

**COMMONWEALTH OF AUSTRALIA**  
**Copyright Regulation 1969**

**WARNING**

This material has been copied and communicated to you by or on behalf  
of **Curtin University of Technology** pursuant to Part VB of the  
*Copyright Act 1968* (**the Act**)

The material in this communication may be subject to copyright under the  
Act. Any further copying or communication of this material by you  
may be the subject of copyright protection under the Act.

Do not remove this notice

# Design and Analysis of Algorithms

## Lecture 3

### Divide and Conquer

# Divide and Conquer

- Mergesort
  - Algorithm description
  - Analysis
- Quicksort
  - Algorithm description
  - Analysis
- Matrix Multiplication
  - Algorithm description
  - Analysis

# Divide & Conquer

- Basic steps:
  - Divide the problem into parts
  - Conquer (solve) each part
  - Combine (merge) the parts to get the result
- There are two ways to do this:
  - Sort when merging (Mergesort)
  - Sort when dividing (Quicksort)

# Divide & Conquer on Selection Sort

Recall Selection Sort on  $A[1..n]$

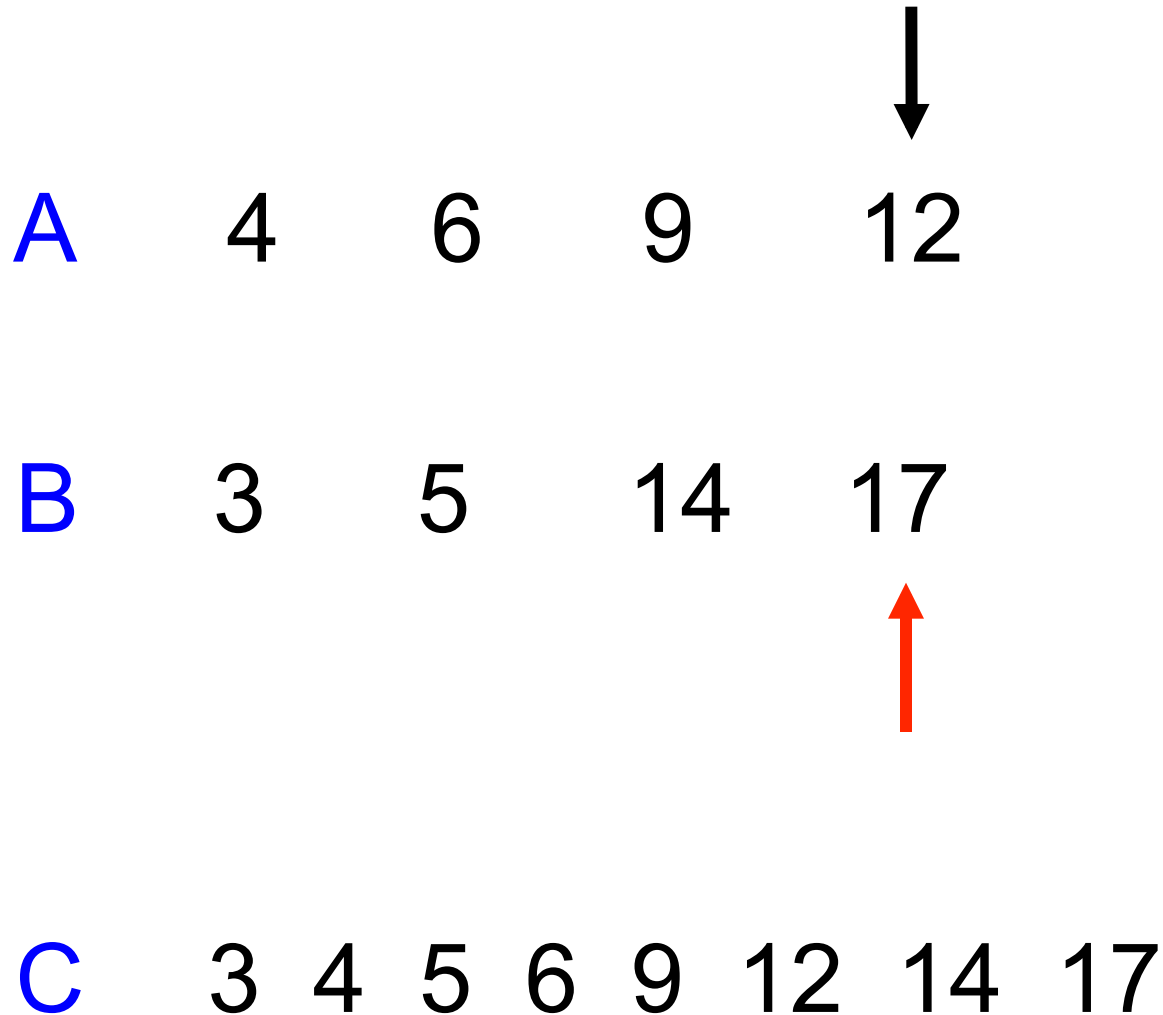
1. **for**  $i = 1$  **to**  $n-1$
2. Find the smallest element in  $A[i+1..n]$
3. Swap smallest with  $A[i]$

**Idea** - what if we split  $A$  into two halves, sort each half, and then combine them?

# Combine or Merge Data

- How do we combine two sorted arrays of  $n/2$  elements each to get a sorted array of  $n$ ?
  - How long (time) does it take?
  - How much space does it use?
- Standard merge algorithm
  - $\Theta(n)$  time
  - $\Theta(n)$  space
    - need another array to put in sorted data so not *in-place* sorting

# Merging two sorted arrays



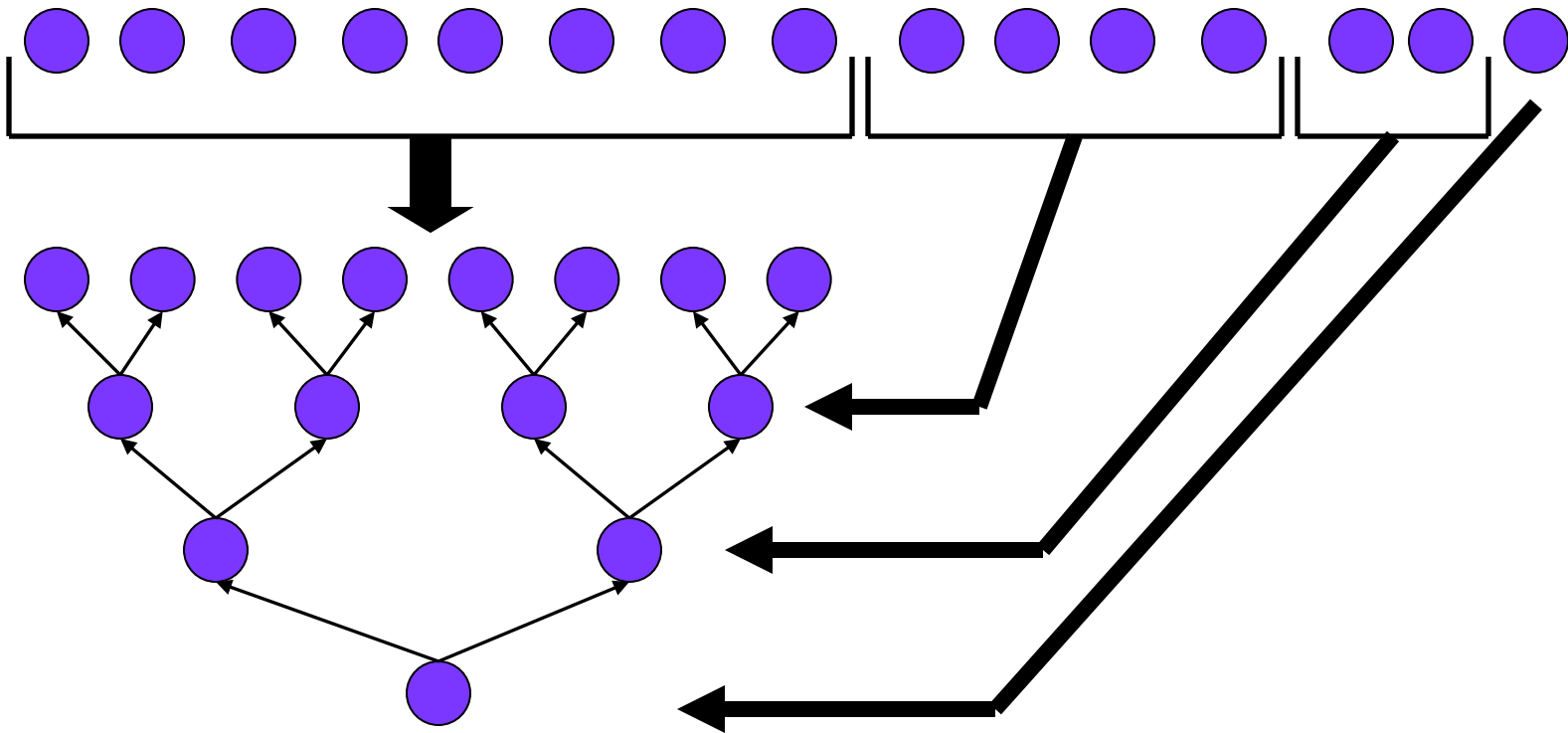
# Now the conquer

- Merge takes  $\Theta(n)$  time and  $\Theta(n)$  extra space
- How much time to Selection Sort each half?
  - $\Theta((n/2)^2) = \Theta(n^2) \rightarrow$  no improvement
- What if we continue to divide?

$$\Theta((n/2)^2) \rightarrow \Theta((n/4)^2) \rightarrow \Theta((n/8)^2) \dots \rightarrow \Theta(1)$$



# Repeated Division



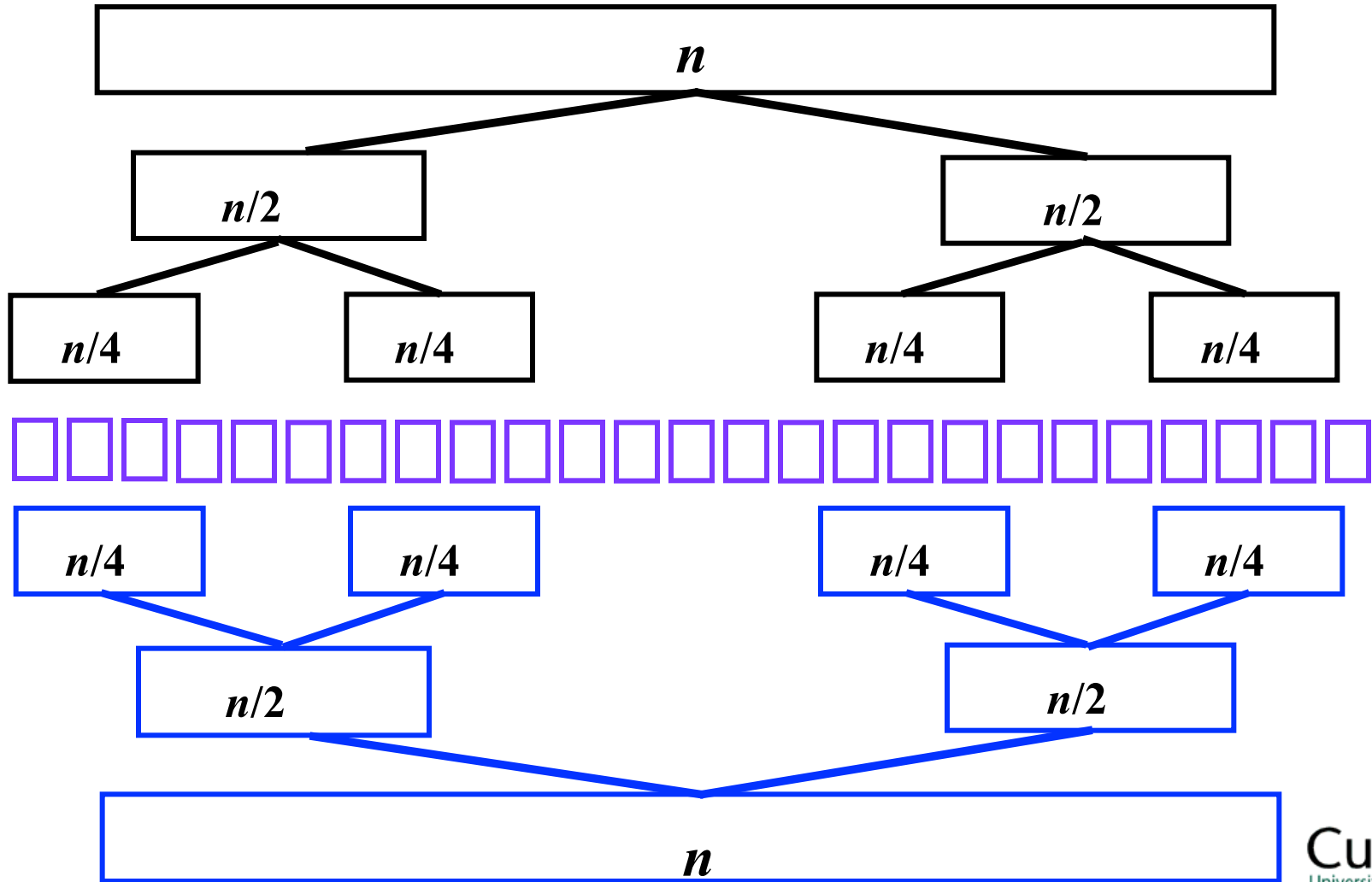
How many times can you halve  $n$  things?

$\log_2 n$  times

# So...

- Divide  $\log_2 n$  times
- Conquer each part in  $\Theta(1)$  time
- Merge parts back together in  $\Theta(n)$  time
- $\Theta(n \log n)$  time and  $\Theta(n)$  space
- This is **Mergesort**

# Mergesort



# Mergesort- example

13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
<u>13</u>	<u>02</u>	18	26	76	87	98	11	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	76	87	98	11	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	76	87	98	11	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	<u>98</u>	<u>11</u>	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	98	11	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	<u>98</u>	<u>11</u>	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	<u>11</u>	<u>98</u>	93	77	65	43	38	09	65	06
<u>02</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>11</u>	<u>76</u>	<u>87</u>	<u>98</u>	93	77	65	43	38	09	65	06
<u>02</u>	<u>11</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	<u>98</u>	<u>93</u>	<u>77</u>	<u>65</u>	<u>43</u>	<u>38</u>	<u>09</u>	<u>65</u>	<u>06</u>
<u>02</u>	<u>11</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>76</u>	<u>87</u>	<u>98</u>	<u>06</u>	<u>09</u>	<u>38</u>	<u>43</u>	<u>65</u>	<u>65</u>	<u>77</u>	<u>99</u>
<u>02</u>	<u>06</u>	<u>09</u>	<u>11</u>	<u>13</u>	<u>18</u>	<u>26</u>	<u>38</u>	<u>43</u>	<u>65</u>	<u>65</u>	<u>76</u>	<u>77</u>	<u>87</u>	<u>93</u>	<u>98</u>

# Mergesort Algorithm

**MERGESORT**( $A, l, r$ )      // Call with  $l = 1, r = n$  initially

**Input:** An array  $A$  in the range  $l$  to  $r$

**Output:** Sorted array  $A$

**if**  $l < r$       // when  $l = r$  we're done

**then**  $m = \lfloor (l+r)/2 \rfloor$

**MERGESORT**( $A, l, m$ )      // left sub-array

**MERGESORT**( $A, m+1, r$ )      // right sub-array

**MERGE** ( $A, l, m, r$ )

# Merge Algorithm

## **MERGE( $A, l, m, r$ )**

**Inputs:** Two sorted sub-arrays  $A[l .. m]$  and  $A[m+1 .. r]$

**Output:** Merged and sorted array  $A[l .. r]$

$i = 1$

$j = m+1$

$k = 1$

**while** ( $i \leq m$ ) and ( $j \leq r$ ) **do** // check if not at end of each sub-array

**if**  $A[i] \leq A[j]$  **then** // check for smaller element

$TEMP[k++] = A[i++]$

**else** // copy smaller element

$TEMP[k++] = A[j++]$  // into TEMP array

**while** ( $i \leq m$ ) **do**

$TEMP[k++] = A[i++]$  // copy all other elements

**while** ( $j \leq r$ ) **do** // to TEMP array

$TEMP[k++] = A[j++]$

Copy  $TEMP[l .. r]$  to  $A[l .. r]$  //  $A[l .. r]$  is in sorted order

See also MERGE on page 31 (text 3<sup>rd</sup> ed.)

# Merge Algorithm (from Cormen, et al.)

**MERGE**( $A, l, m, r$ )

$n_1 = m - l + 1$

$n_2 = r - m$

let  $L[1 .. n_1 + 1]$  and  $R[1 .. n_2 + 1]$  be new arrays

**for**  $i=1$  to  $n_1$  **do** // copy the first half contents of array  $A$  to array  $L$

$L[i] = A[l + i - 1]$  **then**

**for**  $j=1$  to  $n_2$  **do**

$R[j] = A[m + j]$  // copy the second half contents of array  $A$  to array  $R$

$L[n_1 + 1] = \infty$  // use a sentinel to indicate the end of elements

$R[n_2 + 1] = \infty$

$i=1$

$j=1$

**for**  $k = l$  to  $r$  **do**

**if**  $L[i] \leq R[j]$  **then**

$A[k] = L[i]$

$i = i + 1$

**else**

$A[k] = R[j]$

$j = j + 1$

The idea is the same but using sentinels and copying the elements to be merged in two new subarrays first

# Mergesort Analysis

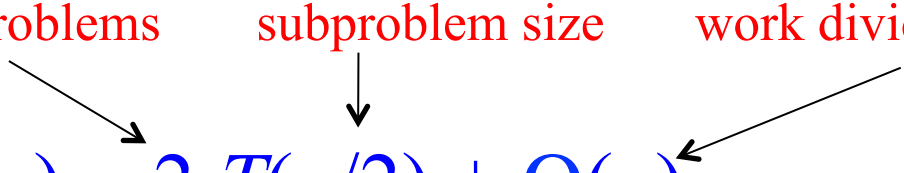
**Divide:** This step computes the middle of the array, takes constant time,  $\Theta(1)$

**Conquer :** Two sub-problems, each of size  $n/2$  are recursively solved.  
Each sub-problem contributes  $T(n/2)$  to the running time.

**Combine:** Two sorted sequences are merged, this takes  $\Theta(n)$  time (since go through all elements)

# subproblems      subproblem size      work dividing and merging

→  $T(n) = 2 T(n/2) + \Theta(n)$





# Analysis (Cont.)

## Using *Iteration Method*:

$$T(n) = 2 T(n/2) + \Theta(n) \text{ // overall}$$

$$T(n/2) = 2 T(n/4) + n/2 \text{ // initial iteration}$$

$$\rightarrow T(n) = 2 \{2 T(n/4) + n/2\} + n = 4 T(n/4) + 2 n$$

$$T(n/4) = 2 T(n/8) + n/4 \text{ // next iteration}$$

$$\begin{aligned} \rightarrow T(n) &= 4 \{2 T(n/8) + n/4\} + 2n = 8 T(n/8) + 3 n \\ &= 2^3 T(n/2^3) + 3n \end{aligned}$$

**So in general:**  $T(n) = 2^k T(n/2^k) + k n$ , where  $k = \text{iteration \#}$

If  $n = 2^k$  then  $k = \log_2 n$

$$\begin{aligned} \therefore T(n) &= 2^k T(1) + n \log_2 n \quad \text{// } T(1) \text{ is base case: 1 element array} = \Theta(1) \\ &= n \Theta(1) + n \log_2 n = \mathbf{O(n \log_2 n)} \end{aligned}$$

# Analysis (Cont.)

$T(n) = 2 T(n/2) + \Theta(n)$  is in the form  $aT(n/b) + f(n)$

- We can use **Master Theorem** where:

$$a = 2, b = 2, f(n) = n \rightarrow n^{\log_b a} = n^{\log_2 2} = n$$

- We have **case 2**

$$\therefore T(n) = \Theta(n \log_2 n)$$

# Quicksort

- Mergesort
  - All the work is done in *combine (merge)*
  - *Divide* is not clever
- Quicksort
  - All the work is in *divide (partitioning)*
  - *Combine* doesn't do any real work

# Quicksort (cont.)

**Divide:** partition the array  $A[l..r]$  into two non-empty sub-arrays  $A[l..m-1]$  and  $A[m+1..r]$  such that each element of  $A[l..m-1]$  is less than or equal to  $A[m]$  and each element of  $A[m+1..r]$  is larger than  $A[m]$

**Conquer:** sort the two sub-arrays  $A[l..m-1]$  and  $A[m+1..r]$  by recursive calls to Quicksort

**Combine:** the sub-arrays are already sorted with respect to each other, so no work is needed to combine them

Notice that  $A[m]$  is already at the right position for each recursion

# Quicksort Algorithm

**Quicksort**( $A, l, r$ )

**Input:** Unsorted Array ( $A, l, r$ )

**Output:** Sorted subarray  $A(1 \dots r)$

**if**  $l < r$

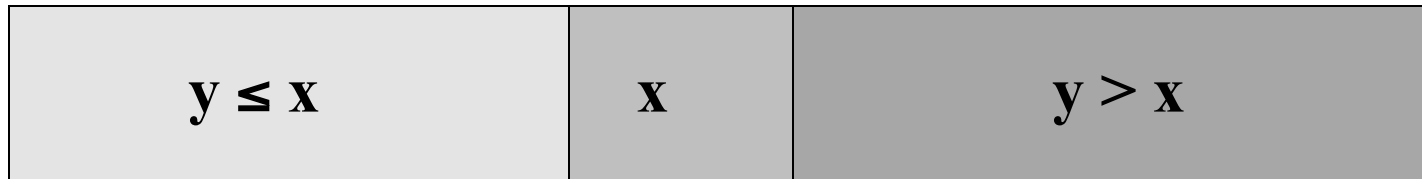
**then**  $m \leftarrow \text{PARTITION}(A, l, r)$

**QUICKSORT**( $A, l, m-1$ )

**QUICKSORT**( $A, m+1, r$ )

To sort the entire array  $A$ , set  $l = 1$  and  $r = \text{length}[A]$

$x$  is the **pivot** // after **PARTITION** will be at  $A[m]$



# Partition Algorithm

**PARTITION**( $A, l, r$ )

**Input:** Array  $A(l .. r)$

**Output:**  $A$  and  $m$  such that  $A[i] \leq A[m]$  for all  $i < m$  and  
 $A[j] > A[m]$  for all  $j > m$

$x = A[l]; i = l; j = r;$      // use the **first element** in  $A$  as **pivot**

**while**  $i < j$  **do**

    // search from both ends to find elements on both sides and swap them

**while**  $A[i] \leq x$  and  $i < j$  **do**  $i = i + 1$      // find  $A[i] > \text{pivot}$

**while**  $A[j] > x$  and  $j > i$  **do**  $j = j - 1$      // find  $A[j] \leq \text{pivot}$

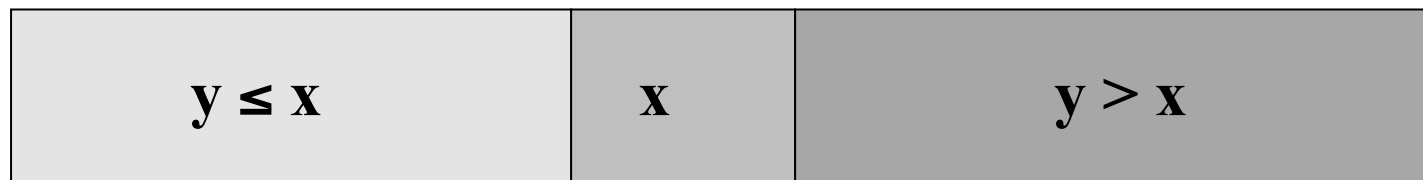
**if**  $i < j$  **then**

            exchange  $A[i] \leftrightarrow A[j]$

$m = j$

exchange  $A[l] \leftrightarrow A[m]$

return  $m$



# Quicksort - Example

<u>13</u>	02	<u>18</u>	26	76	87	98	11	93	77	65	43	38	09	65	<u>06</u>
13	02	<u>06</u>	<u>26</u>	76	87	98	11	93	77	65	43	38	<u>09</u>	65	<u>18</u>
13	02	06	<u>09</u>	<u>76</u>	87	98	<u>11</u>	93	77	65	43	38	<u>26</u>	65	18
<u>13</u>	02	06	09	<u>11</u>	87	98	<u>76</u>	93	77	65	43	38	26	65	18
<u>11</u>	02	06	09	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
<u>11</u>	02	06	<u>09</u>	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
<u>09</u>	02	06	<u>11</u>	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
<u>09</u>	02	<u>06</u>	<u>11</u>	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
<u>06</u>	02	<u>09</u>	<u>11</u>	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
<u>06</u>	<u>02</u>	<u>09</u>	<u>11</u>	<u>13</u>	87	98	76	93	77	65	43	38	26	65	18
02	<u>06</u>	<u>09</u>	<u>11</u>	<u>13</u>	<u>87</u>	<u>98</u>	76	93	77	65	43	38	26	65	<u>18</u>
02	06	09	11	13	87	<u>18</u>	76	<u>93</u>	77	65	43	38	26	<u>65</u>	<u>98</u>
02	06	09	11	13	<u>87</u>	18	76	<u>65</u>	77	65	43	38	<u>26</u>	<u>93</u>	98
02	06	09	11	13	<u>26</u>	<u>18</u>	76	65	77	65	43	38	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	76	65	<u>77</u>	65	43	<u>38</u>	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	<u>76</u>	65	<u>38</u>	65	<u>43</u>	<u>77</u>	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	<u>43</u>	<u>65</u>	<u>38</u>	65	<u>76</u>	<u>77</u>	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	<u>43</u>	<u>38</u>	<u>65</u>	65	<u>76</u>	<u>77</u>	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	<u>38</u>	<u>43</u>	<u>65</u>	<u>65</u>	<u>76</u>	<u>77</u>	<u>87</u>	93	98
02	06	09	11	13	18	<u>26</u>	<u>38</u>	<u>43</u>	<u>65</u>	<u>65</u>	<u>76</u>	<u>77</u>	<u>87</u>	<u>93</u>	<u>98</u>

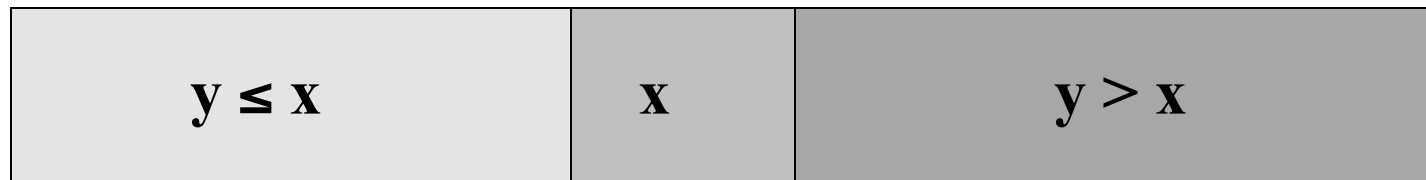
# PARTITION (from Textbook)

**PARTITION**( $A, l, r$ )

**Input:** Array  $A(l \dots r)$

**Output:**  $A$  and  $m$  such that  $A[i] \leq A[m]$  for all  $i \leq m$  and  
 $A[j] > A[m]$  for all  $j > m$

```
 $x = A[r]$            // use right-most element as pivot, store value in  $x$   
 $i = l$   
for  $j = l$  to  $r - 1$  do // kind of like selection sort but only selecting for  $\leq x$   
    if  $A[j] \leq x$  then  
        exchange  $A[i] \leftrightarrow A[j]$   
         $i = i + 1$   
  
exchange  $A[i] \leftrightarrow A[r]$  // put pivot element in the pivot's position  
return  $i$ 
```





# Example (following Textbook's algorithm)

13	<u>02</u>	18	26	76	87	98	11	93	77	65	43	38	09	65	<u>06</u>
<b>02</b>	13	18	26	76	87	98	11	93	77	65	43	38	09	65	<u>06</u>
<b>02</b>	<u>06</u>	18	26	76	87	98	11	93	77	65	43	38	09	65	<u>13</u>

18	26	76	87	98	<u>11</u>	93	77	65	43	38	09	65	<u>13</u>
<b>11</b>	26	76	87	98	18	93	77	65	43	38	<u>09</u>	65	<u>13</u>
<b>11</b>	<b>09</b>	76	87	98	18	93	77	65	43	38	26	65	<u>13</u>
<b>11</b>	<b>09</b>	<u>13</u>	87	98	18	93	77	65	43	38	26	65	76

87	98	<u>18</u>	93	77	65	43	38	26	65	<u>76</u>
<b>18</b>	98	87	93	77	65	43	38	26	65	<u>76</u>
<b>18</b>	<b>65</b>	87	93	77	98	43	38	26	65	<u>76</u>
<b>18</b>	<b>65</b>	<b>43</b>	93	77	98	87	38	26	65	<u>76</u>
<b>18</b>	<b>65</b>	<b>43</b>	<b>38</b>	77	98	87	93	26	65	<u>76</u>
<b>18</b>	<b>65</b>	<b>43</b>	<b>38</b>	<b>26</b>	98	87	93	77	65	<u>76</u>
<b>18</b>	<b>65</b>	<b>43</b>	<b>38</b>	<b>26</b>	<b>65</b>	87	93	77	98	<u>76</u>
<b>18</b>	<b>65</b>	<b>43</b>	<b>38</b>	<b>26</b>	<b>65</b>	<u>76</u>	93	77	98	87

93	77	98	<u>87</u>
<b>77</b>	93	98	<u>87</u>
<b>77</b>	<u>87</u>	98	93

98	<u>93</u>
<u>93</u>	98

# Quicksort – Time Complexity

$$T(n) = n-1 + T(m-1) + T(n-m)$$

- It takes  $n-1$  comparisons for the partition
- Then we sort smaller sequences of size  $m-1$  and  $n-m$  since  $l=1$  and  $r=n$  at the beginning
- Each element has the same probability of being selected as the pivot – so  $m$  is not known in advance
- So useful to consider *average* running time, by averaging over  $m$ :

$$\begin{aligned}T(n) &= n-1 + \frac{1}{n} \sum_{m=1}^n T(m-1) + \frac{1}{n} \sum_{m=1}^n T(n-m) \\&= n-1 + \frac{2}{n} \sum_{m=1}^n T(m-1) \\&= \dots \\&= O(n \log n)\end{aligned}$$

- Need more math calculations to prove the above

# Performance of Quicksort

Depends on whether the partitioning is

- Balanced  $\rightarrow \Theta(n \log n)$
- Unbalanced  $\rightarrow \Theta(n^2)$
- Balanced and unbalanced depend on which elements are used for partitioning

# Worst Case Partitioning

- Occurs if elements are already sorted  
partitioning produces:
  - 1 subproblem with  $n-1$  elements, and
  - 1 subproblem with 0 element
- It occurs in each recursive call:

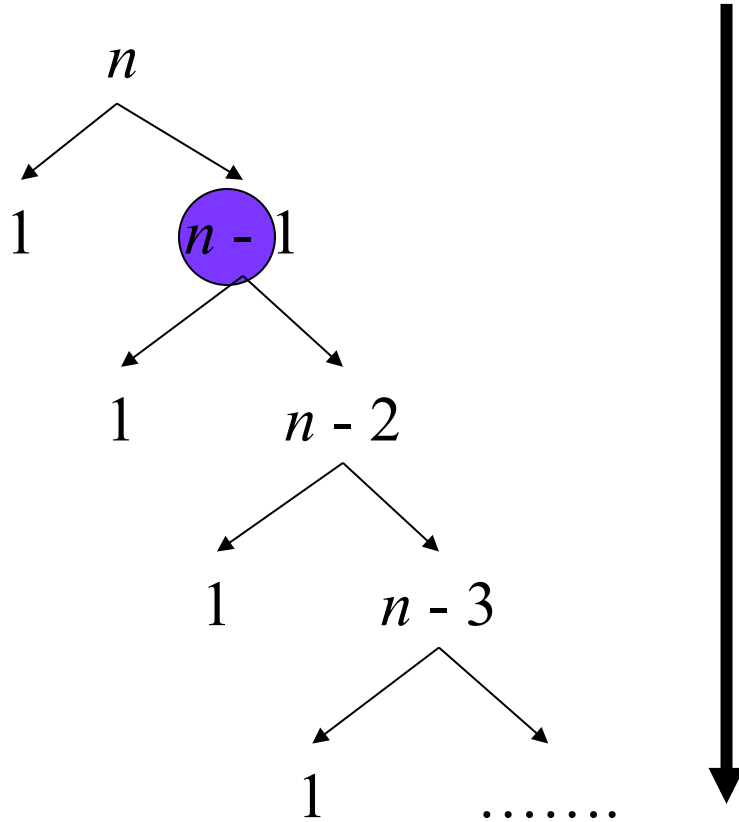
$$T(n) = T(n - 1) + \Theta(n)$$

$$T(n) = \Theta(n^2)$$

# Worst case (continued)

$$T(n) = n-1 + T(m-1) + T(n-m)$$

$m = 1$  for every level



## Recursion Tree:

- Depth of tree =  $\Theta(n)$
- Partition at each level =  $\Theta(n)$
- Worst case  $\rightarrow \Theta(n^2)$
- Can you prove it by induction?

# Best Case Partitioning

- Occurs when partitioning produces:
  - 1 subproblem with  $\lfloor n/2 \rfloor$  elements, and
  - 1 subproblem with  $\lceil n/2 \rceil - 1$  elements
- If this occurs in each recursive call:  
$$T(n) \leq 2 T(n/2) + \Theta(n) \rightarrow \text{Master Theorem}$$
$$\rightarrow T(n) = \Theta(n \log n)$$

# Balanced Partitioning

- Average case is closer to best case
- $T(n) = O(n \log n)$  if the split has **constant proportionality**

## Example:

$$9:1 \rightarrow T(n) \leq T(9n/10) + T(n/10) + cn$$

$$99:1 \rightarrow T(n) \leq T(99n/100) + T(n/100) + cn$$

$$\rightarrow T(n) = \Theta(n \log n)$$

**Proof?**

# Intuition for Average Case

- The behaviour of Quicksort depends on the relative ordering of the elements
- Assume that all permutations of elements are equally likely
- For random input, it is unlikely that partitioning always happens the same way for every level  $\rightarrow$  expect some balanced and some unbalanced
- Average case: PARTITION results in a mix of balanced and unbalanced splits
- When levels alternate between good and bad splits  $\rightarrow T(n) = O(n \log n)$
- See discussion in textbook (p. 151)



# Randomized Quicksort

- Considered good for large inputs
- Use random sampling
  - use a random element in  $A[l .. r]$  as a pivot.
- Exchange  $A[r]$  with the random sample. // for partition in the textbook
- Expected running time:  $T(n) = \Theta(n \log n)$

# Randomized Quicksort (cont)

**Randomized-PARTITION**( $A, l, r$ )

$i = \text{RANDOM}(l, r)$

exchange  $A[r] \leftrightarrow A[i]$  // the  $r^{\text{th}}$  element is the pivot.

**return** PARTITION( $A, l, r$ )

**Randomized-QUICKSORT**( $A, l, r$ )

**Input:** Unsorted Array ( $A, l, r$ )

**Output:** Sorted Array  $A(1 \dots r)$

**if**  $l < r$

**then**  $m = \text{Randomized-PARTITION}(A, l, r)$

Randomized-QUICKSORT ( $A, l, m-1$ )

Randomized-QUICKSORT ( $A, m+1, r$ )

To sort the entire array  $A$ ,  $l = 1$ ;  $r = \text{length}[A]$

# Matrix Multiplication

Consider two  $n \times n$  matrices,  $A = (a_{ij})$  and  $B = (b_{ij})$ , for  $1 \leq i, j \leq n$

**Problem:** multiply  $A$  and  $B$

i.e.,  $C = (c_{ij}) = A \times B$ , where  $c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj}$

**MATRIX\_MULTIPLY( $A, B$ )**

**Naïve multiplication takes time  $O(n^3)$**

1. **for**  $i \leftarrow 1$  to  $rows[A]$
2.     **for**  $j \leftarrow 1$  to  $columns[B]$
5.          $C[i, j] \leftarrow 0$ ;
6.         **for**  $k \leftarrow 1$  to  $columns[A]$
7.              $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$ ;
8. **return**  $C$

# Divide and Conquer

Partition each matrix  $A$ ,  $B$  and  $C$  into four  $n/2 \times n/2$  matrices as follows.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}; \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}; \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Thus,  $C = A \times B$  becomes

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

which means

$$C_{11} = A_{11} \times B_{11} + A_{11} \times B_{21}$$

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

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

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

Each equation requires two multiplications of  $n/2 \times n/2$  matrices, and the addition of their  $n/2 \times n/2$  products  $\rightarrow$  in total eight multiplications and four additions

# Divide and Conquer (cont. )

MATRIX\_MULT\_REC ( $A, B$ )

```
1   $n = \text{rows}[A]$ 
2  Let  $C$  be a new  $n \times n$  matrix
3  if  $n = 1$ 
4       $c_{11} = a_{11} \times b_{11}$ 
5  else partition  $A, B$ , and  $C$  each into four  $n/2 \times n/2$  matrices
6       $C_{11} = \text{MATRIX\_MULT\_REC}(A_{11}, B_{11}) + \text{MATRIX\_MULT\_REC}(A_{11} \times B_{21})$ 
7       $C_{12} = \text{MATRIX\_MULT\_REC}(A_{11} \times B_{12}) + \text{MATRIX\_MULT\_REC}(A_{12} \times B_{22})$ 
8       $C_{21} = \text{MATRIX\_MULT\_REC}(A_{21} \times B_{11}) + \text{MATRIX\_MULT\_REC}(A_{22} \times B_{21})$ 
9       $C_{22} = \text{MATRIX\_MULT\_REC}(A_{21} \times B_{12}) + \text{MATRIX\_MULT\_REC}(A_{22} \times B_{22})$ 
10 return  $C$ 
```

**Note:** The partition in Line 5 does not copy any matrix entries; it only uses index calculations (rows and columns) that requires  $O(1)$

# Divide and Conquer (cont. )

## Analysis:

- There are two multiplications of  $n/2 \times n/2$  matrices and one addition of their  $n/2 \times n/2$  product for each equation
  - Total: eight  $n/2 \times n/2$  matrix multiplications and four additions
  - Each  $n/2 \times n/2$  matrix addition requires  $\Theta(n^2)$  scalar additions
    - » There are  $n^2/4$  elements to be added
  - $T(n) = 8 T(n/2) + \Theta(n^2)$ 
    - » By Master theorem  $T(n) = \Theta(n^3)$
    - » This approach does not improve the naïve matrix multiplications

**Can we do better?**

# Strassen's algorithm for Matrix Multiplication

- In 1969, Strassen shows that it is possible to reduce the **eight** matrix multiplications into only **seven**
  - Trade off one matrix multiplication with a constant number of matrix additions / subtractions
  - Thus,  $T(n) = 7 T(n/2) + \Theta(n^2) \rightarrow T(n) = \Theta(n^{\lg 7}) = \Theta(n^{2.81})$

## Four steps of Strassen's algorithm:

- 1) Divide input matrices  $A$  and  $B$  and output matrix  $C$  each into  $(n/2 \times n/2)$  sub-matrices  $\rightarrow \Theta(1)$  using index calculation as before
- 2) Create ten matrices,  $S_i$  for  $i = 1, 2, \dots, 10$ , from the  $(n/2 \times n/2)$  sub-matrices of  $A$  and  $B$ , using only addition and subtraction operations on them.  $\rightarrow$  there are ten additions / subtractions with a total time of  $\Theta(n^2)$

$$S_1 = B_{12} - B_{22}$$

$$S_6 = B_{11} + B_{22}$$

$$S_2 = A_{11} + A_{12}$$

$$S_7 = A_{12} - A_{22}$$

$$S_3 = A_{21} + A_{22}$$

$$S_8 = B_{21} + B_{22}$$

$$S_4 = B_{21} - B_{11}$$

$$S_9 = A_{11} - A_{21}$$

$$S_5 = A_{11} + A_{22}$$

$$S_{10} = B_{11} + B_{12}$$

Eq. (1)

# Strassen's (cont. )

- 3) Recursively create seven matrix products,  $P_i$  for  $i = 1, 2, \dots, 7$ , from the eight sub-matrices of  $A$  and  $B$  in Step 1) and the ten  $S_i$  matrices created in Step 2); thus each  $P_i$  is a  $(n/2 \times n/2)$  sub-matrix

// Eq. (2)

$$\begin{aligned} P_1 &= A_{11} \times S_1 & // A_{11} \times (B_{12} - B_{22}) &= A_{11} \times B_{12} - A_{11} \times B_{22} \\ P_2 &= S_2 \times B_{22} & // (A_{11} + A_{12}) \times B_{22} &= A_{11} \times B_{22} + A_{12} \times B_{22} \\ P_3 &= S_3 \times B_{11} & // (A_{21} + A_{22}) \times B_{11} &= A_{21} \times B_{11} + A_{22} \times B_{11} \\ P_4 &= A_{22} \times S_4 & // A_{22} \times (B_{21} - B_{11}) &= A_{22} \times B_{21} - A_{22} \times B_{11} \\ P_5 &= S_5 \times S_6 & // (A_{11} + A_{22}) \times (B_{11} + B_{22}) &= A_{11} \times B_{11} + A_{11} \times B_{22} + A_{22} \times B_{11} + A_{22} \times B_{22} \\ P_6 &= S_7 \times S_8 & // (A_{12} - A_{22}) \times (B_{21} + B_{22}) &= A_{12} \times B_{21} + A_{12} \times B_{22} - A_{22} \times B_{21} - A_{22} \times B_{22} \\ P_7 &= S_9 \times S_{10} & // (A_{11} - A_{21}) \times (B_{11} + B_{12}) &= A_{11} \times B_{11} + A_{11} \times B_{12} - A_{21} \times B_{11} - A_{21} \times B_{12} \end{aligned}$$



# Strassen's (cont. )

- 4) Compute each sub-matrix of  $C$ , i.e.,  $C_{11}$ ,  $C_{12}$ ,  $C_{21}$ , and  $C_{22}$  by adding or subtracting various combinations of the seven  $P_i$  generated in Step 3) → there are eight additions / subtractions with total time of  $\Theta(n^2)$

// Eq. (3)

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$// \text{ Thus, } C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

$$C_{12} = P_1 + P_2$$

$$// \text{ Thus, } C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$

$$C_{21} = P_3 + P_4$$

$$// \text{ Thus, } C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$// \text{ Thus, } C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

## Example:

$$C_{12} = P_1 + P_2 = (A_{11} \times B_{12} - A_{11} \times B_{22}) + (A_{11} \times B_{22} + A_{12} \times B_{22}) = A_{11} \times B_{12} + A_{12} \times B_{22}$$

**Show that the computation of the other three sub-matrices of  $C$  from the seven  $P_i$ 's per Eq. (3) would produce the correct results!**

# Strassen (cont. )

Strassen ( $A, B$ )

```
1   $n = \text{rows}[A]$ 
2  Let  $C$  be a new  $n \times n$  matrix
3  if  $n = 1$ 
4       $c_{11} = a_{11} \times b_{11}$  // Directly compute  $A \times B$ 
5  else // Use Eq. (2)
6       $P_1 = \text{Strassen}(A_{11}, B_{12} - B_{22})$ 
7       $P_2 = \text{Strassen}(A_{11} + A_{12}, B_{22})$ 
8       $P_3 = \text{Strassen}(A_{21} + A_{22}, B_{11})$ 
9       $P_4 = \text{Strassen}(A_{22}, B_{21} - B_{11})$ 
10      $P_5 = \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22})$ 
11      $P_6 = \text{Strassen}(A_{12} - A_{22}, B_{21} + B_{22})$ 
12      $P_7 = \text{Strassen}(A_{11} - A_{21}, B_{11} + B_{21})$ 

13      $C_{11} = P_5 + P_4 - P_2 + P_6$  // Use Eq. (3)
14      $C_{12} = P_1 + P_2$ 
15      $C_{21} = P_3 + P_4$ 
16      $C_{22} = P_5 + P_1 - P_3 - P_7$ 
17 return  $C$ 
```

# Strassen's (cont. )

## Analysis

- Strassen's algorithm is a **divide and conquer** algorithm by computing the product of two  $n \times n$  matrices recursively using seven products and 18 additions / subtractions of  $n/2 \times n/2$  matrices!
- The 18 matrix additions / subtractions require  $\Theta(n^2)$
- Time complexity:  $T(n) = 7 T(n/2) + \Theta(n^2)$ 
  - Master Theorem:  $T(n) = \Theta(n^{2.808})$

## Can we do better?

- Winograd shows how to reduce the 18 additions / subtractions of  $n/2 \times n/2$  matrices into only 15! **HOW?**
  - What is the time complexity of Winograd's algorithm?
- Using similar idea, the current best algorithm is  $O(n^{2.32})$ 
  - It divides a matrix into a bigger number of sub-matrices
  - It is not practical since it has very large constant factor

# Strassen's – can we do better?

**Winograd:** The following includes 24 additions / subtractions, but requires only 15 **distinct** additions / subtractions

// Eight *distinct* additions / subtractions

$$P_1 = (A_{21} + A_{22} - A_{11}) \times (B_{22} - B_{12} + B_{11})$$

$$P_2 = A_{11} \times B_{11}$$

$$P_3 = A_{12} \times B_{21}$$

$$P_4 = (A_{11} - A_{21}) \times (B_{22} - B_{12})$$

$$P_5 = (A_{21} + A_{22}) \times (B_{12} - B_{11})$$

$$P_6 = (A_{12} + A_{11} - A_{21} - A_{22}) \times B_{22} \rightarrow (A_{12} - (A_{21} + A_{22} - A_{11})) \times B_{22}$$

$$P_7 = A_{22} \times (B_{22} - B_{12} + B_{11} - B_{21})$$

// Seven *distinct* additions / subtractions

$$C_{11} = P_2 + P_3 \quad // \text{ Thus, } C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

$$C_{12} = P_1 + P_2 + P_5 + P_6 \quad // \text{ Thus, } C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$

$$C_{21} = P_1 + P_2 + P_4 - P_7 \quad // \text{ Thus, } C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$

$$C_{22} = P_1 + P_2 + P_4 + P_5 \quad // \text{ Thus, } C_{21} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

# Is Strassen's algorithm practical?

- Although  $\Theta(n^{2.808})$  is asymptotically faster than  $\Theta(n^3)$  using brute-force, it has a larger constant factor
- For sparse matrices, better to use another algorithm suited for the matrices
  - Strassen's algorithm is practical for use in large dense matrices
  - When the matrix size is sufficiently small at some point in the recursion, use a simpler method, e.g., the brute-force.
- Strassen's algorithm produces larger errors when it is used for non-integer values
  - Researchers argue that the errors are acceptable for some applications
- Strassen's algorithm requires space to store submatrices constructed at recursion
  - Some researchers have proposed techniques to reduce space

# The End

- See you in the tutorials.