (Greedy) from the subproblems upfront [G]
 - (d) Traversal or Evaluation of the recursion allows for pruning or pre-emption based on solutions already found. [BB]

3. Proof of correctness
4. Analysis of Complexity [Recurrence Eqns]
5. Data Structures [Asymptotic Analysis]

Problems we have examined

1. Max, Max-Min, Max1-Max2
2. FIB
3. Coins

Overview of Algorithm Design

1. Initial Solution

- a. Recursive Definition – A set of Solutions
- b. Inductive Proof of Correctness
- c. Analysis Using Recurrence Relations

2. Exploration of Possibilities

- a. Decomposition or Unfolding of the Recursion Tree
- b. Examination of Structures formed
- c. Re-composition Properties

3. Choice of Solution & Complexity Analysis

- a. Balancing the Split, Choosing Paths
- b. Identical Sub-problems

4. Data Structures & Complexity Analysis

- a. Remembering Past Computation for Future
- b. Space Complexity

5. Final Algorithm & Complexity Analysis

- a. Traversal of the Recursion Tree
- b. Pruning

6. Implementation

- a. Available Memory, Time, Quality of Solution, etc

1. Core Methods

- a. Divide and Conquer ✓
- b. Greedy Algorithms ✓
- c. Dynamic Programming ✓
- d. Branch-and-Bound ✓
- e. Analysis using Recurrences
- f. Advanced Data Structuring

2. Important Problems to be addressed

- a. Sorting and Searching
- b. Strings and Patterns
- c. Trees and Graphs
- d. Combinatorial Optimization

3. Complexity & Advanced Topics

- a. Time and Space Complexity
- b. Lower Bounds
- c. Polynomial Time, NP-Hard
- d. Parallelizability, Randomization

Thank you

Any Questions?