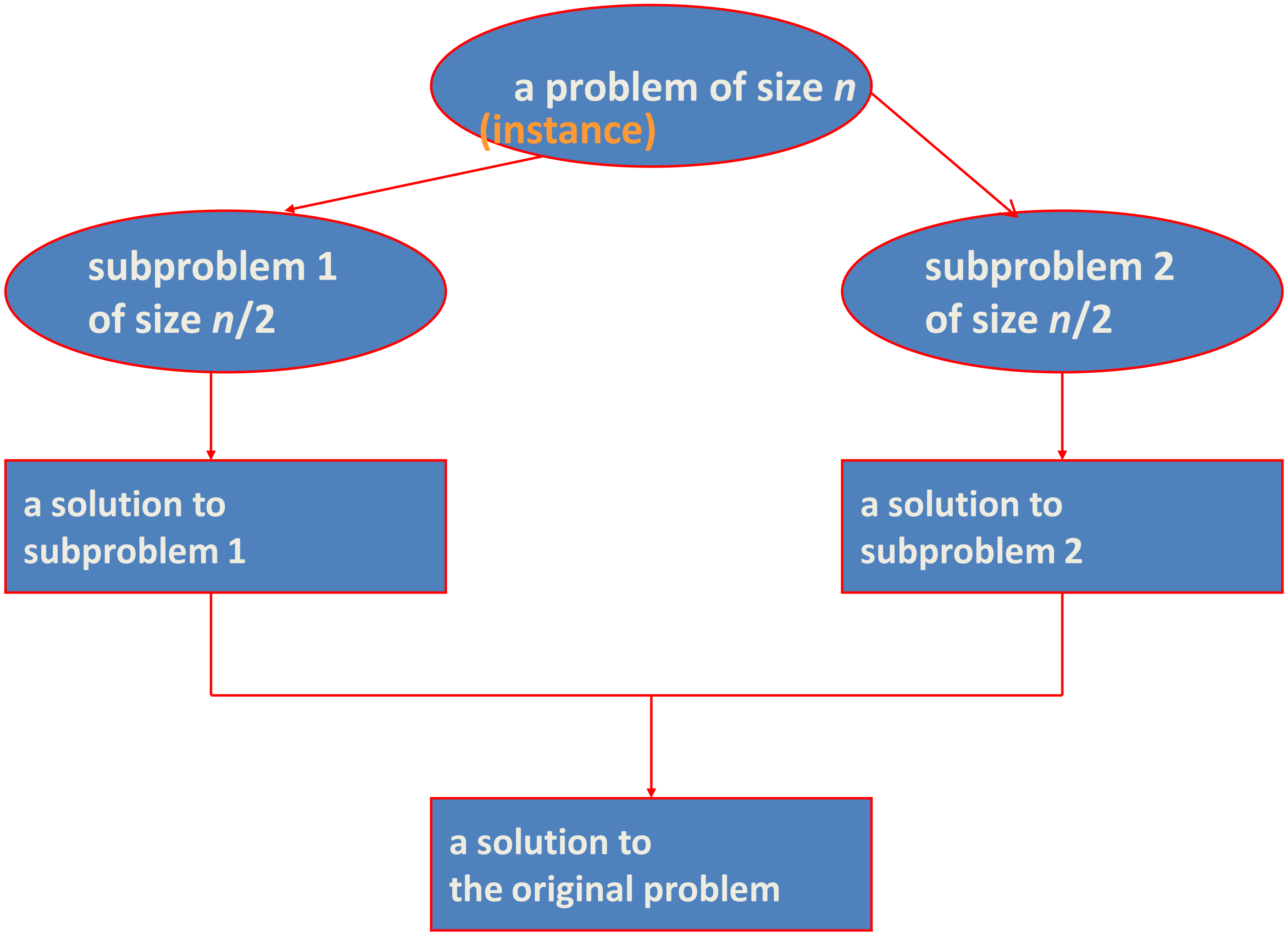
SNS College of Engineering
Coimbatore - 641107

Divide & Conquer

Introduction

The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances
2. Solve smaller instances recursively
3. Obtain solution to original (larger) instance by combining these solutions


a problem of size *n*
(instance)
subproblem 1
of size *n*/2
subproblem 2
of size *n*/2
a solution to
subproblem 1
a solution to
subproblem 2
a solution to
the original problem

Examples

- Sorting: merge sort and quick sort
- Binary search
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms

General Divide-and-Conquer Recurrence

$T(n) = aT(n/b) + f(n)$ where $f(n) \in \Theta(n^d)$, $d \geq 0$

Master Theorem: If $a < b^d$, $T(n) \in \Theta(n^d)$

If $a = b^d$, $T(n) \in \Theta(n^d \log n)$

If $a > b^d$, $T(n) \in \Theta(n^{\log_b a})$

Note: The same results hold with O instead of Θ.

Examples: $T(n) = 4T(n/2) + n \Rightarrow T(n) \in$? **Θ(n^2)**

$T(n) = 4T(n/2) + n^2 \Rightarrow T(n) \in$? **Θ(n^2log n)**

$T(n) = 4T(n/2) + n^3 \Rightarrow T(n) \in$? **Θ(n^3)**

Algorithm

```
Algorithm DC(p)
{
If p is too small then
return solution of p
{
divide (p) & obtain p1,p2,…pn
where n>=1
Apply DC to each sub problem
return combine(DC(p1), DC(p2),…DC(pn));
}
}
```

Break

- **Alphabet Exercise**

  Allocate a number of letters of the alphabet to each group and then give them 5 minutes to find as many objects as possible they can point to in the room that begins with one of their allocated letters. No real rules so you find some people can get very creative, bringing items in from outside the room, naming parts of the body (e.g. pupil/cornea..), emptying their handbags to find items, etc. Creates great energy and fun.

Mergesort

- Split array A[0..*n*-1] into about equal halves and make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into array A as follows:
  - Repeat the following until no elements remain in one of the arrays:
    - compare the first elements in the remaining unprocessed portions of the arrays
    - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
  - Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

Algorithm of Merge sort

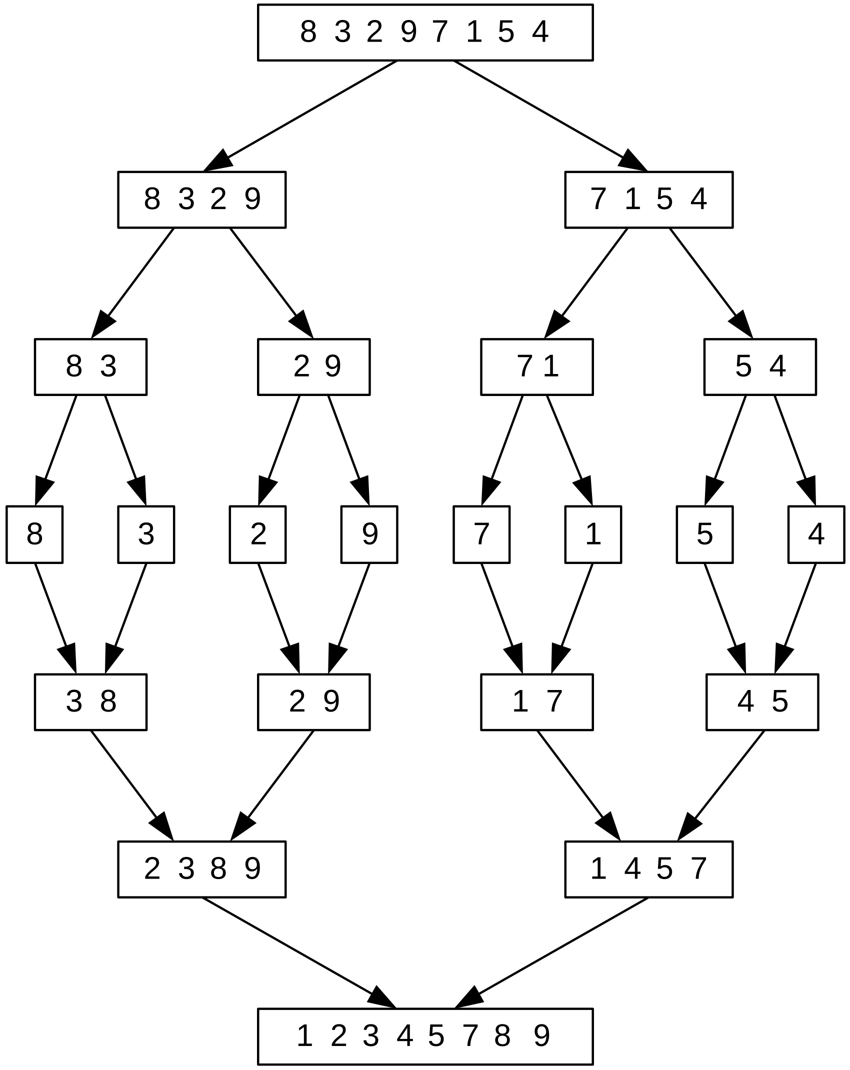
- \\ Problem Description: Merge 2 sorted array into one sorted array.
- \\ Input: Array A(0---n-1,low,high)
- \\ Output: Sorted array of element

```
If(low<high)then
{
Mid← (low+high)/2
Merge sort(A, low, mid)
Merge sort(A,Mid+1,high)
Combine(A,low,mid,high)
}
Algorithm combine(A[0—n-1], low, mid, high)
{
 K←low;
i←low;
J←mid+1;
```

Algorithm of Merge

```
While(i<=mid & j<= high) do
{
If(A[i]<=A[j]) then
{
Temp[k]←A[i]
i←i+1
K←k+1
}
else
{
Temp[k]←A[j]
J←j+1
K←k+1
}
}
```

Mergesort Example



The non-recursive version of Merge sort starts from merging single elements into sorted pairs.

Analysis of Mergesort

- All cases have same efficiency: $\Theta(n \log n)$

  $T(n) = 2T(n/2) + \Theta(n), T(1) = 0$

- Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting:

  $$\lceil \log_2 n! \rceil \approx n \log_2 n - 1.44n$$

- Space requirement: $\Theta(n)$ (<u>not</u> in-place)

- Can be implemented without recursion (bottom-up)

5 3 8 2 9 1 7 4