CS10003:
Programming & Data Structures

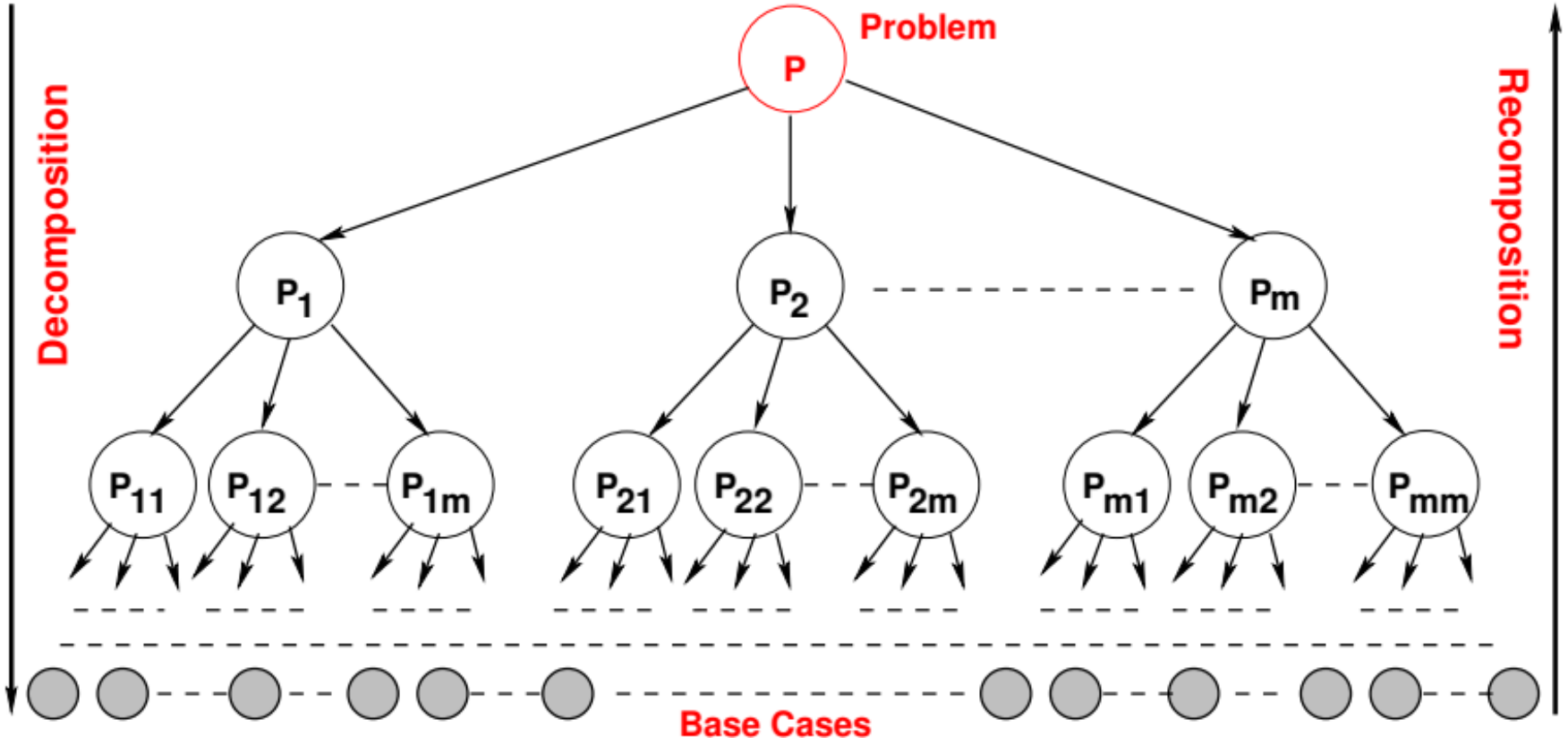
Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Autumn 2020

Problem Solving using Recursion

Solving Problems *Recursively*

- Top-Down Approach:
  - ***Base Cases:*** elementary problem instances with known solutions
  - ***Decompose:*** split problems into smaller instance sub-problems
  - ***Recursive Calls:*** call same function recursively for each sub-problems
  - ***Recompose:*** combine solutions from sub-problems to get result



Problem-0: *Sum of Squares* (Recall ...)

- **Sum of squares within range [m,n], i.e.**

$$m^2 + (m+1)^2 + (m+2)^2 + \ldots + (n-1)^2 + n^2$$

- ***Solution:***

```
int sumSquares (int m, int n)
{
      int middle ;
      if (m == n) return m*m;
      else
      {
            middle = (m+n)/2;
            return sumSquares(m,middle)
                  + sumSquares(middle+1,n);
      }
}
```

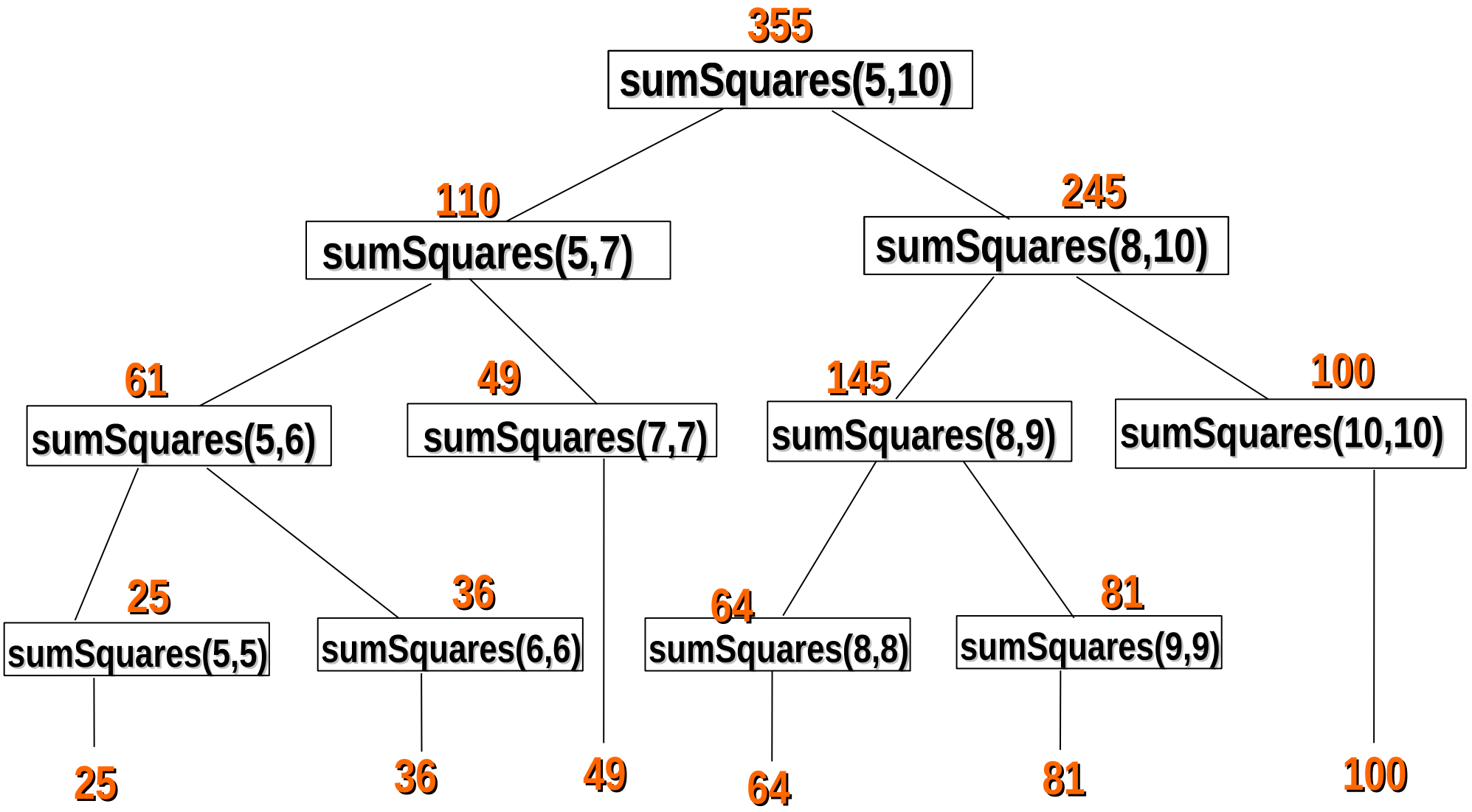
Base

Decompose

Recurse

Recompose

Problem-0: *Annotated Recursion Tree*



Problem – 1 : *Searching an Element*

- Function, find (L, n, x), to find element x from unordered list L of n elements
  - ***Base Case:*** if n == 1, then compare the only element with x and return
  - ***Decompose:*** split list, L, into two parts, L_1 and L_2, having n_1 and n_2 elements each ($n_1 + n_2 = n$), respectively
  - ***Recursive Calls:*** $f_1 \leftarrow$ find (L_1, n_1, x) and $f_2 \leftarrow$ find (L_2, n_2, x)
  - ***Recompose:*** if (f_1 || f_2), then return found, else return not-found

- Points to Ponder:
  - *Does the split position really matters?*
    - Middle, Arbitrary, Always 1 and (n-1) sized-division … etc.
  - *The naive approach is a special case of this recursive solution*
    - Always split into 1 and (n-1) sized lists (think!)
  - *Need to compare n times to find x among n elements*

Problem – 2 : *Finding MAX*

- Function, **max (L, n)**, to find maximum from a list **L** of **n** elements:
  - ***Base Case:*** if n == 1, then return the only element
  - ***Decompose:*** split list, L, into two parts, L_1 and L_2, having n_1 and n_2 elements each ($n_1 + n_2 = n$), respectively
  - ***Recursive Calls:*** $m_1 \leftarrow \max(L_1, n_1)$ and $m_2 \leftarrow \max(L_2, n_2)$
  - ***Recompose:*** if ($m_1 > m_2$), then return m_1, else return m_2

- Points to Ponder:
  - *Does the split position really matters?*
    - Middle, Arbitrary, Always 1 and (n-1) sized-division … etc.
  - *The naive approach is a special case of this recursive solution*
    - Always split into (n-1) and 1 sized lists (think!)
  - *Need to compare (n-1) times to get maximum among n elements*

Problem – 3: *Finding MAX+MIN*

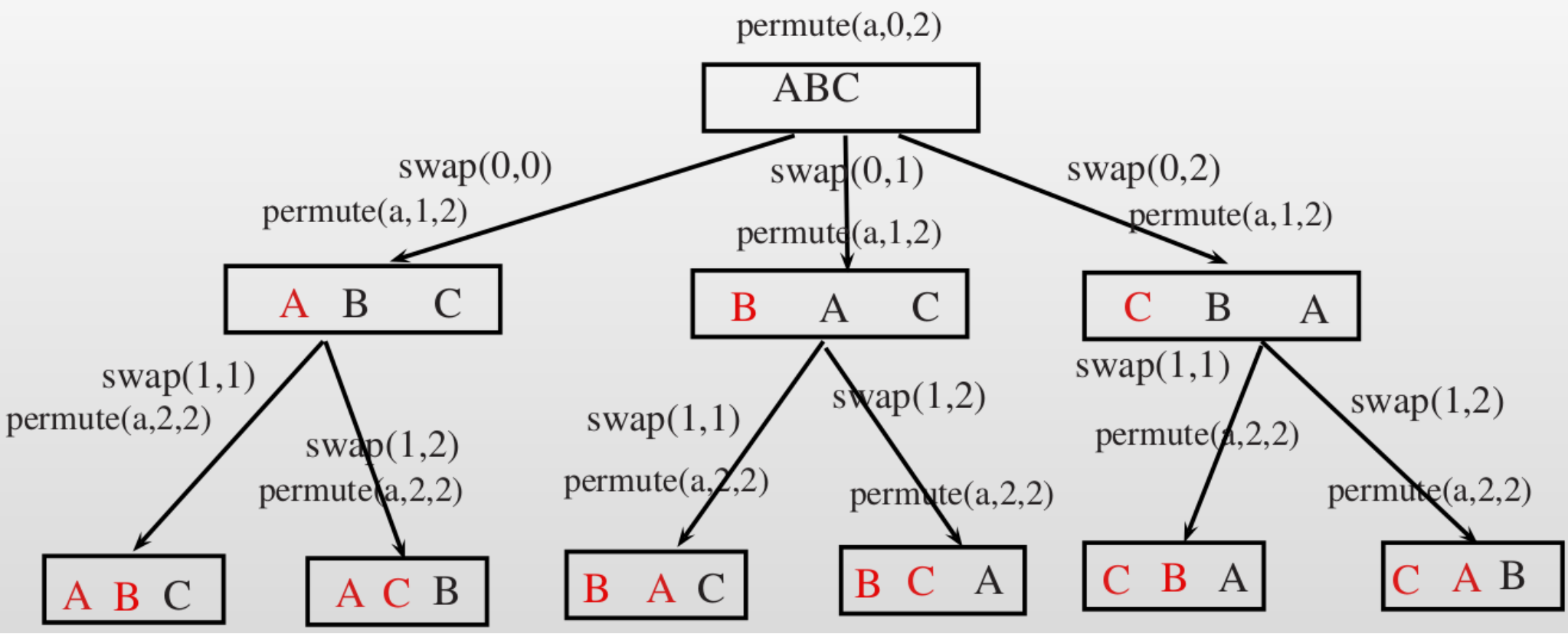
- Function, **maxmin (L, n)**, to find max+min from list **L** of **n** elements:
  - ***Base Cases*:**
    - if n == 1, then return the only element as maximum and minimum
    - If n == 2, then compare between two elements and return maximum and minimum
  - ***Decompose*:** split list, L, into two parts, L_1 and L_2, having n_1 and n_2 elements each ($n_1 + n_2 = n$), respectively
  - ***Recursive Calls*:**
    - $h_1 \leftarrow \max(L_1, n_1)$ and $h_2 \leftarrow \max(L_2, n_2)$
    - $l_1 \leftarrow \min(L_1, n_1)$ and $l_2 \leftarrow \min(L_2, n_2)$
  - ***Recompose*:**
    - if ($h_1 > h_2$), then return h_1 as maximum, else return h_2 as maximum
    - if ($l_1 < l_2$), then return l_1 as minimum, else return l_2 as minimum

Problem – 3: *Finding MAX+MIN*

- **Points to Ponder:**
  - The naive approach is to traverse the elements twice – *once for finding max and again for finding min*
    - **Total comparisons = (n - 1) + (n - 1) = 2n - 2**
  - *Smart* Observations leads to better approach!
    - **Take two elements at a time when finding max**
    - **Total comparisons for max finding = n/2 + n/4 + … + 1 = (n – 1)**
    - **Then, the first round losers (i.e. n/2 elements) are candidates for min**
    - **Total comparisons for min finding now = (n/2 - 1)**
  - Does the split position really matters to reduce number of comparison?
    - **Check for comparisons with 1 and n-1 split → 2(n - 1)**
    - **Check for comparisons with 2 and n-2 split → 3n/2 - 2**
    - **Check for comparisons with middle split → 3n/2 - 2**

Problem – 4: *Generate all Permutations of String*

- **Strategy:**



- **Recursive Formulation and Solution:**
  - **Do Yourself ! :-)**

Thank You!