Divide and Conquer

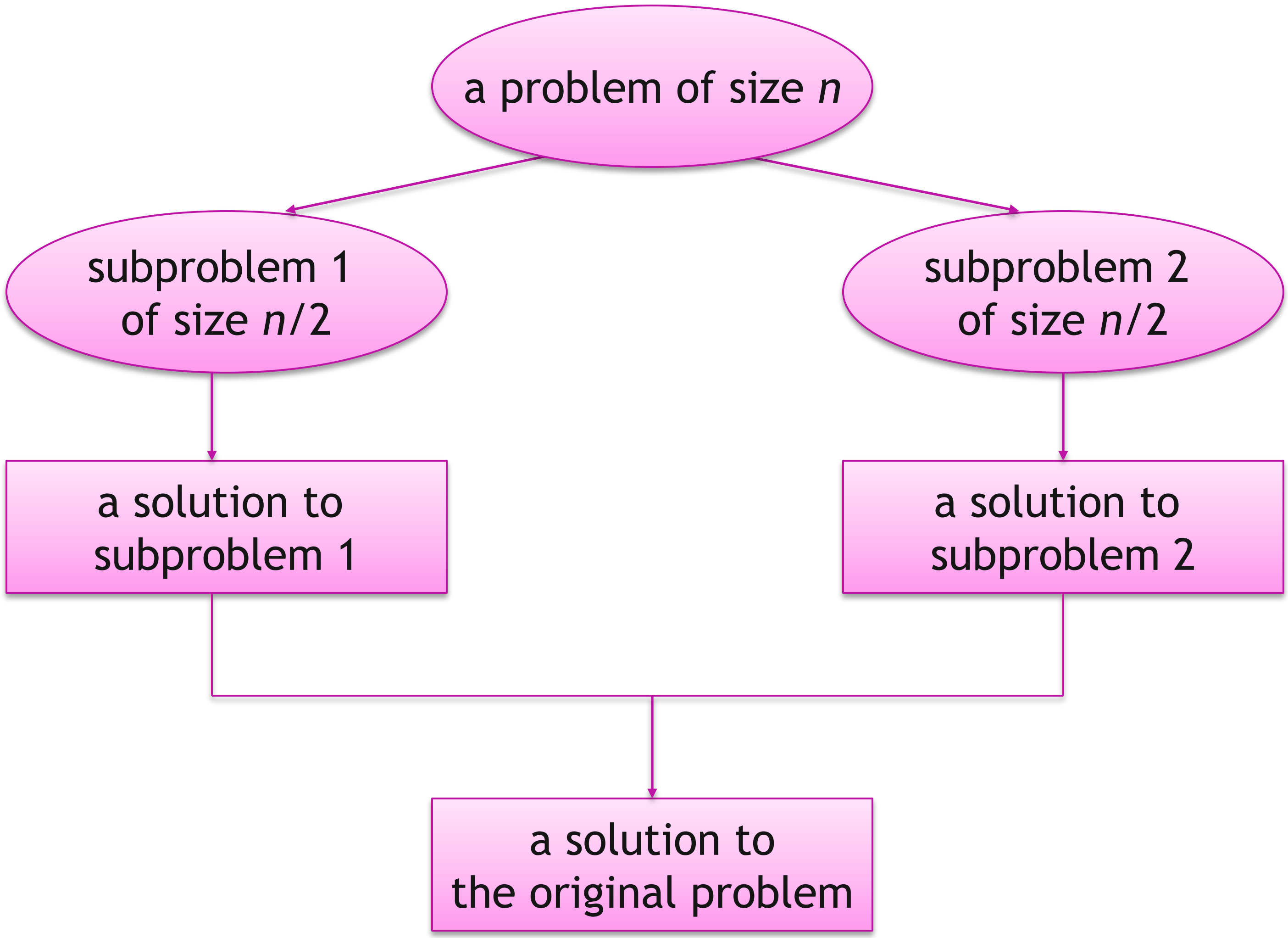
Chapter 4

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 (if necessary)

Divide-and-conquer technique



Divide and Conquer Examples

- Mergesort
- Quicksort
- Binary search
- Multiplication of Large integers
- Matrix multiplication-Strassen’s algorithm

Divide and Conquer

- *More generally, An instance of size n can be divided into several instances of size n/b with a of them need to be solved.*

with $\boldsymbol{a \geq 1}$ ***and*** $\boldsymbol{b > 1}$ *then*

- *Then* $T(n) = aT(n/b) + f(n)$

  where f(n) is the time for dividing and conquering.

This is called the ***general divide and conquer recurrence***

Master Theorem

- *IF* $f(n) \in \Theta(n^d)$ *where d>=0 in the general divide and conquer recurrence* $T(n) = aT(n/b) + f(n)$ with a>=1 and b>1 then,

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

- Ex: Consider the problem of computing the sum of n numbes recursively.

  $a_0+a_1+a_2+\ldots. +a_{n-1} = (a_0+\ldots+.a_{n/2}) + (a_{n/2}+\ldots. +a_{n-1})$

  $A(n) = 2A(n/2)+1$

  a=2, b=2 and d=0

  $a > b^d$

  Therefore $A(n) \in \Theta(n^{\log_b a}) = \Theta(n)$

Merge Sort

Algorithm:

- Split array A[1..n] in two and make copies of each half

 in arrays B[1.. $n/2$] & C[1.. $n/2$]

- Sort arrays B and C
- 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.

Mergesort

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

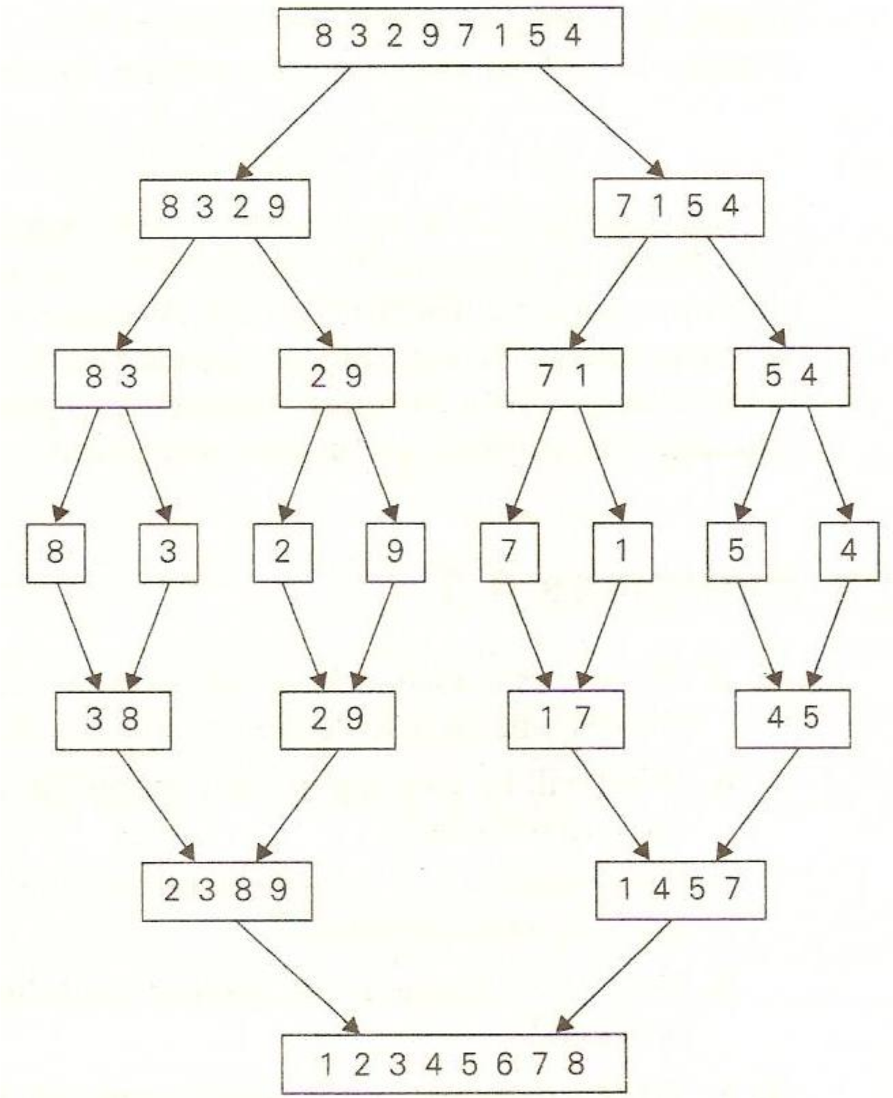
//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
  copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
  copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2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

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
i ← 0; j ← 0; k ← 0
while i < p and j < q do
    if B[i] ≤ C[j]
        A[k] ← B[i]; i ← i + 1
    else A[k] ← C[j]; j ← j + 1
    k ← 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



Efficiency of mergesort

- Assuming n is a power of 2

$$C(n) = 2C(n/2) + C_{merge}(n) \text{ for } n > 1, \; C(1) = 0.$$

- For the worst Case

$$C_{merge}(n) = n - 1,$$

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \text{ for } n > 1, \; C_{worst}(1) = 0.$$

$$C_{worst}(n) = \Theta(n \log n)$$

Quicksort

- Merge sort : Input array is divided based on position.
- Quick Sort : Input array is divided based on their value.
- Method :
  - Partition the given array of size l to r using the pivot element say a[s] such that all the elements towards the left of pivot are <= pivot and all the elements towards the right of pivot are >= pivot.
  - Now we have 2 sub arrays of size l-(s-1) and (s+1)-r
  - Repeat the process for sub arrays.
  - Left to right scan : if A[i]<=p   i=i+1
  - Right to left scan : if A[j]>=p   j=j-1

Quicksort

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

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

The partition algorithm

ALGORITHM *Partition(A[l….r])*

// Partitions a sub array by using its first element as a pivot.

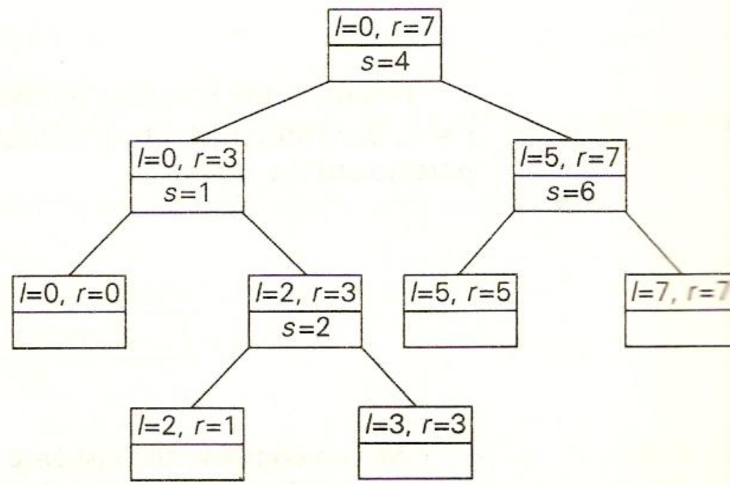
// Input : A Sub array A[l….r] of A[0….n-1], defined by its left and right indices l and r (l<r).

// Output : A partition of A[l…r], with the split position returned as this function's value.

```
p← A[l]
i ←l ; j ← r+1
while (true)
    repeat i ← i+1 until A[i]≥ p
    repeat j ← j-1 until A[j]≤ p
    if(i<j) then Swap (A[i],A[j])
    else
        Swap (A[l],A[j])
        return j
```

Quicksort Example

0	1	2	3	4	5	6	7
5	$\overset{i}{3}$	1	9	8	2	4	$\overset{j}{7}$
5	3	1	$\overset{i}{9}$	8	2	$\overset{j}{4}$	7
5	3	1	$\overset{i}{4}$	8	2	$\overset{j}{9}$	7
5	3	1	4	$\overset{i}{8}$	$\overset{j}{2}$	9	7
5	3	1	4	$\overset{i}{2}$	$\overset{j}{8}$	9	7
5	3	1	4	$\overset{j}{2}$	$\overset{i}{8}$	9	7
2	3	1	4	**5**	8	9	7
2	$\overset{i}{3}$	1	$\overset{j}{4}$				
2	$\overset{i}{3}$	$\overset{j}{1}$	4				
2	$\overset{i}{1}$	$\overset{j}{3}$	4				
2	$\overset{j}{1}$	$\overset{i}{3}$	4				
1	**2**	3	4				
1							
		3	$\overset{ij}{4}$				
		$\overset{j}{\mathbf{3}}$	$\overset{i}{4}$				
			4				
					8	$\overset{i}{9}$	$\overset{j}{7}$
					8	$\overset{i}{7}$	$\overset{j}{9}$
					8	$\overset{j}{7}$	$\overset{i}{9}$
					7	**8**	9
					7		
							9



(b)

Partition Algorithm from Text Book

The partition algorithm

```
ALGORITHM  Partition(A[l..r])
    //Partitions a subarray by using its first element as a pivot
    //Input: A subarray A[l..r] of A[0..n − 1], defined by its left and right
    //        indices l and r (l < r)
    //Output: A 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
```

Design and Analysis of Algorithms - Chapter 4 13

Note : Text book copy of the algorithm

Efficiency of quicksort

- *<u>Best case</u>*: split in the middle — $\Theta(n \log n)$
- *<u>Worst case</u>*: sorted array — $\Theta(n^2)$
- *<u>Average case</u>*: random arrays — $\Theta(n \log n)$

Binary Search

ALGORITHM *BinarySearch*$(A[0..n-1], K)$

//Implements nonrecursive binary search
//Input: An array $A[0..n-1]$ sorted in ascending order and
// a search key K
//Output: An index of the array's element that is equal to K
// or -1 if there is no such element
$l \leftarrow 0;\ r \leftarrow n-1$
while $l \le r$ **do**
    $m \leftarrow \lfloor (l+r)/2 \rfloor$
    if $K = A[m]$ **return** m
    else if $K < A[m]$ $r \leftarrow m-1$
    else $l \leftarrow m+1$
return -1

Multiplication of Large Integers

- Ex: 29 * 15 = 435
- $29 = 2*10^1 + 9*10^0$ and $15 = 1*10^1 + 5*10^0$

$$\begin{aligned} 29 * 15 &= (2*10^1 + 9*10^0) * (1*10^1 + 5*10^0) \\ &= (2*1)10^2 + (9*1 + 2*5)10^1 + (9*5)10^0 \\ &= 200 \quad + \quad 190 \quad + \quad 45 \\ &= 435 \end{aligned}$$

- We can reduce 2 multiplication in the middle term with one multiplication.
- $(9*1 + 2*5) = (2+9)*(1+5) - \underbrace{(2*1)}_{} - \underbrace{(9*5)}_{}$

Previously calculated

Multiplication of Large Integers

- We can obtain the following formula for any pair of two digit numbers $a=a_1a_0$ & $b=b_1b_0$ their product c can be computed by the following formula.
- $C = a*b = c_2 10^2 + c_1 10^1 + c_0$
- $C2 = a_1*b_1$ product of their first digits
- $C0 = a_0*b_0$ product of their second digits
- $C_1 = \underbrace{(a_1+a_0)}_{\text{Sum of a's digits}} * \underbrace{(b_1+b_0)}_{\text{Sum of b's digits}} - (c_2+c_0)$

Sum of a's digits Sum of b's digits

Multiplication of Large Integers

For 2 n-digit integers where n is positive even number.

$a=a_1a_0 \quad b=b_1b_0$

First half of a $\rightarrow a_1$ First half of b $\rightarrow b_1$

Second half of a $\rightarrow a_0$ Second half of b $\rightarrow b_0$

$a=a_1a_0 \qquad b=b_1b_0$

$\Rightarrow a=a_1 10^{n/2} + a_0 \qquad \Rightarrow b=b_1 10^{n/2} + b_0$

$C=a*b = (a_1 10^{n/2} + a_0)*(b_1 10^{n/2} + b_0)$

$= (a_1*b_1)10^n +(a_1*b_0+a_0*b_1)10^{n/2}+(a_0*b_0)$

$= c_2 10^n + c_1 10^{n/2} +c_0$

Where

$C2 = a_1*b_1$ product of their first halves

$C0 = a_0*b_0$ product of their second halves

$C_1 = (a_1+a_0) * (b_1+b_0) - (c_2+c_0)$

Sum of a's digits Sum of b's digits

Strassen's matrix multiplication

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

$$= \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}$$

Strassen's matrix multiplication

- $m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$
- $m_2 = (a_{10} + a_{11}) * b_{00}$
- $m_3 = a_{00} * (b_{01} - b_{11})$
- $m_4 = a_{11} * (b_{10} - b_{00})$
- $m_5 = (a_{00} + a_{01}) * b_{11}$
- $m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$
- $m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

Strassen's matrix multiplication

$$\begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} * \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

C_{00} can be computed either as $A_{00} * B_{00} + A_{01} * B_{10}$ or as M1 + M4 – M5 + M7 (using Strassen's formulas in which numbers are replaced by corresponding sub-matrices).

Efficiency of Strassen's algorithm

If M(n) is the number of multiplications made by   in multiplying 2 n-by-n matrices,

$$M(n) = 7M(n/2) \text{ for } n > 1, \ M(1) = 1.$$

Since $n = 2^k$,

$$M(2^k) = 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2M(2^{k-2}) = \cdots$$
$$= 7^i M(2^{k-i}) \cdots = 7^k M(2^{k-k}) = 7^k.$$

Since $k = \log_2 n$,

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807},$$

The End

Thank You