Computer Algorithms

Lecture 5: Divide-and-Conquer – Ch 5

Dr. Manal Helal, Spring 2014.

Lecture Learning Objectives

1. Use a Divide & Conquer algorithm design strategy to solve an appropriate problem such as sorting.

Divide-and-Conquer

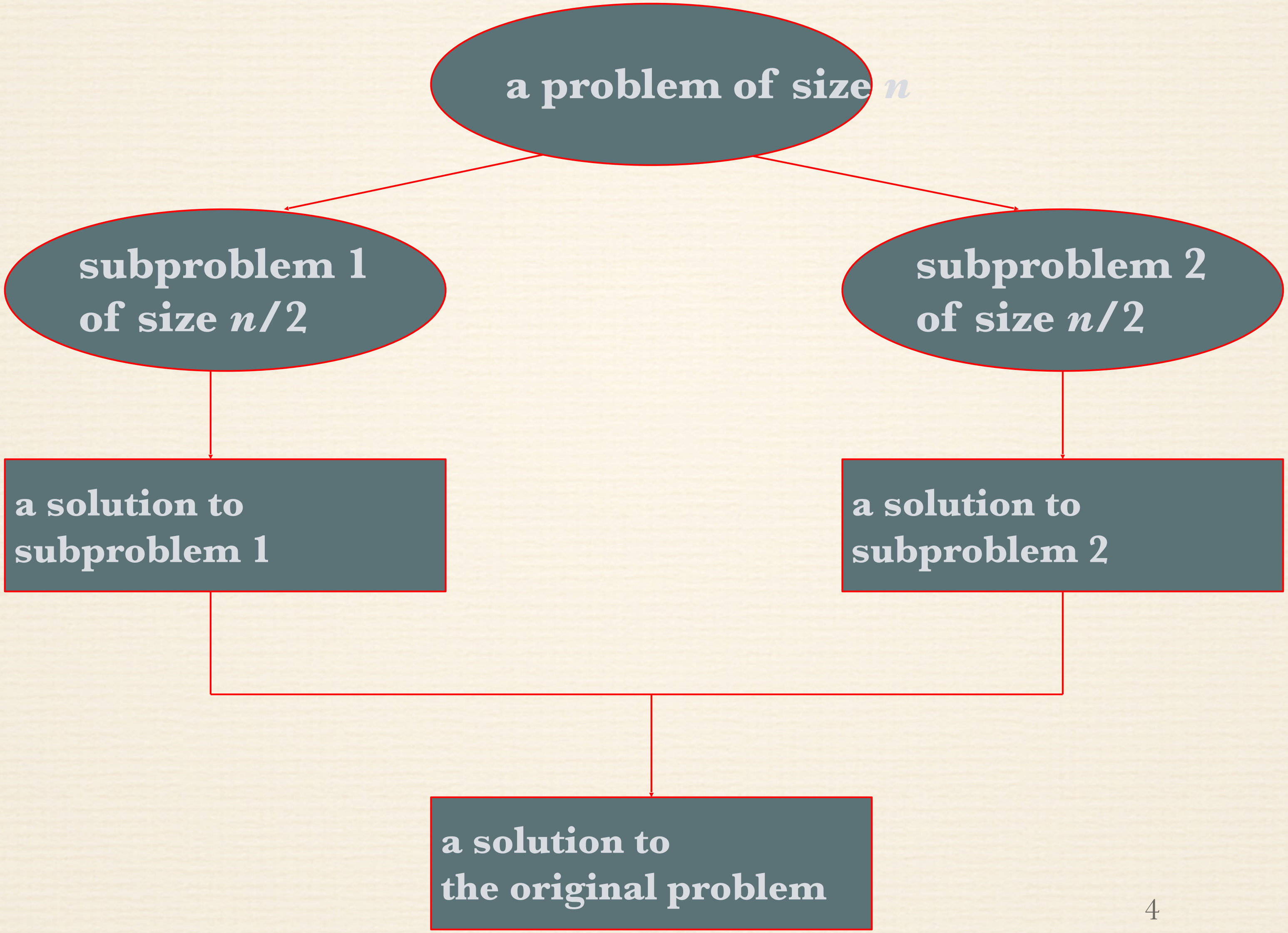
The most-well known algorithm design strategy:

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

Divide-and-Conquer Technique



Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen's algorithm
- Closest-pair and convex-hull algorithms

- Binary search: decrease-by-half (or degenerate divide&conq.)

General Divide-and-Conquer Recurrence

$$T(n) = aT(n/b) + f(n),$$

n can be divided into b instances of size n/b, with a of them need to be solved.
$f(n)$ is a function that accounts for the time spent dividing and combining.

Master Theorem If $f(n) \in \Theta(n^d)$ where $d \geq 0$ in recurrence (5.1), then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Analogous results hold for the O and Ω notations, too.

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

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

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

Mergesort

- Split array A[0..n-1] in two 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.

Pseudocode of Mergesort

ALGORITHM *Mergesort*$(A[0..n-1])$

//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$
Mergesort$(C[0..\lceil n/2 \rceil - 1])$
Merge(B, C, A) //see below

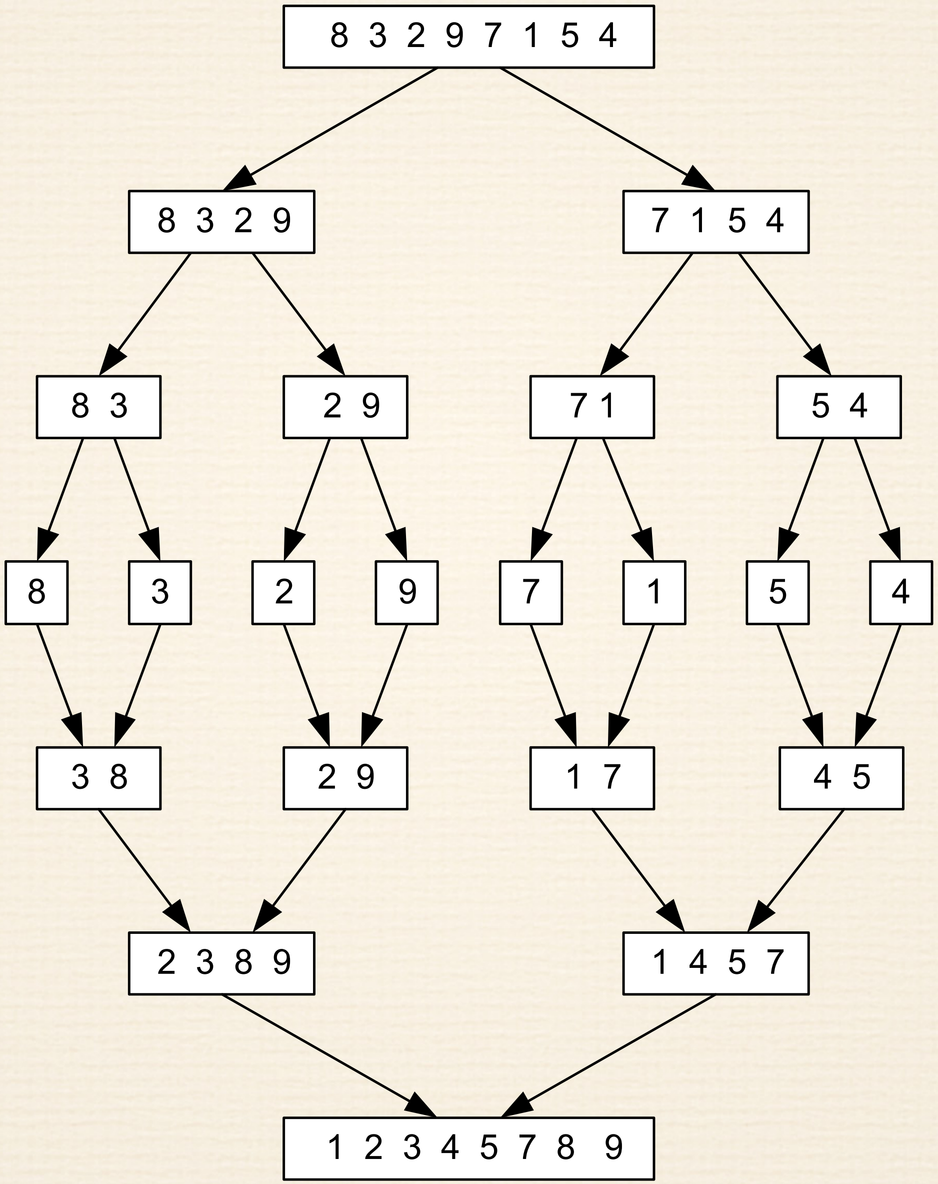
Pseudocode of Merge

ALGORITHM $Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C

$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
  if $B[i] \leq C[j]$
    $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$
  else $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$
  $k \leftarrow k + 1$
if $i = p$
  copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Mergesort Example

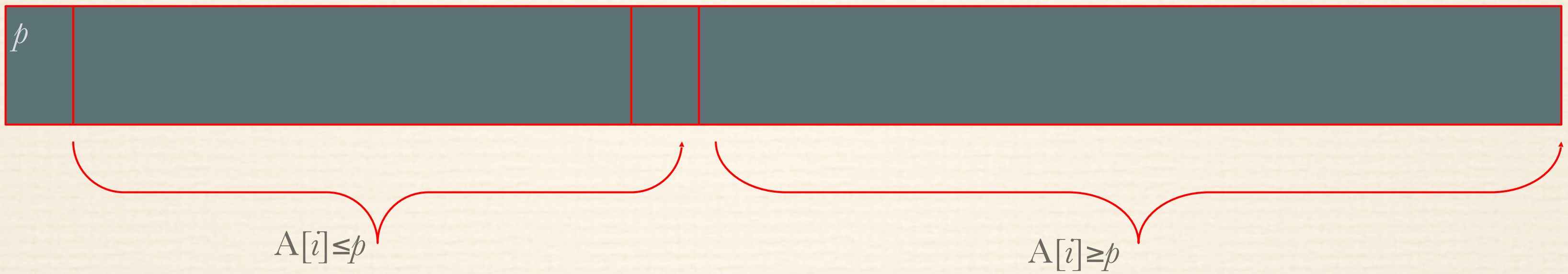


Analysis of Mergesort

- $C(n) = 2C(n/2) + C_{merge}(n)$ for $n > 1$, $C(1) = 0$.
- $C_{merge}(n) = n - 1$
- All cases have same efficiency: $\Theta(n \log n)$
- 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)

Quicksort

- Select a *pivot* (partitioning element) – here, the first element
- Rearrange the list so that all the elements in the first *s* positions are smaller than or equal to the pivot and all the elements in the remaining *n-s* positions are larger than or equal to the pivot (see next slide for an algorithm)



- Exchange the pivot with the last element in the first (i.e., ≤) subarray — the pivot is now in its final position
- Sort the two subarrays recursively

Quicksort Pseudo-code

ALGORITHM *Quicksort*$(A[l..r])$

//Sorts a subarray by quicksort
//Input: Subarray of array $A[0..n-1]$, defined by its left and right
//   indices l and r
//Output: Subarray $A[l..r]$ sorted in nondecreasing order
if $l < r$
    $s \leftarrow Partition(A[l..r])$ //s is a split position
    $Quicksort(A[l..s-1])$
    $Quicksort(A[s+1..r])$

Hoare's Partitioning Algorithm

```
ALGORITHM  HoarePartition(A[l..r])
    //Partitions a subarray by Hoare's algorithm, using the first element
    //        as a pivot
    //Input: Subarray of array A[0..n − 1], defined by its left and right
    //        indices l and r (l < r)
    //Output: Partition of A[l..r], with the split position returned as
    //        this function's value
    p ← A[l]
    i ← l; j ← r + 1
    repeat
        repeat i ← i + 1 until A[i] ≥ p
        repeat j ← j − 1 until A[j] ≤ p
        swap(A[i], A[j])
    until i ≥ j
    swap(A[i], A[j])   //undo last swap when i ≥ j
    swap(A[l], A[j])
    return j
```

5	3	1	9	8	2	4	7

Analysis of Quicksort

$C_{worst}(n) = (n + 1) + n + \ldots + 3 = ((n + 1)(n + 2) / 2) - 3 \in (n^2).$

Worst case: sorted array! — $\Theta(n^2)$

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0.$$

Average case: random arrays — $\Theta(n \log n)$

Best case: split in the middle — $\Theta(n \log n)$

Improvements:

- better pivot selection: median of three partitioning
- switch to insertion sort on small subfiles
- elimination of recursion
- These combine to 20-25% improvement

Considered the method of choice for internal sorting of large files ($n \geq 10000$)