Introduction to Recursive Formulations for Algorithm Design: II

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

Sample Problems:

Sequential recursion

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

Generalized recursive formulation

3rd Problem: Largest and Second Largest

Sequential Comparison

max1max2(L)

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

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

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

  $\langle y_1, y_2 \rangle =$ 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 $m_2 = y_2$

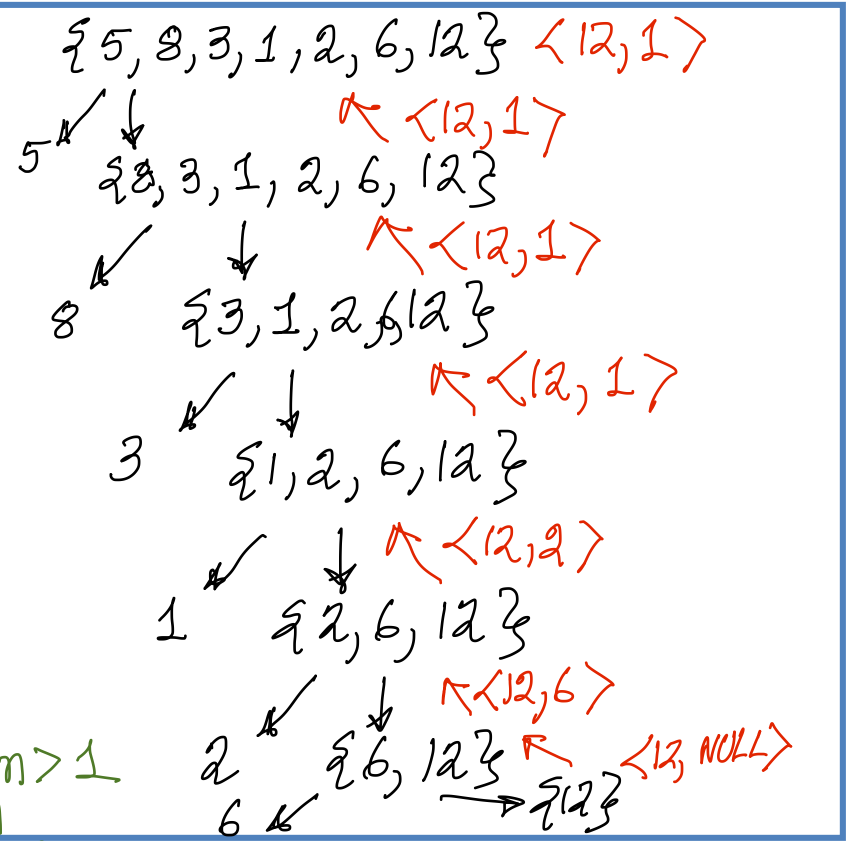
  }

  return($\langle m_1, m_2 \rangle$)

}

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

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



Largest and 2nd Largest: Recursive Formulation

max1max2B(L)
{ Let $L = \{x_1, x_2, \ldots, x_n\}$
if $|L| = 1$ return ($\langle x_1, \text{NULL} \rangle$)
if $|L| = 2$ { if ($x_1 \geq x_2$) { $m1 = x_1$; $m_2 = x_2$ }
else { $m_1 = x_2$, $m_2 = x_1$ }
return ($\langle m_1, m_2 \rangle$)
}
if $|L| > 2$
split L into 2 non-empty sets L_1, L_2
$\langle y_1, y_2 \rangle$ = max1max2B (L_1)
$\langle z_1, z_2 \rangle$ = max1max2B (L_2)

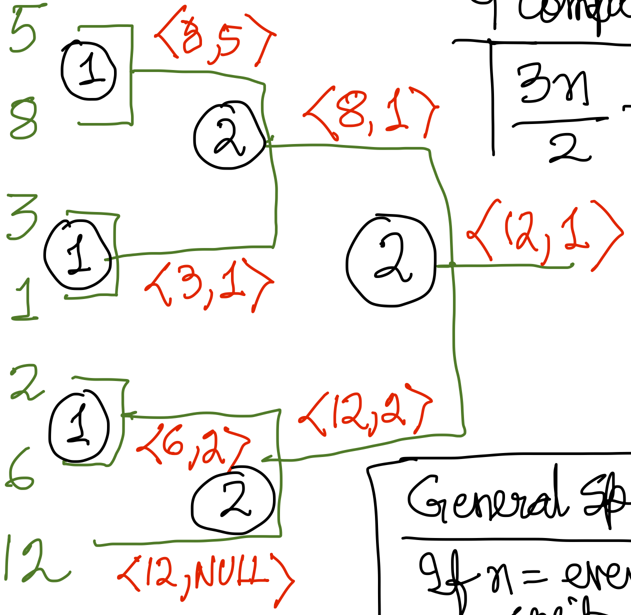
$$\boxed{T(n) = \left\lceil \frac{3n}{2} - 2 \right\rceil}$$

if ($y_1 \geq z_1$) { $m_1 = y_1$
if ($z_1 \geq y_2$) $m_2 = z_1$
else $m_2 = y_2$
}
else { $m_1 = z_1$
if ($y_1 \geq z_2$) { $m_2 = y_1$ }
else { $m_2 = z_2$ }
}
return ($\langle m_1, m_2 \rangle$)
}

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

optimal split will occur for $k = 2$

Largest and Next: Tournament



9 Comparisons

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

General Split Rule

If n = even
split even-even
If n = odd
split odd-even

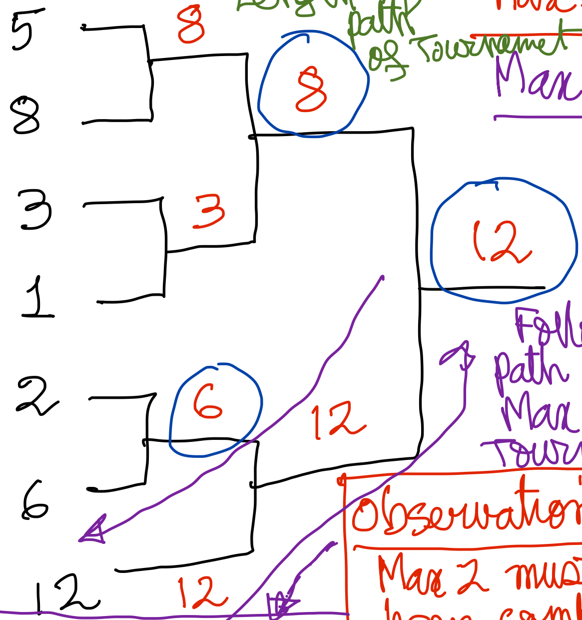
$\left\lceil \frac{3n}{2} - 2 \right\rceil$

(n-1) for Max 1
Length of path of Tournament

Max 1 = 12

Max 2 = 8

Height of the Tournament

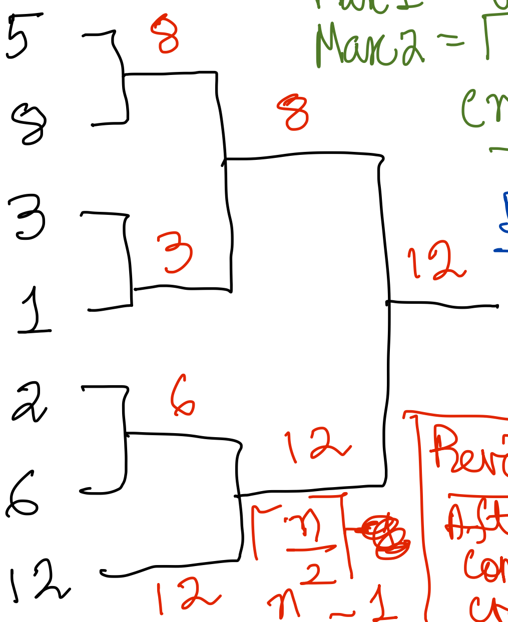


Follow the path back of Max 1 in the Tournament

Observation :-
Max 2 must have been compared with Max 1 in the tournament

To find Max 2, we need to only compare those elements which were compared with max 1.

Tournaments & Sorting



Max 1 = $(n-1)$

Max 2 = $\lceil \log n \rceil - 1$

$(n + \log n - 2)$

Data Structure

Tournament comparison Tree

Revisit Max & Min

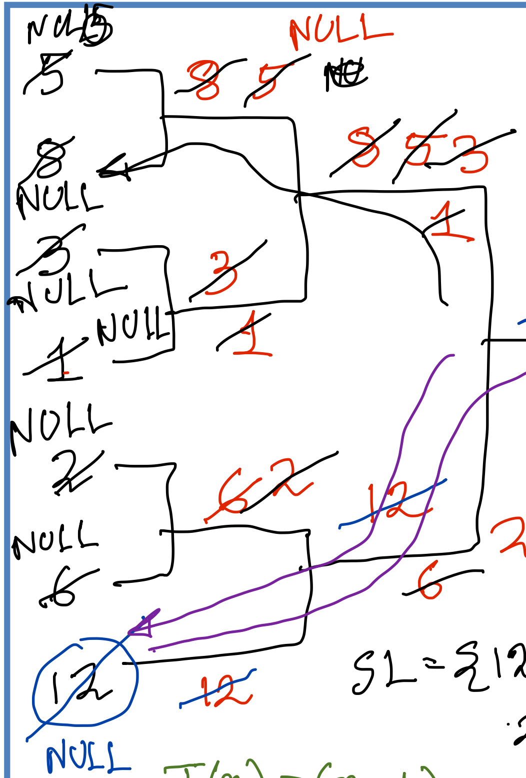
After pairwise comparison we create 2 sets : W, L

$\lceil \frac{n}{2} \rceil$

$\frac{n}{2} - 1$

$\frac{n}{2} - 1$

Balanced Tournament : height will be $\lceil \log n \rceil$



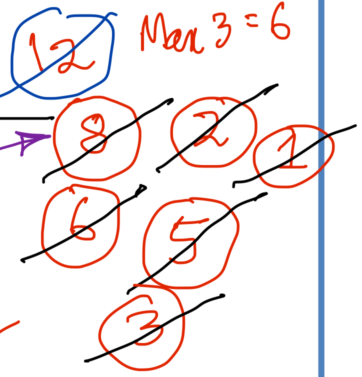
Max 1 = 12

Min = 1

NULL = smaller than smallest

Max 2 = 8

Max 3 = 6



SL = {12, 8, 6, 5, 3, 2, 1} $O(n \log n)$

$T(n) = (n-1) + n(\log n) = n \log n$

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 furthure 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

$((n-1) + \lceil \log n \rceil)$ SORTING

Algorithms + Data Structures = Progams

Max: Use a temporary variable "max"

Is there a better algorithm possible?

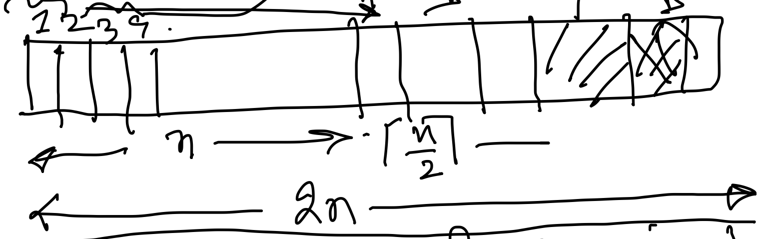
PROVE: Finding the largest by comparison using only a comparision operation cannot be done in less than $(n-1)$ comparisons for n numbers

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 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

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?