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

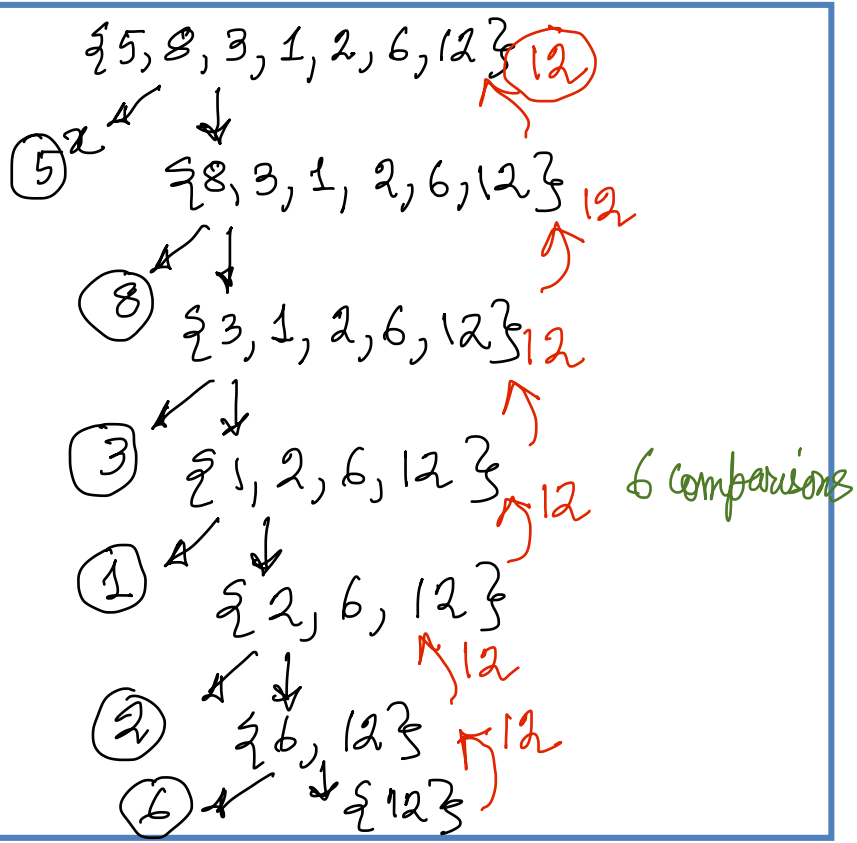
First Problem: Largest of a Set of Numbers

Sequential Comparison

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

```
max(L)
{ L = {x1, x2, ..., xn}
  if |L| = 1 return (x1)
    L' = L - {x1}
    y = max(L')
    if (x1 ≥ y) return (x1)
       else (y)
}
```

$$T(n) = T(n-1) + 1, \quad n > 1$$
$$= 0, \quad \text{if } n = 1$$
$$= (n-1)$$

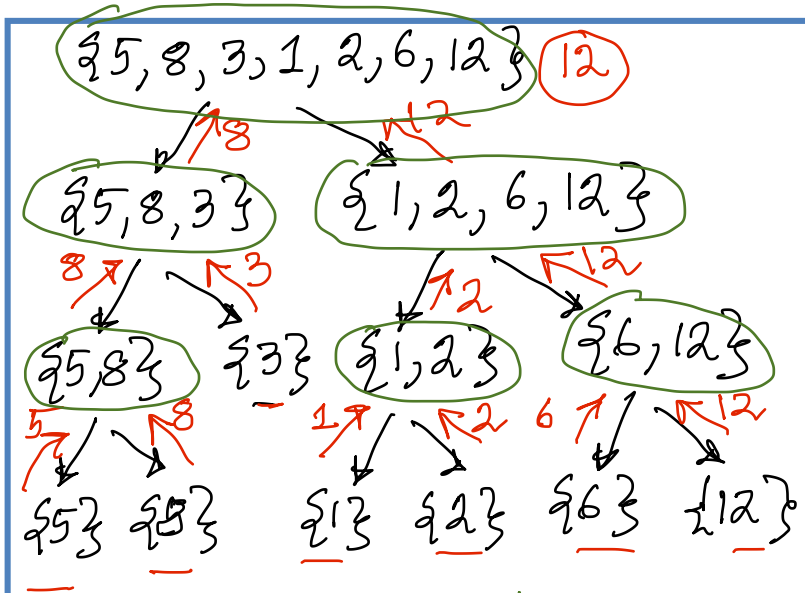


Finding Largest: Recursive Formulation

```
max2(L)
{  L = {x1, x2, ..., xn}
   if |L| = 1 return (x1)
   split L into 2 non-empty sets L1, L2
   y1 = max2(L1)
   y2 = max2(L2)
   if (y1 ≥ y2) return (y1)
      else return (y2)
}
```

Correctness proof by INDUCTION

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



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

$$T(n) = T(k) + T(n-k) + 1 \quad \text{for } n > 1$$

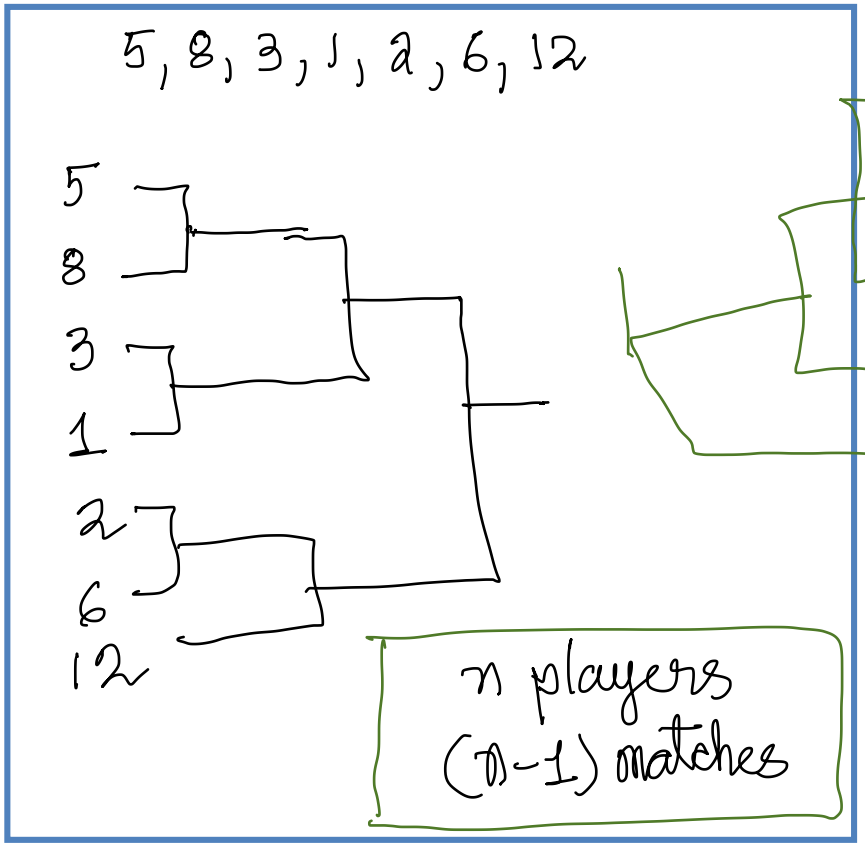
$$= 0 \quad , n = 1$$

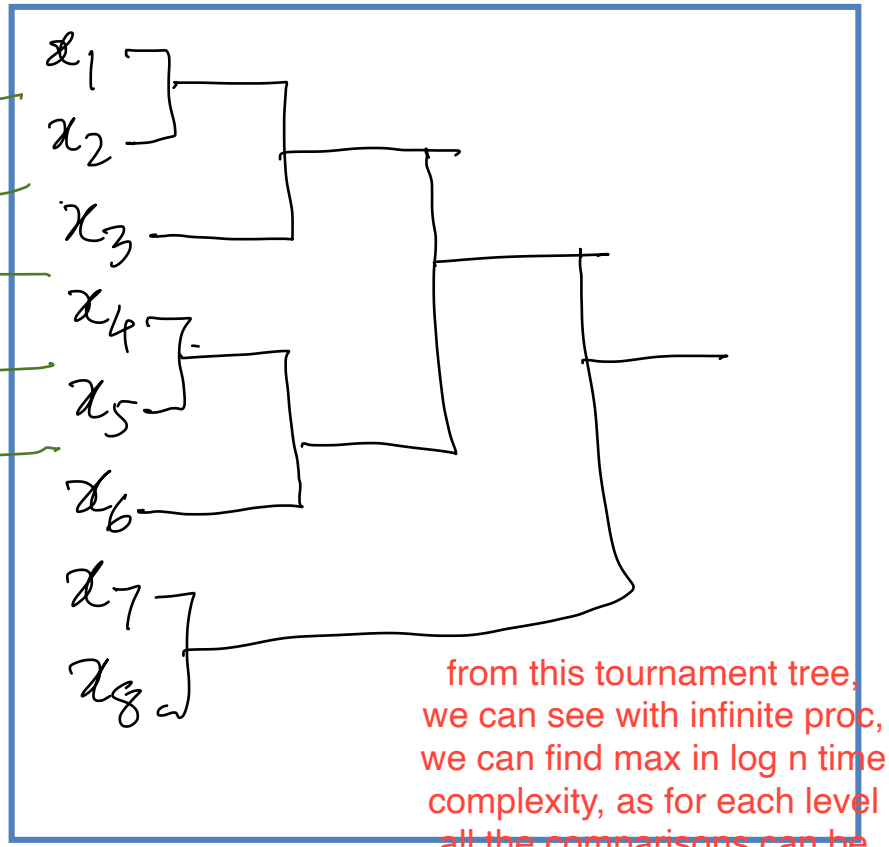
$$T(x) = x - 1$$

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

$$= n - 1$$

Comparison Tournament





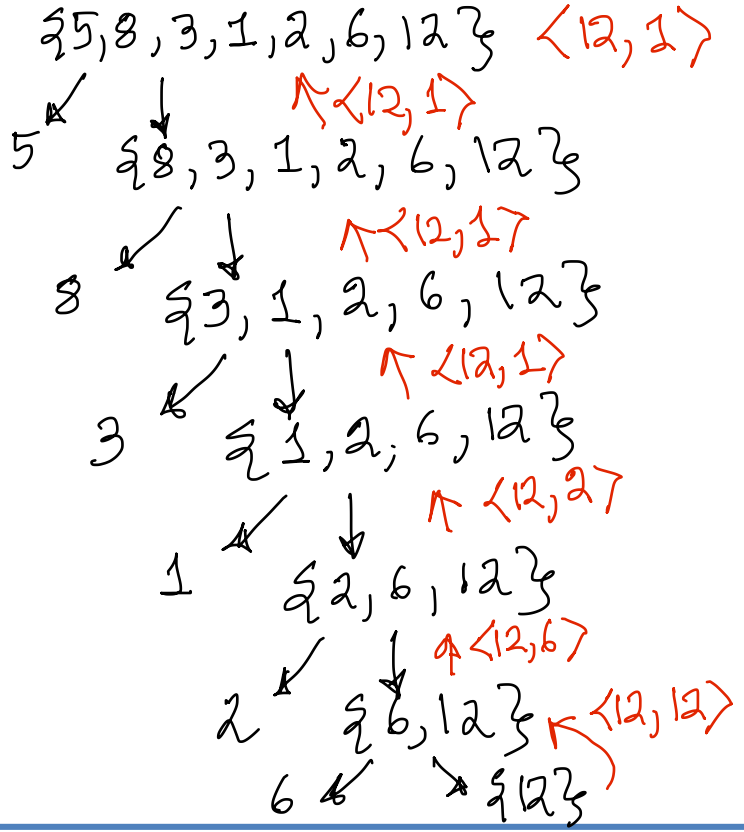
from this tournament tree, we can see with infinite proc, we can find max in log n time complexity, as for each level all the comparisons can be done in parallel.

2nd Problem: Largest and Smallest

Sequential Comparison

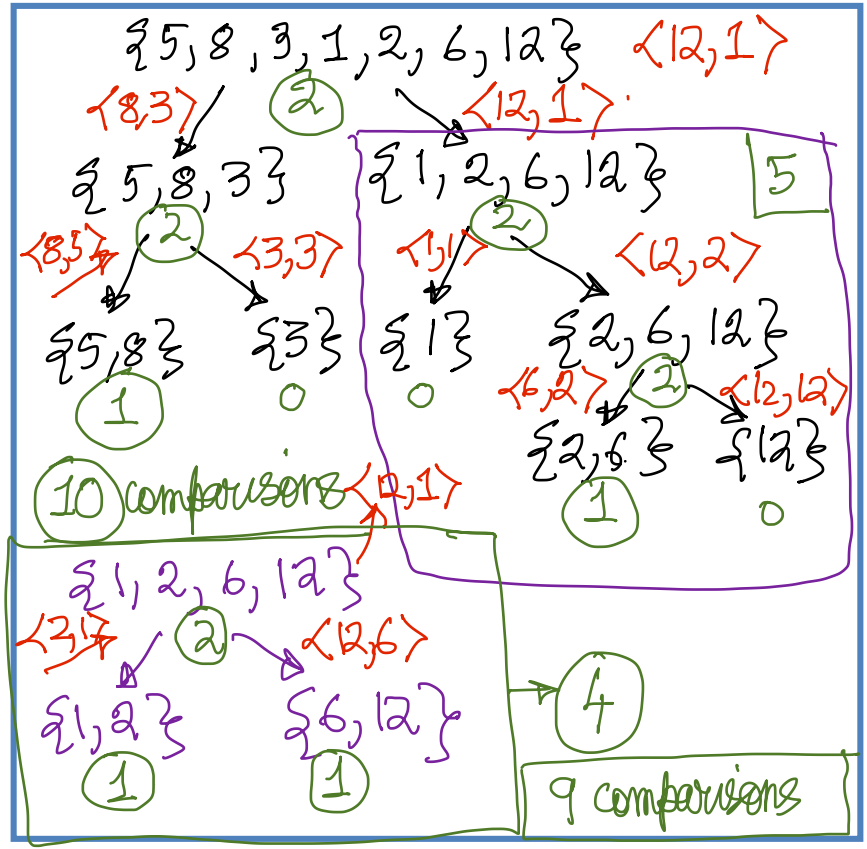
```
maxmin (L)
{ Let L = {x1, x2, ..., xn}
  if |L| = 1  return (<x1, x1>)
    L' = L - {x1}
    <y1, y2> = maxmin (L')
    if (x1 ≥ y1) then m1 = x1
                 else m1 = y1
    if (x1 ≤ y2) then m2 = x1
                 else m2 = y2
  return (<m1, m2>)
```

$$T(n) = 0, \text{ if } n = 1$$
$$= T(n-1) + 2, \; n > 1$$
$$= 2(n-1)$$

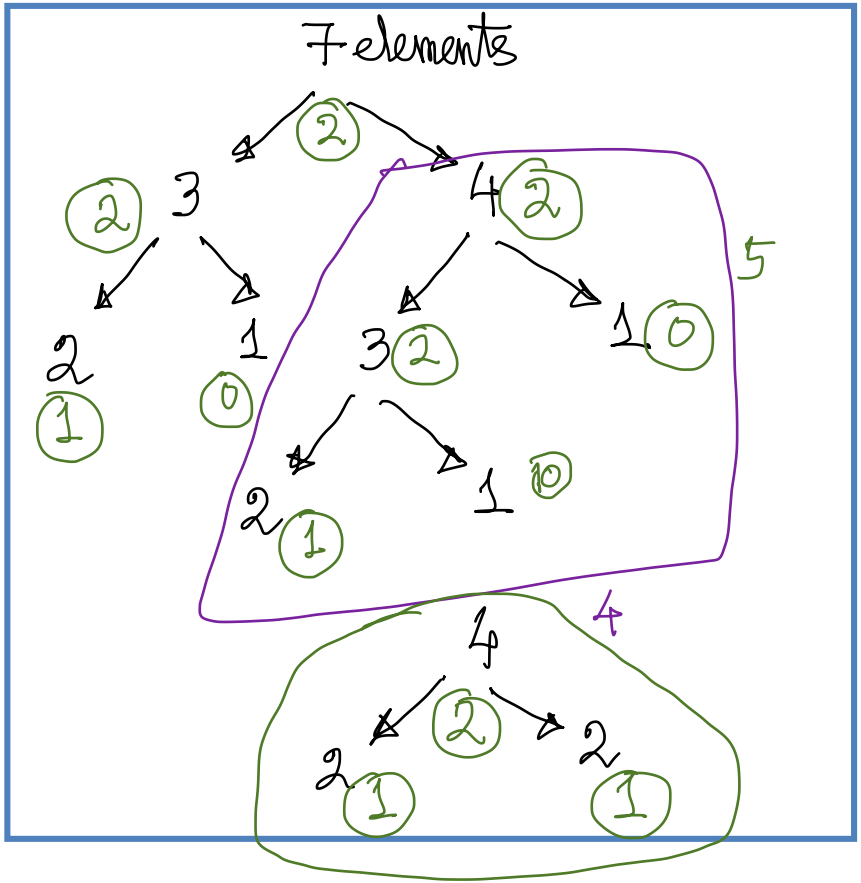


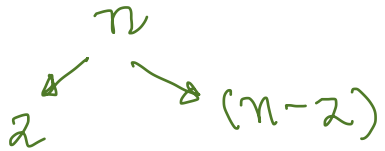
Largest and Smallest: Recursive Definition

```
maxmin2(L)
{ Let L = {x1, x2, ..., xn}
  if |L| = 1, return (<x1, x1>)
  if |L| = 2, if (x1 ≥ x2)
                return (<x1, x2>)
              else return (<x2, x1>)
  split L into 2 non-empty sets L1, L2
  <y1, y2> = maxmin2(L1)
  <z1, z2> = maxmin2(L2)
  if (y1 ≥ z1) m1 = y1 else m1 = z1
  if (y2 ≤ z2) m2 = y2 else m2 = z2
    return (<m1, m2>)
}
```



Largest & Smallest: Choice of Split





$T(n) = 0$, if $n = 1$

$= 1$, if $n = 2$

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

If we choose $k = 1$

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

$= 2n - 3$ (base case of $T(2) = 1$)

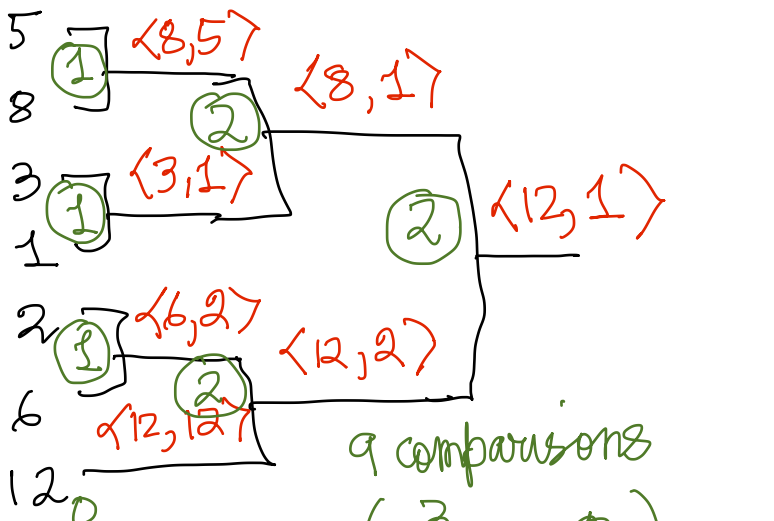
choose $k = 2$

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

$= T(2) + T(2) + T(n-4) + 2 + 2$

$= (3n/2 - 2)$

Tournament Based Analysis & Design



optimal split ✓

$|L|$ = even then split into even, even parts

4 → 3, 1

2, 2

$|L|$ = ~~odd~~ then split into even, odd

6 elements

→ 3, 3

→ 4, 2

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

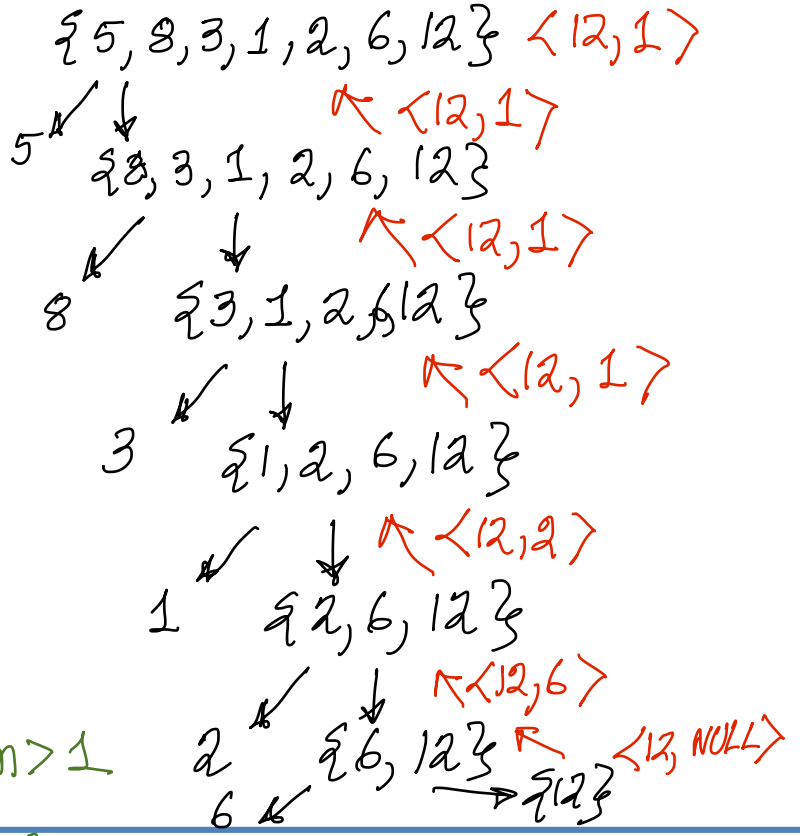
3rd Problem: Largest and Second Largest

Sequential Comparison

```
max1max2(L)
{ Let L = {x1, x2, ..., xn}
  If |L| = 1 return(<x1, NULL>)
  L' = L - {x1}
  <y1, y2> = max1max2(L')
  If (x1 ≥ y1) { m1 = x1
                 m2 = y1 }
  else { m1 = y1
         if (x1 ≥ y2) m2 = x1
         else m2 = y2 }
  return(<m1, m2>)
}
```

$T(n) = T(n-1) + 2, \quad n > 1$
$= 0, \quad n = 1$

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



Largest and 2nd Largest: Recursive Formulation

```
max1max2B(L)
{ Let L = {x1, x2, ..., xn}
  if |L| = 1 return (<x1, NULL>)
  if |L| = 2 { if (x1 ≥ x2) {m1 = x1;
                             m2 = x2 }
               else { m1 = x2, m2 = x1 }
               return (<m1, m2>)
             }
  if |L| > 2
    split L into 2 non-empty sets L1, L2
    <y1, y2> = max1max2B(L1)
    <z1, z2> = max1max2B(L2)
```

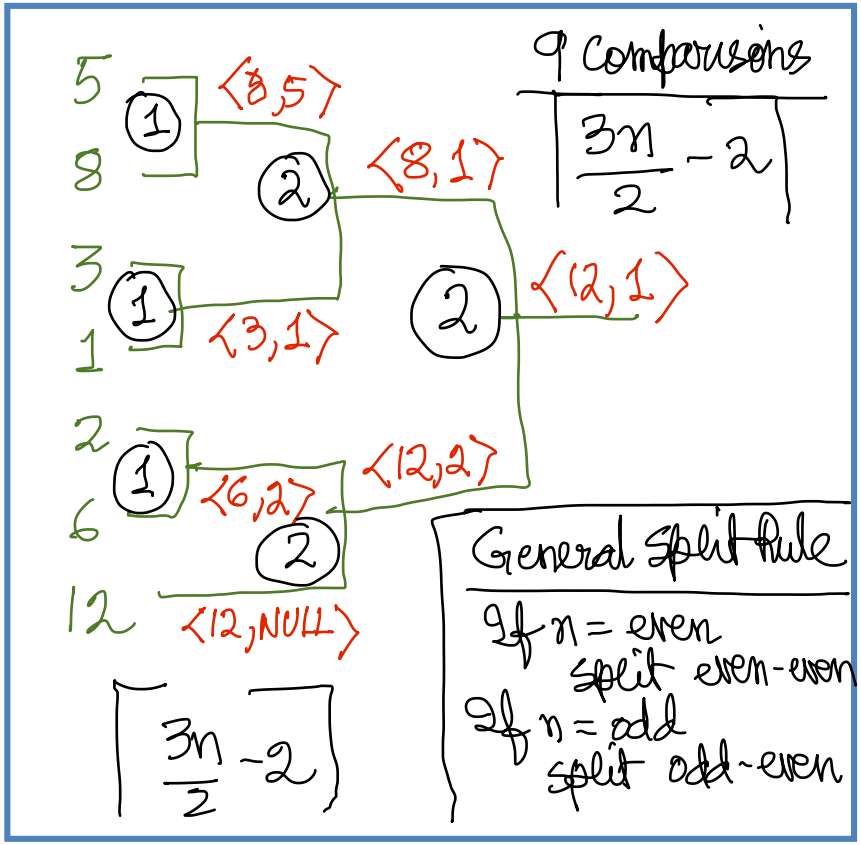
$$T(n) = \left\lceil \frac{3n}{2} - 2 \right\rceil$$

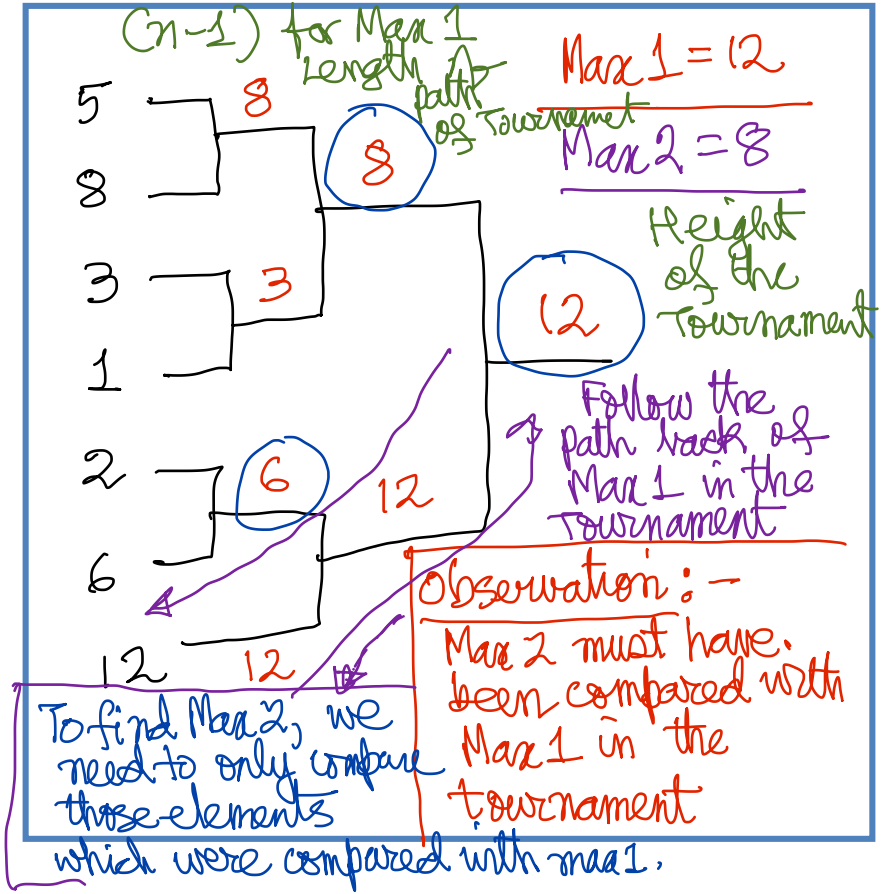
```
if (y1 ≥ z1) { m1 = y1
               if (z1 ≥ y2) m2 = z1
                 else m2 = y2
             }
  else { m1 = z1
         if (y1 ≥ z2) { m2 = y1 }
           else { m2 = z2 }
       }
  return (<m1, m2>)
}
```

$$T(n) = T(k) + T(n-k) + 2, \quad n \geq 2$$
$$= 1, \quad n = 2$$
$$= 0, \quad n = 1$$

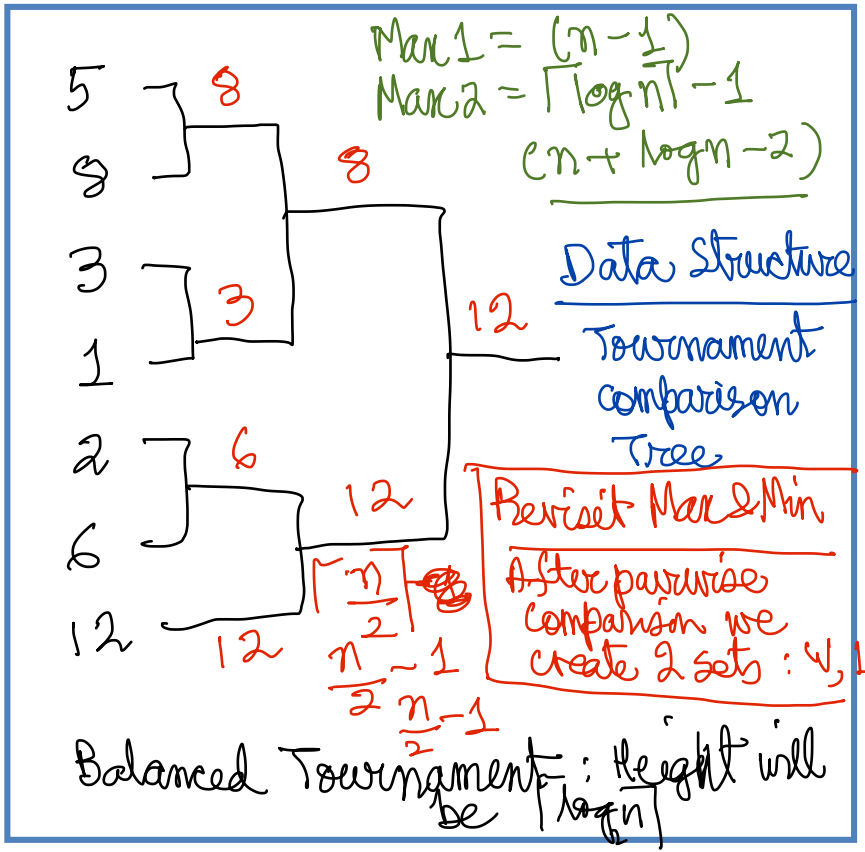
optimal split will occur for $k = 2$

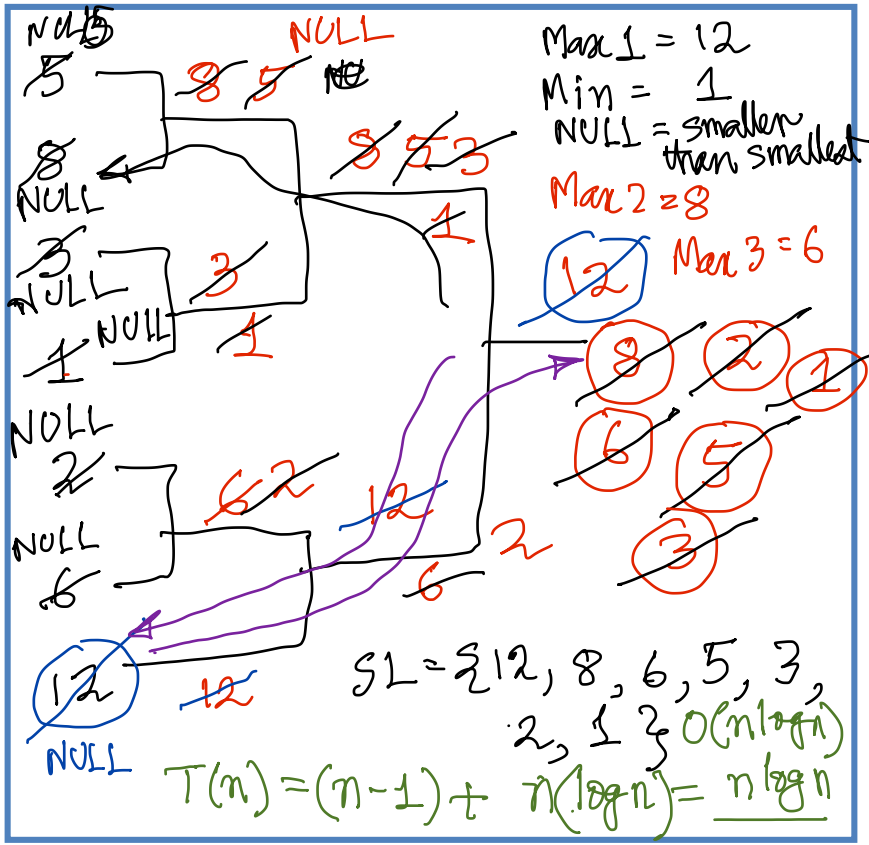
Largest and Next: Tournament





Tournaments & Sorting





Final Algorithms & Data Structuring

max , maxmin, max1 max2

Recursive Definition

- Max : → Any split is fine
  - choose 1, n-1
  - (n-1)
- Max Min → (2, n-2) split
  - $\frac{3}{2}n - 2$
- Max1 Max2 → (2, n-2) split
  - $\frac{3}{2}n - 2$
- There are many others

Refinement of the Recursion Structure

Max : → No further refinement

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

New Idea

↳ Pairwise compare and form 2 lists of Winners & Losers $\frac{3}{2}n - 2$

Max1 Max2 : Balanced Tournament split which minimizes the height of the Tournament allows a more efficient algorithm

((n-1) + ⌈log n⌉) SORTING

Algorithms + Data Structures = Progams

Max: Use a temporary variable "max"

Is there a better algorithm possible?

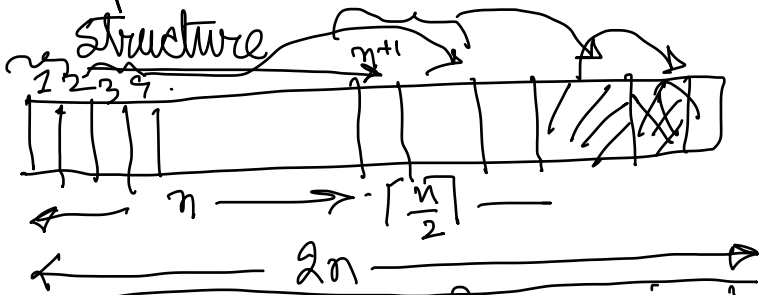
PROVE: Finding the largest by comparison using only a comparision operation cannot be done in less than $(n-1)$ comparisons for n numbers

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 Definition

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 Chandaḥśāstra 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

$f(n) = 0$ if $n = 0$ $(n \le 0)$

$= 1$ if $n = 1$

$= f(n-1) + f(n-2)$, if $n > 1$

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

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

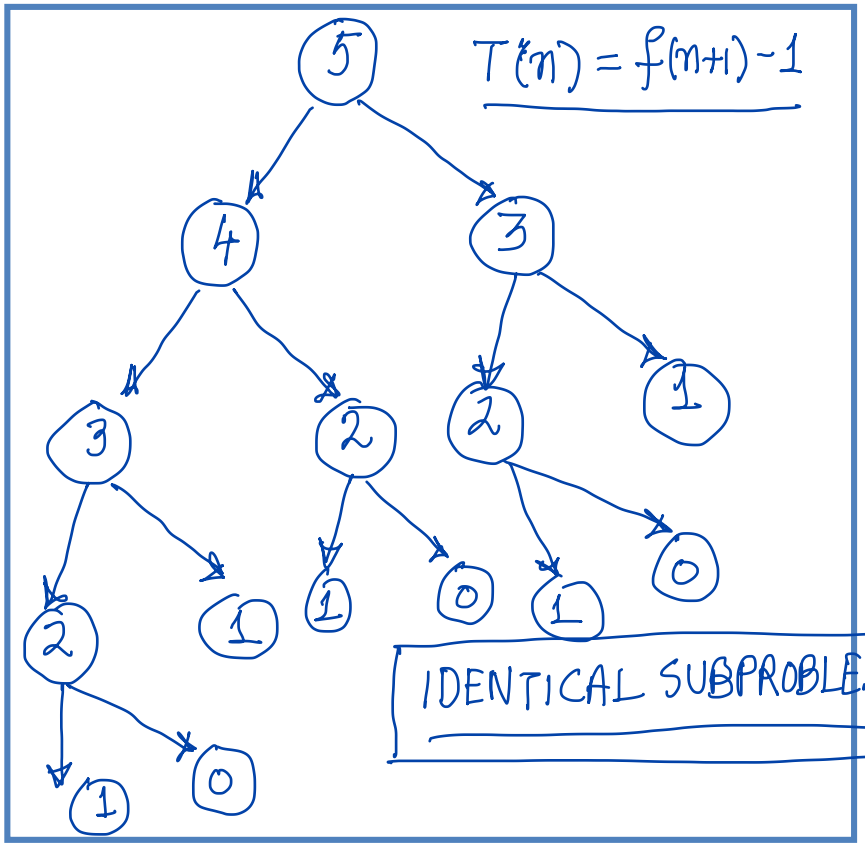
```
fib(n)
{  if (n <= 0) return (0)
   if (n == 1) return (1)
   m = fib(n-1) + fib(n-2)
   return (m)
}
```

$T(n) = 0$ if $n \le 1$

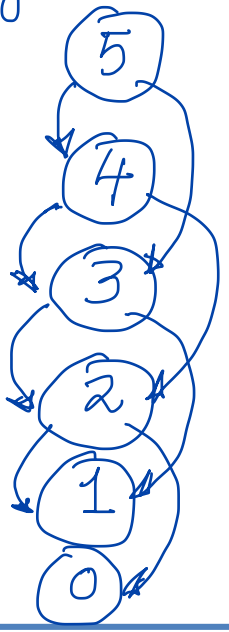
$= T(n-1) + T(n-2) + 1$

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

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

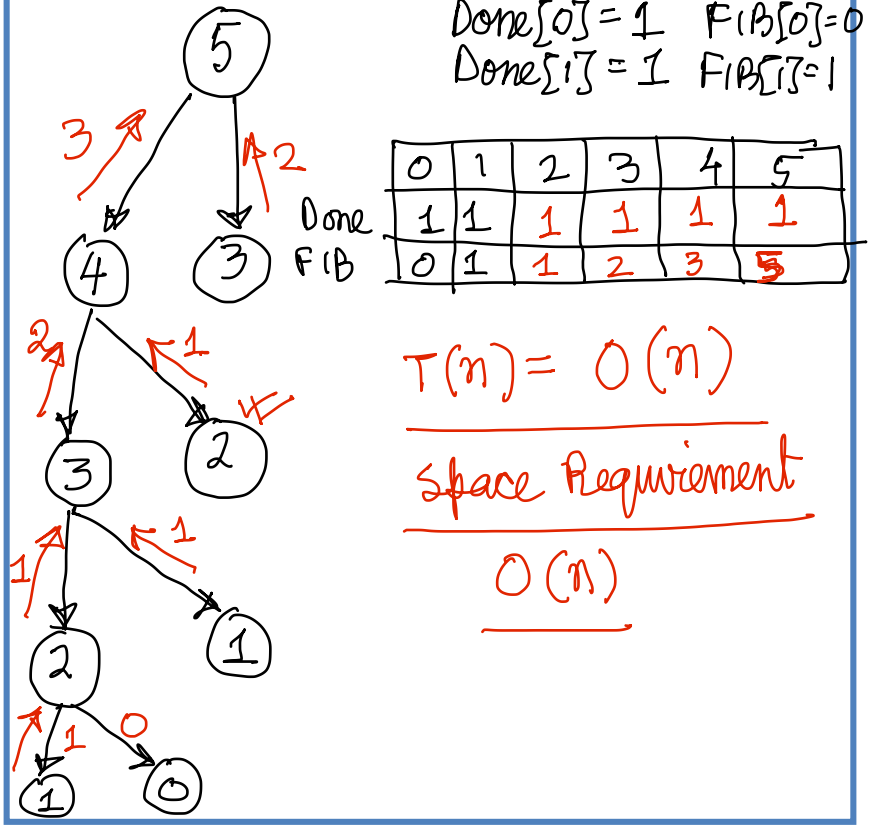
FIB[ ] , FIB[0] = 0 FIB[1] = 1

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

```
fib3(n)                    FIB[0]=0
                           FIB[1]=1
{ for i=2 to n do
    FIB[i] = FIB[i-1] +
             FIB[i-2]
}
```

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

BOTTOM-UP EVALUATION

```
fib4(n)
{ x1=1  // FIB[1]
  x2=0  // FIB[0]
  for i=2 to n do
  { m = x1+x2
    x2 = x1
    x1 = m
  }
  return (m)
}
```

fib5(n, --)

Using TAIL RECURSION

Variations

1. $f(n) = f(n-1) + f(n-157)$
2. $f(n) = f(n-1) + f(n - \frac{n}{2})$
3. $f(n) = f(n+1) + f(n+3)$ if n is even

   $f(\frac{n-1}{2})$ if n is odd
4. $f(n) = f(g(n)) + f(h(n))$

   ↳ possibility of cyclic dependencies

Evaluation of Fibonacci-like Recurrences

$$f(n) = r(n) \text{ if } c(n) \text{ is true}$$
$$= f(g(n)) + f(h(n)) \text{ if } c(n) \text{ is false}$$

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[] and Done[]

```
eval-f(n)
{ if (c(n) = true) return(r(n))
    x = g(n)
    y = h(n)
    z = f(x) + f(y)
    return (z)
}
```

F[] to store value

Done[i]
- 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) = true)
    { Done[n] = 2 ; F[n] = r(n) ;
      return (F[n]); }
  if (Done[n] = 1) { print ("CYCLE")
                     exit }
  if (Done[n] = 2) return (F[n]);
    Done[n] = 1
    x = g(n) ; y = h(n) ;
    z = eval-f(x) + eval-f(y).
    F[n] = z
    Done[n] = 2
    return (F[n])
}
```

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

Memoization Data Structure

Data Structure

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 $\{c_1, c_2, \ldots, c_n\}$ 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: $C = \{8, 6, 5, 2, 1\}$, $V = 11$

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

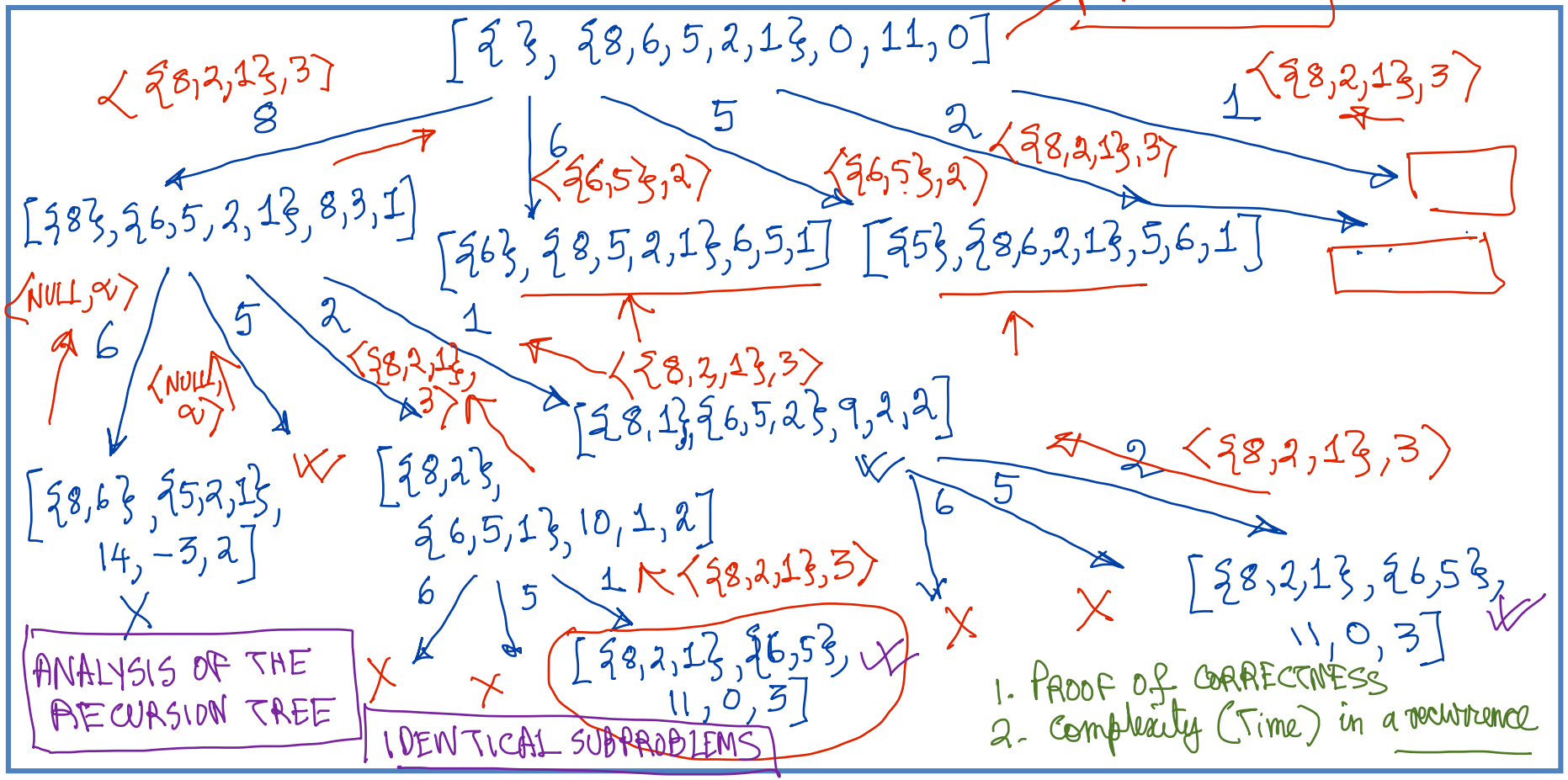
First Recursive Definition

```
⟨P, d⟩ = coins(S, T, x, z, n)
{  Let S = {s1, s2, ..., sn}
       T = {t1, t2, ..., tm}

BASE CONDITIONS
   if (z = 0) return (⟨S, n⟩)
   if (z < 0) return (⟨NULL, ∞⟩)
   if (T = NULL) return (⟨NULL, ∞⟩)
   Pmin = NULL
   min = ∞
```

```
RECURSIVE CONDITION
   for (i = 1 to m) do
   {  W = S + {ti}
      U = T − {ti}
      ⟨P', d'⟩ = coins(W, U, x + ti,
                       z − ti, n + 1)
      if (d' < min)
      {  min = d'
         Pmin = P'
      }
   }
   return (⟨Pmin, min⟩)
}
```

Example


[{}, {8,6,5,2,1}, 0, 11, 0]
<{6,5}, 3>
<{8,2,1}, 3>
8
6
5
2
1
[{8}, {6,5,2,1}, 8, 3, 1]
[{6}, {8,5,2,1}, 6, 5, 1]
[{5}, {8,6,2,1}, 5, 6, 1]
<{6,5}, 2>
<{8,2,1}, 3>
<NULL, ∞>
[{8,6}, {5,2,1}, 14, -3, 2]
[{8,2}, {6,5,1}, 10, 1, 2]
[{8,1}, {6,5,2}, 9, 2, 2]
[{8,2,1}, {6,5}, 11, 0, 3]
ANALYSIS OF THE RECURSION TREE
IDENTICAL SUBPROBLEMS
1. PROOF OF CORRECTNESS
2. Complexity (Time) in a recurrence

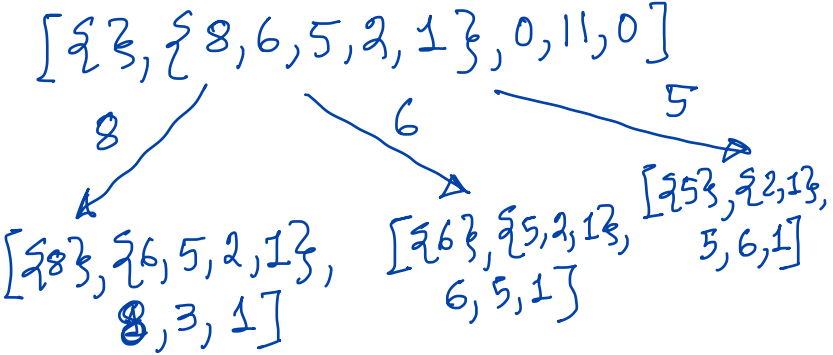
Improved Recursive Definition

Instead of

$U = T - \{t_i\}$

we do the following

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



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

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

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

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

BASE CONDITIONS

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

if $(z < 0)$ return $(\langle NULL, \infty \rangle)$

if $(T = NULL)$ return $(\langle NULL, \infty \rangle)$

$P_{min} = NULL$

$min = \infty$

Recursive condition

(Inclusion - Exclusion Principle)

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

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

if $(d_1 \leq d_2)$

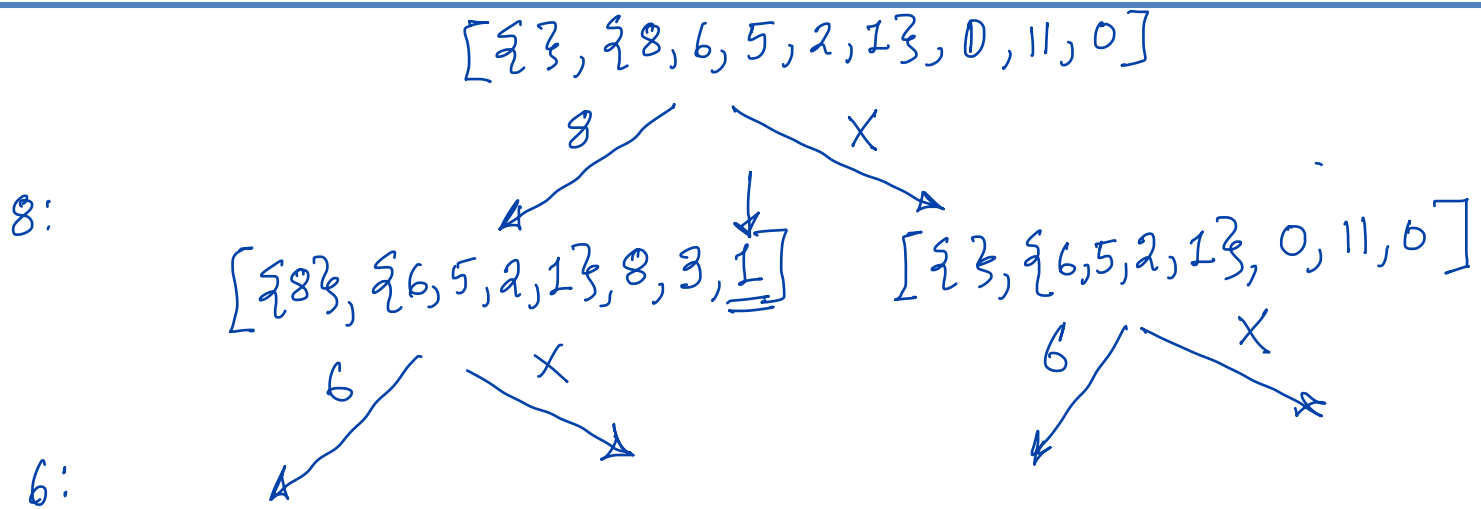
{ $P_{min} = P_1$; $min = d_1$ }

else { $P_{min} = P_2$; $min = d_2$ }

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

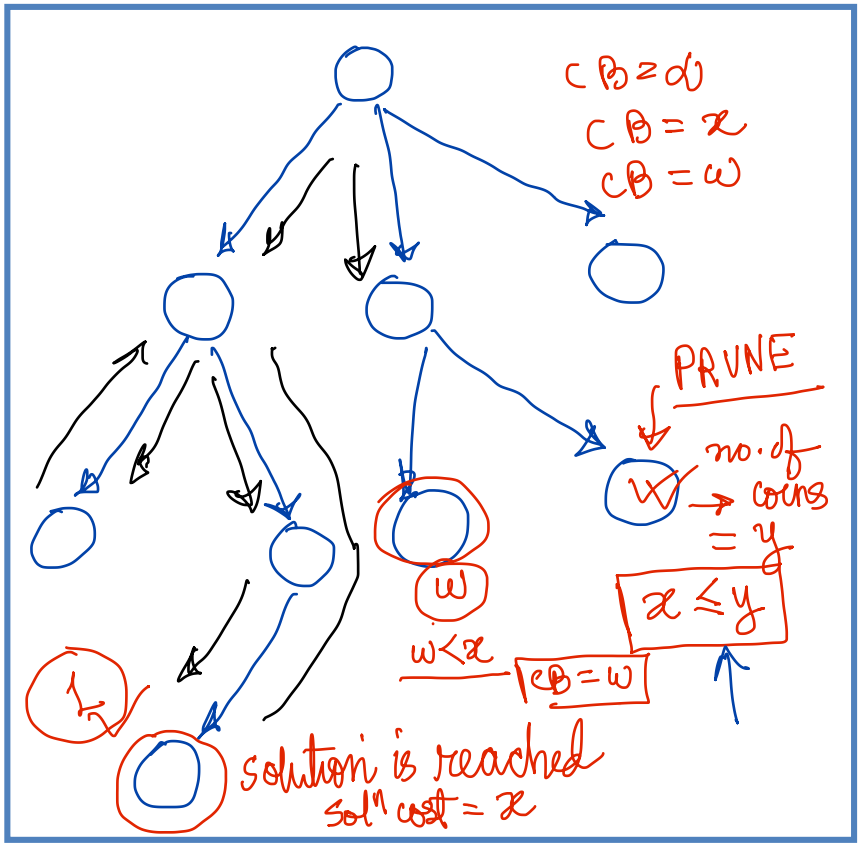
}

Example



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

Traversal and Potential Pruning



Maintain a global current best

CB = ∝ (initially)

Recursion is evaluated in a depth-first manner

BASE CONDITIONS are Revised for Pruning

```
if (Z=0) { if (n < CB), CB = n
           [update the current best]
           return(<S, n>)
}
if (Z<0) return (<NULL, ∞>)
if (T = NULL) return (<NULL, ∞>)
if (n ≥ CB) return(<NULL, ∞>)
```

PRUNING

WORK IT OUT ON OUR EXAMPLE

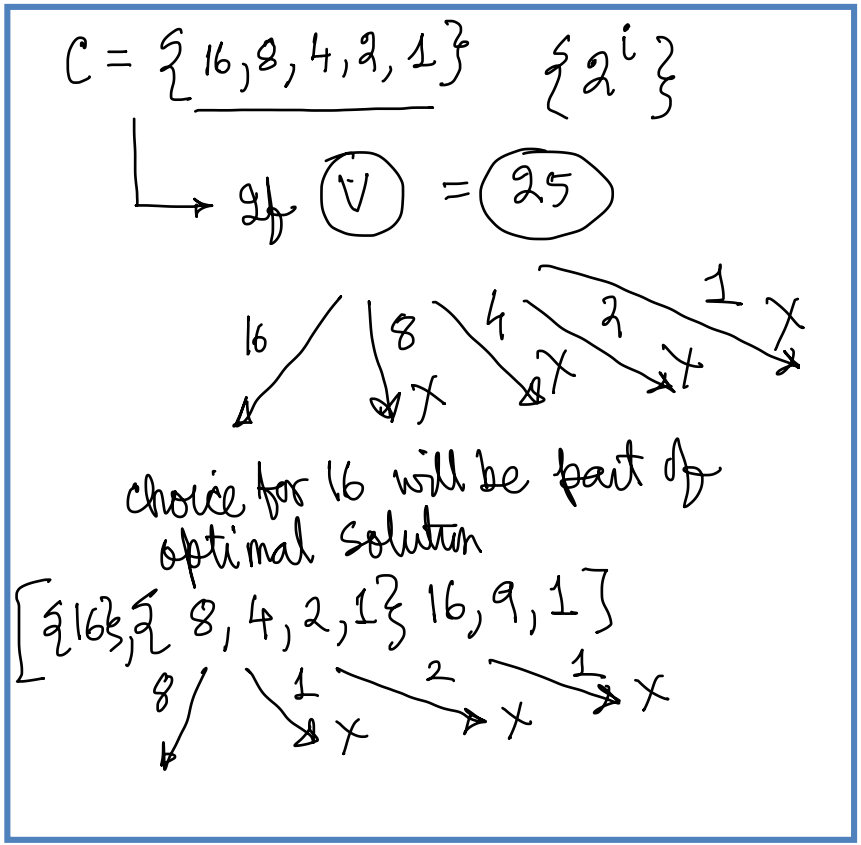
Finalizing the Algorithm

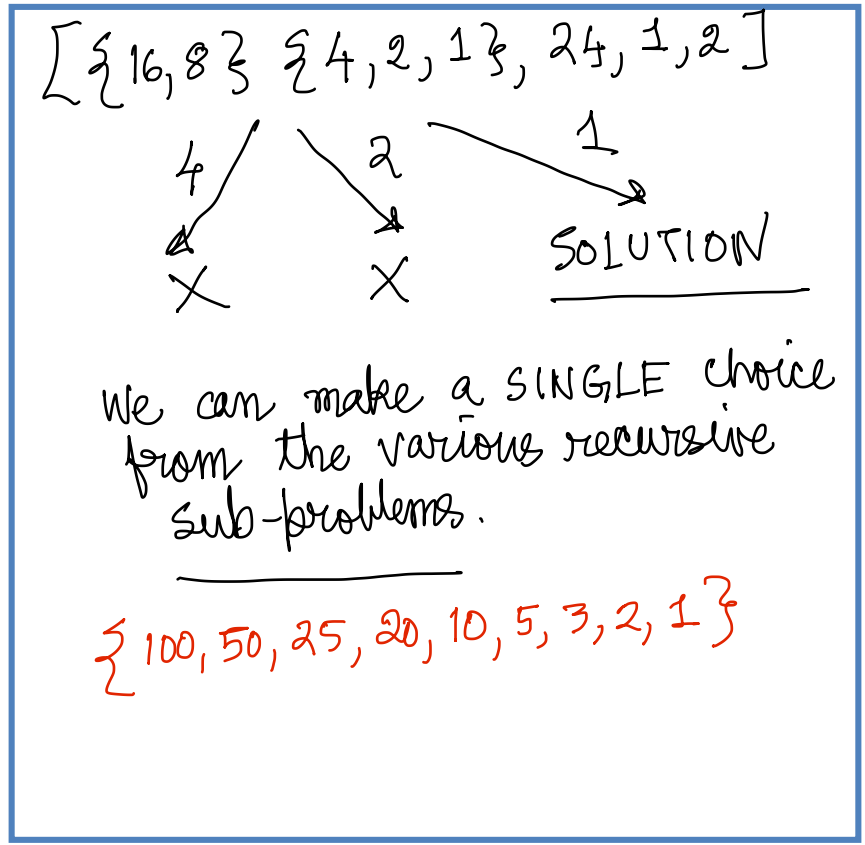
2 options of Recursive Definitions

PRUNING (

→ IDENTICAL SUBPROBLEMS ARISE

Special Case





Summary

1. Initial Solution
2. ~~Analyse the Recursion~~
   (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, Max 1-Max 2
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?