CS146: Data Structures and Algorithms
Lecture 4

DIVIDE AND CONQUER- MERGE SORT & MATRIX MULTIPLICATION

INSTRUCTOR: KATERINA POTIKA

CS SJSU

Designing algorithms

- 1st Technique: Divide and conquer
  - **Divide** the problem into a number of sub-problems.
  - **Conquer** the sub-problems by solving them recursively.
    - ***Base case:*** If the sub-problems are small enough, just solve them by brute force.
  - **Combine** the sub-problem solutions to give a solution to the original problem

Mergesort

- Algorithm 2: *Mergesort*
  - Split the input into 2 parts.
  - Recursively sort each of them.
  - Merge the two sorted parts.

Mergesort – more details

- Each sub-problem as sorting a sub-array $\boldsymbol{A[p\,..\,r]}$.
  - Initially, $p = 1$ and $r = n$, but these values change as we recursively solve sub-problems.
- To sort $A[p\,..\,r]$:
  - **Divide** by splitting into two sub-arrays
    - $A[p\,..\,q]$
    - $A[q + 1\,..\,r]$, where q is the halfway point of $A[p\,..\,r]$.
  - **Conquer** by *recursively* sorting the two sub-arrays $A[p\,..\,q]$ and $A[q + 1\,..\,r]$.
  - **Combine** by merging the two sorted sub-arrays $A[p\,..\,q]$ and $A[q + 1\,..\,r]$ to produce a single sorted sub-array $A[p\,..\,r]$.
    - MERGE*(A, p, q, r)* // basic "sort" operation
- The recursion ends when the sub-array has just 1 element, so that it's trivially sorted.

How do we merge?

- Input: 2 sorted sub-array *A[p..q]* and *A[q+1..r]*
- Output: A sorted sub-array *A[p..r]* which contains all the elements.

- *Merge*(*A,p,q,r*)
  - **while** there are still elements in the 2 sub-arrays **do**
  - Compare the 1st elements of the sorted 2 sub-arrays.
  - Move the minimum of them from its corresponding list to the end of output sub-array.

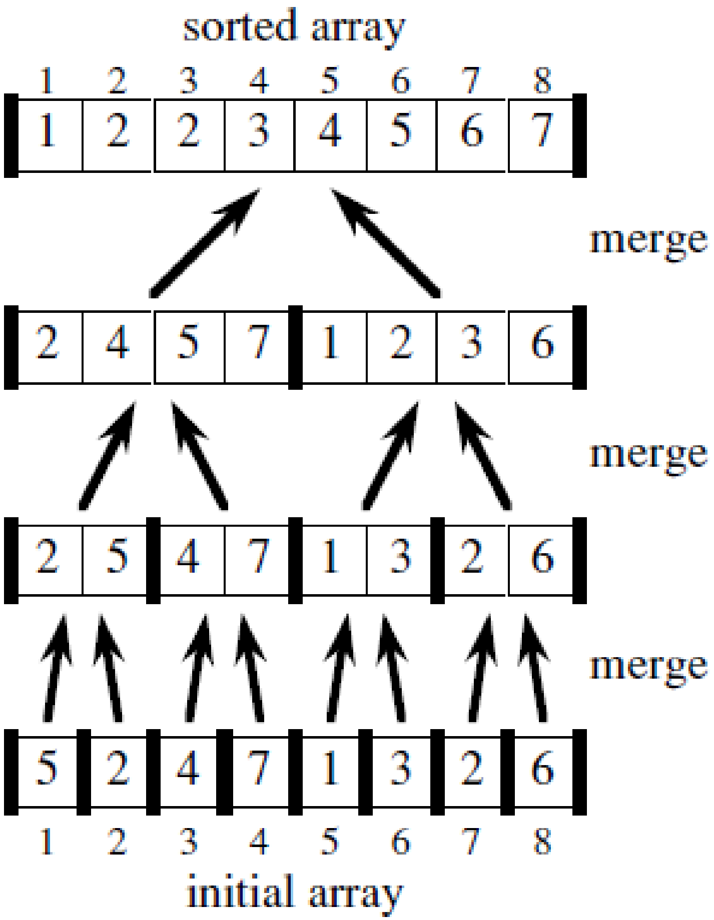
MERGE-SORT(A, p, r)

- **if** $p < r$ // Check for base case
- **then** $q \leftarrow (p + r)/2$ //Divide
  - MERGE-SORT(A, p, q) // Conquer
  - MERGE-SORT$(A, q + 1, r)$ // Conquer
  - MERGE(A, p, q, r) // Combine
- ***Initial call:*** MERGE-SORT$(A, 1, n)$

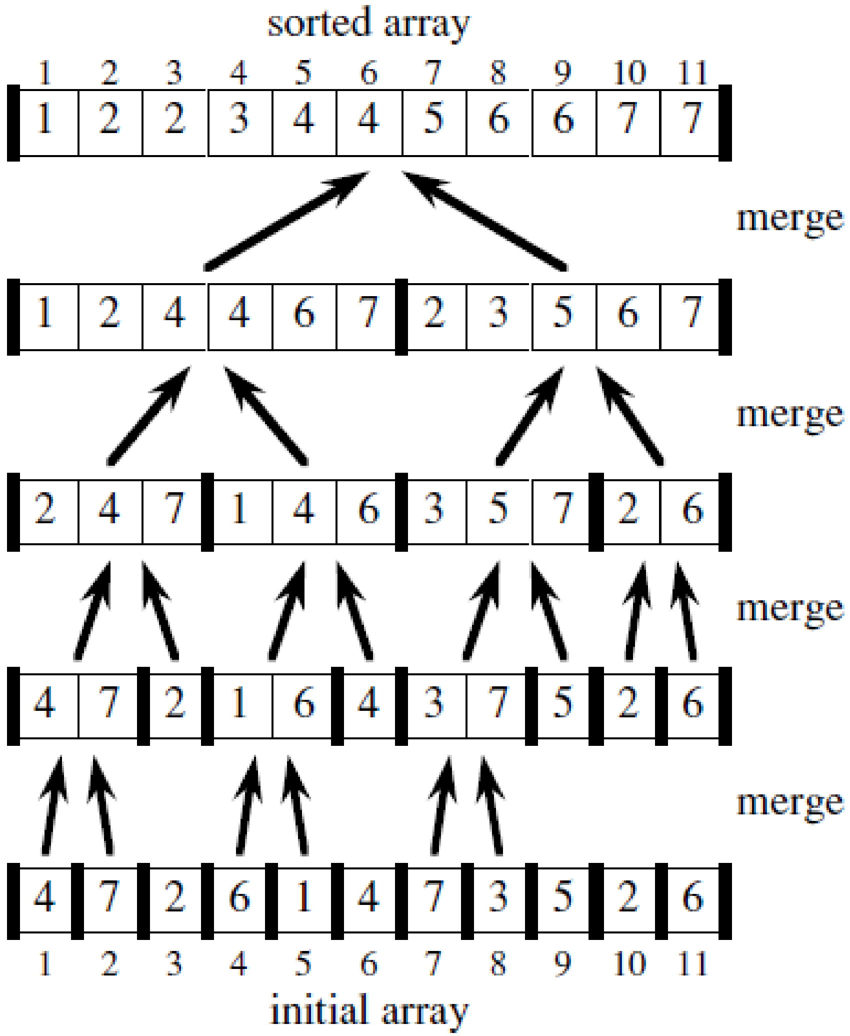
MERGE(*A, p, q, r*)

$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
create arrays $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
for $i \leftarrow 1$ **to** n_1
  do $L[i] \leftarrow A[p + i - 1]$
for $j \leftarrow 1$ **to** n_2
  do $R[j] \leftarrow A[q + j]$
$L[n_1 + 1] \leftarrow \infty$
$R[n_2 + 1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow p$ **to** r
  do if $L[i] \le R[j]$
    then $A[k] \leftarrow L[i]$
      $i \leftarrow i + 1$
    else $A[k] \leftarrow R[j]$
      $j \leftarrow j + 1$

Example with n=8



Example with n=11



Analysis of Merge Sort


```
Merge-Sort(A, p, r)                       //T(n)
  if (p < r)                              //Θ(1)
    q = ⌊(p + r)/2⌋                       //Θ(1)
    Merge-Sort(A, p, q);                  //T(n/2)
    Merge-Sort(A, q+1, r);                //T(n/2)
    Merge(A, p, q, r);                    //Θ(n)
```

Time analysis

- If the problem size is small, say c for some constant c, we can solve the problem in constant, i.e., $\Theta(1)$ time.
- Let $T(n)$ be the time needed to sort for input of size n.
- Let cn be the time needed to merge 2 lists of total size n. We know that $cn = \Theta(n)$.
- Assume that the problem can be split into 2 subproblems in constant time and that $c = 1$.

Recurrences

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

- is a *recurrence*.
  - Recurrence: an equation that describes a function in terms of its value on smaller functions

Recurrence Examples

$$s(n) = \begin{cases} 0 & n = 0 \\ s(n-1) + c & n > 0 \end{cases}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ s(n-1) + n & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\dfrac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\dfrac{n}{b}\right) + cn & n > 1 \end{cases}$$

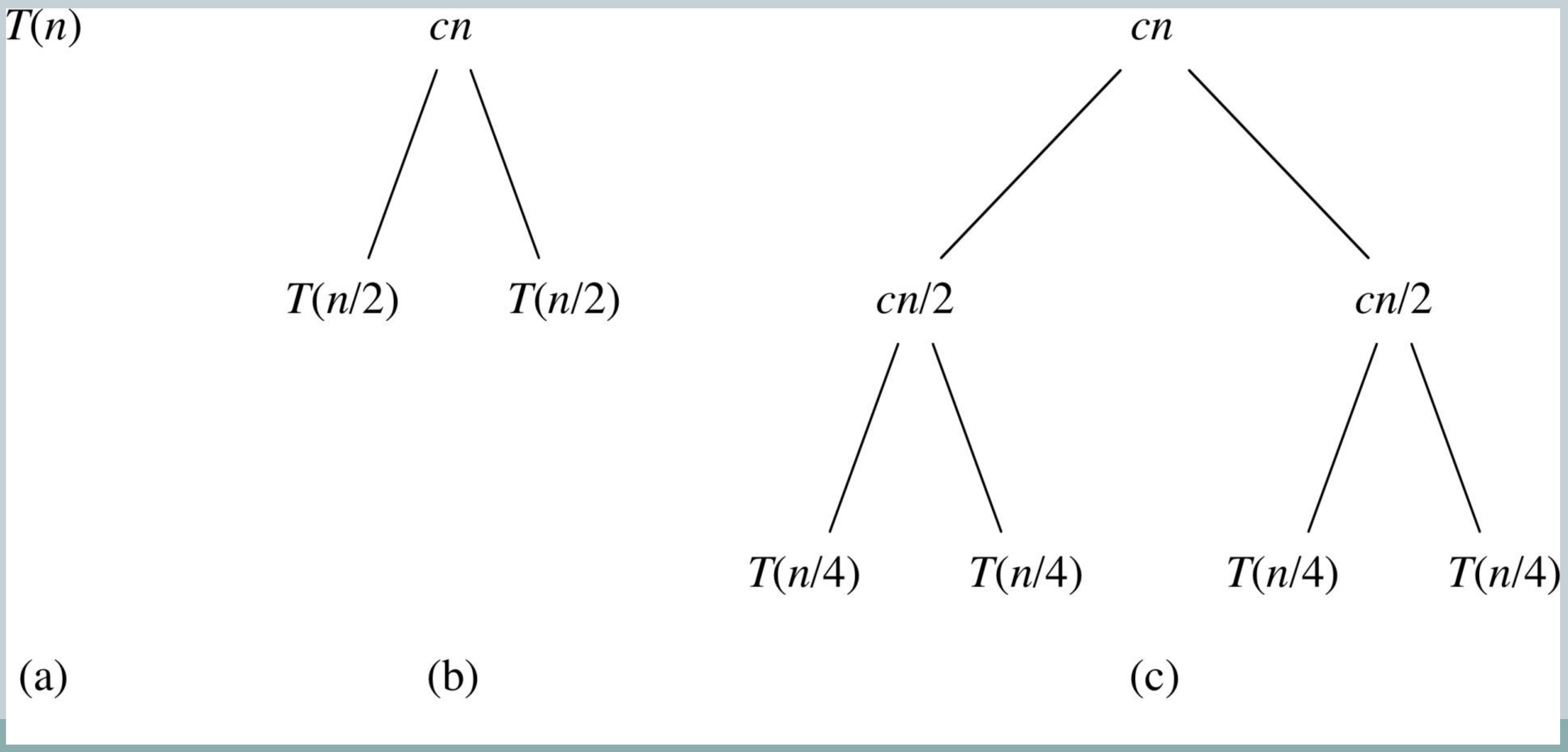
How to we find T(n)

- Question: Is there a closed form for $T(n)$?
- W.l.o.g., assume $n = 2^k$ (or, lg $n = k$).

- *Note:* $lgn = log_2 n$

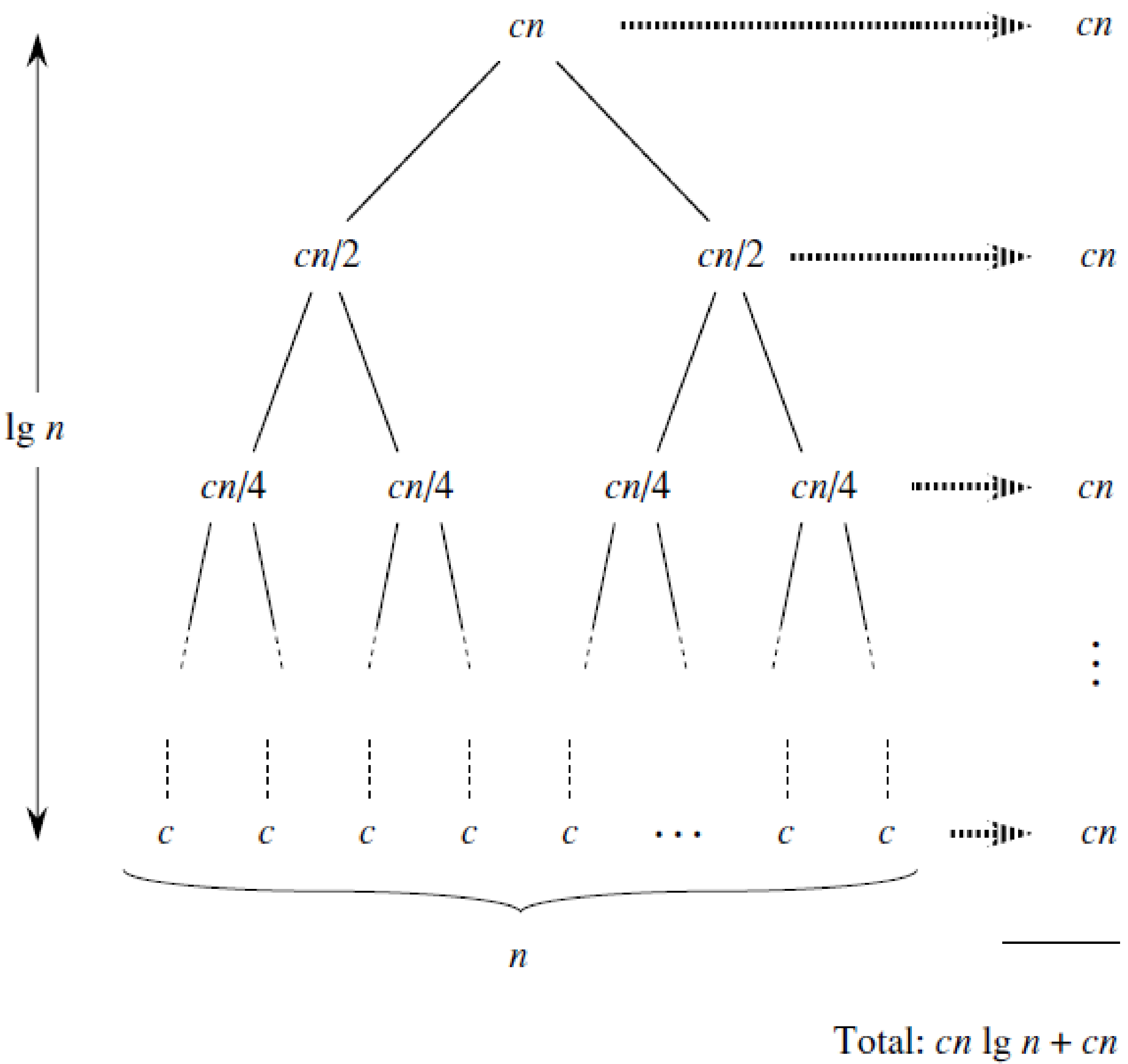
Recursion tree

- For the original problem, cost c*n+2 subproblems, each of them c*n/2 + subproblems



Recursion tree – cont.

Continue expanding until the problem sizes get down to 1:



Cost of each level

- Top level: cn
- Next level: c(n/2)+c(n/2)=cn
- Next next level: 4c(n/4)=cn

- General:
  - i-th level from top has 2^i nodes
  - each with cost c(n/ 2^i)
  - Total cost of this level: cn
  - Bottom level: n nodes, each cost c

Total number of levels

- is **lgn+1**, where n: input size (number of leaves)
- Use induction to prove this
- Base case: n=1, only one level lg1 =0
- Inductive Hypothesis: number of levels with 2^i *leaves* is $\lg 2^i + 1 = i + 1$
- Prove that for $n = 2^{i+1}$ *leaves* $(power\ of\ 2)$ one more level than with 2^i leaves, i.e. (i+1)+1=lg $2^{i+1} + 1$

Running time of Merge-sort

- $\lg n+1$ levels each with cost cn= $cn(\lg n+1)$

- Ignore lower order term and c
- $\Theta(n\lg n)$

Practice

merge sort on the array

3, 41, 52, 26, 38, 57, 9, 49

Matrix Multiplication (Strassen's Algorithm)

- Another Divide and Conquer Algorithm
- Matrix Multiplication: If A =(aij) and B = (bij) are square nxn matrices, then in the product C= A *B, we define the entry c(ij) , for i,j =1,2,…n:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Basic Matrix Multiplication


```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

algorithm

Time analysis

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c = cn^3 = O(n^3)$$

Basic Divide and Conquer Matrix Multiplication

Suppose we want to multiply two matrices of size $n$xn: for example $A * B = C$.

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$$\mathbf{C_{11} = A_{11}B_{11} + A_{12}B_{21}}$$

$$\mathbf{C_{12} = A_{11}B_{12} + B_{12}B_{22}}$$

$$\mathbf{C_{21} = A_{21}B_{11} + A_{22}B_{21}}$$

$$\mathbf{C_{22} = A_{21}B_{12} + A_{22}B_{22}}$$

2x2 matrix multiplication can be accomplished in 8 multiplication.$\mathbf{(2^{\log_2 8} = 2^3)}$

Recurrence for the running time of the basic D&C algorithm

- Why?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$$

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed in general as follows:

$$\left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right) = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) * \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right)$$

$$= \begin{pmatrix} P_5 + P_4 - P_5 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}$$

Formulas for Strassen's Algorithm

$P_1 = A_{11} * (B_{12} - B_{22})$

$P_2 = (A_{11} + A_{12}) * B_{22}$

$P_3 = (A_{21} + A_{22}) * B_{11}$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) * (B_{11} + B_{12})$

How much time for computing each parenthesis (10 total)?

7 multiplications

18 additions

Analysis of Strassen's Algorithm

If n is not a power of 2, matrices can be padded with zeros.

What if we count both multiplications and additions?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1\,. \end{cases}$$

Solution: T(n) = $n^{\log_2 7} \approx n^{2.807}$ vs. n^3 of brute-force and basic D&C alg.

(see next how to find running time easy)

Algorithms with better asymptotic efficiency are known but they are even more complex and not used in practice.

Example:

- Use Strassen's algorithm to compute the matrix product
- $\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} * \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$
- Show your work.