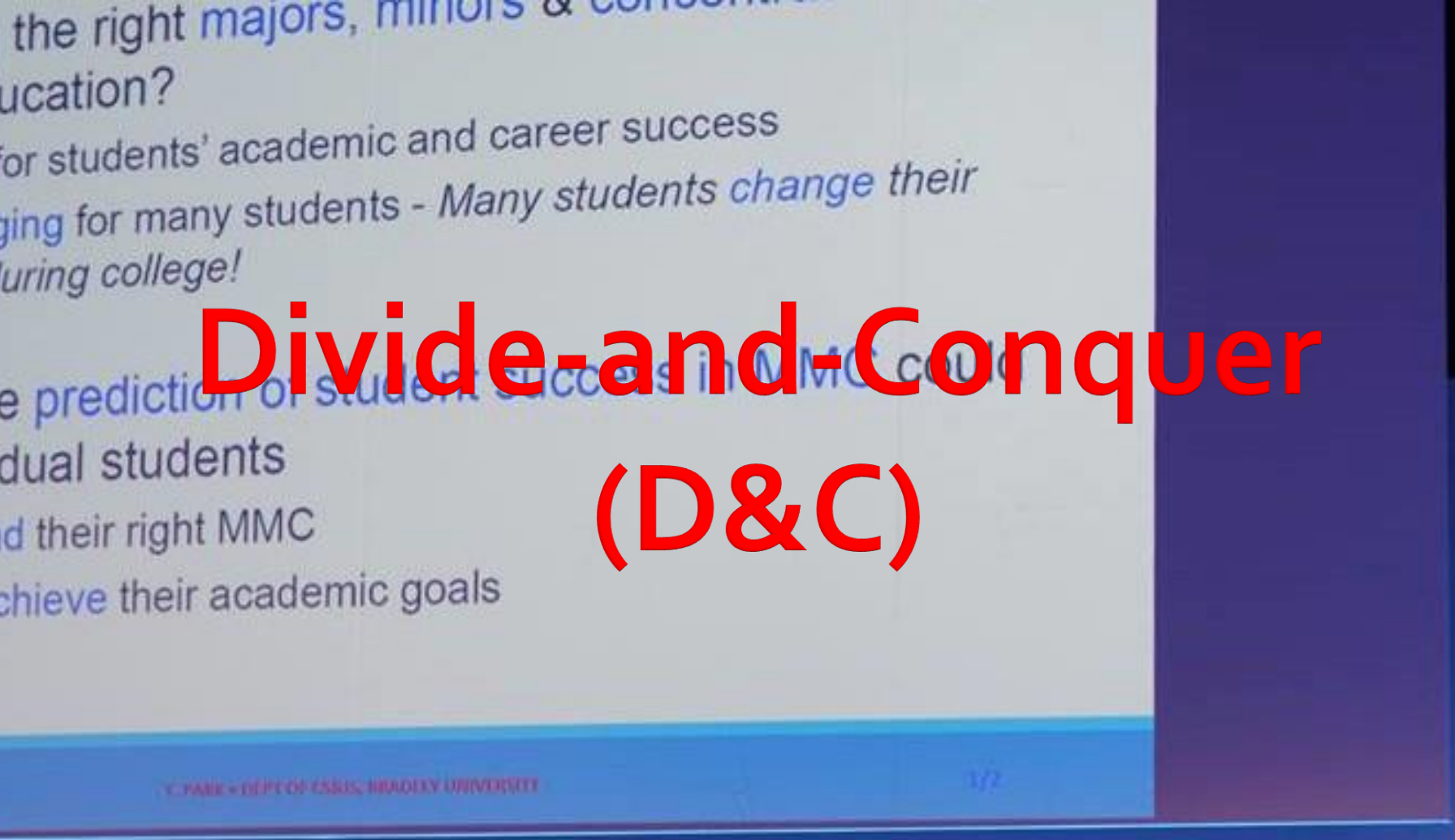
Divide-and-Conquer (D&C)



Prof. Young Park

✓ Divide-and-Conquer (D&C)

- An approach/paradigm to designing algorithms!

Divide-and-Conquer (D&C)

- This approach **divides** an instance of a problem into one or more smaller instances of the original problem.
- This process of dividing the instances continues **until** they are so small that a solution is readily obtainable.
- If solutions to the smaller instances can be obtained readily, the solution to the original instance can be obtained by **combining** these solutions.

Divide-and-Conquer

- The divide-and-conquer approach is a **top-down** approach.
- The solution of a top-level instance of a problem is obtained by going down and obtaining solutions to smaller instances.

Divide-and-Conquer

- **Divide (or Decrease)-and-Conquer (and Combine)**

✓ Divide-and-Conquer (D&C)-based Algorithms

Divide-and-Conquer (D&C)-based Algorithms

1. Binary Search – Searching via D&C
2. Merge Sort – Sorting via D&C
3. Quick Sort – Sorting via D&C
4. Matrix Multiplication via D&C

5. Quick Select & MMSelect – Selection via D&C

1. Binary Search – Searching via D&C

- The search problem on a **sorted** array!
- A **linear sequential search** algorithm
  - **O(n)** worst-case search time
- A better search algorithm?
  - Apply **D&C**!
  - **O(log n)** worst-case search time
  - Searching worst-case lower bound is **Ω(logn).**

Binary Search – Searching via D&C

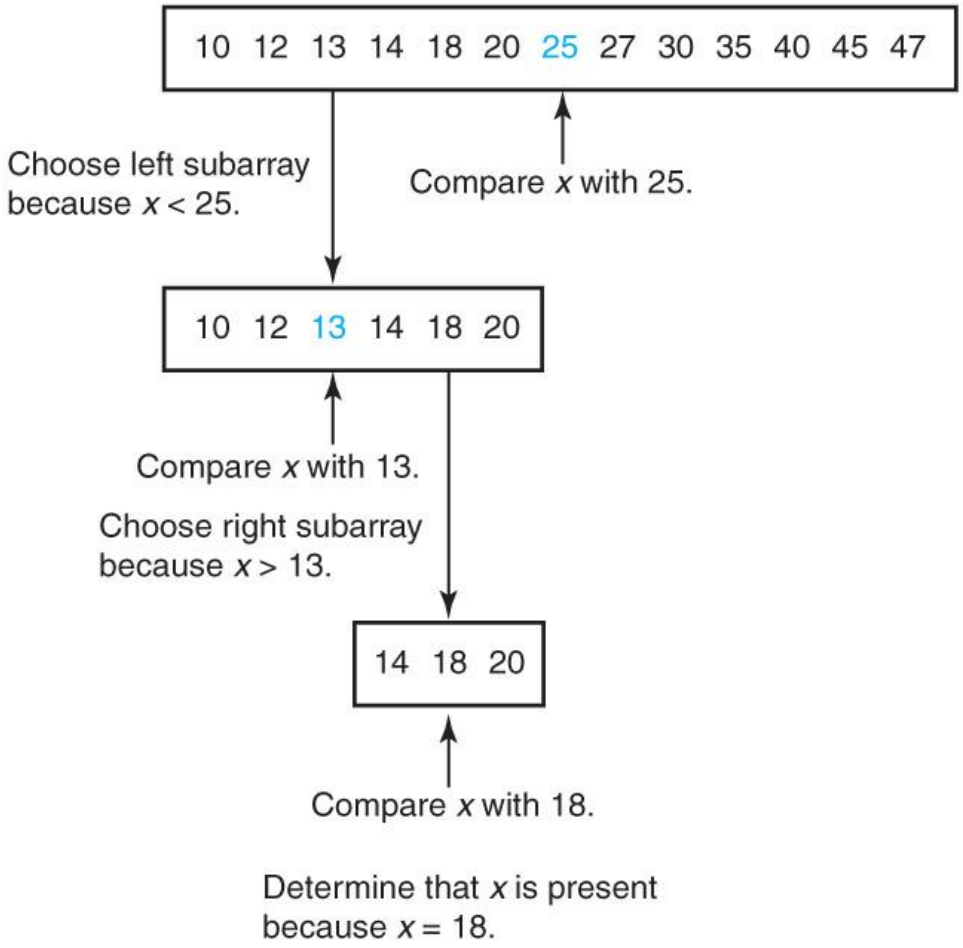
- A divide-and-conquer-based searching algorithm on a **sorted** array!

Binary Search – Searching via D&C

- Example:
  - BinarySearch for **18** in the array
    [10 12 13 14 18 20 25 27 30 35 40 45 47]

  - Example 2.1
  - Figure 2.1

Binary Search – Searching via D&C



Binary Search – Iterative via D&C

- **Algorithm 1.5** Binary Search (**Iterative**)
- Analysis of Algorithm 1.5
  - O(log n)

Binary Search - Recursive via D&C

- **Algorithm 2.1** Binary Search (**Recursive**)
- Example 2.1
- Figure 2.1

▶QUIZ? Binary Search

- BinarySearch for **18** in the array
  [10 12 13 14 18 20 25 27 30 35 40 45 47] **?**

Binary Search - Analysis

- Analysis of Algorithm 2.1 Binary Search (Recursive)

$$W(n) = 1 \quad \text{if } n=1$$
$$W(n) = W(n/2) + 1 \quad \text{if } n>1$$

Binary Search - Analysis

- **W(n) = W(n/2) + 1** if n>0
- **W(1) = 1**
  - **W(n)** = **W(n/2^1)** + 1
  - = **W(n/2^2) + 1** + 1
  - = **W(n/2^3) + 1** + 1 + 1
  - = **W(n/2^4) + 1** + 1 + 1 + 1
  - .
  - .
  - = **W(n/$2^{\log n}$) + 1** + 1 + ……. + 1
  - = 1 + log n x 1
  - = 1 + log n
  - = **O(log n)**
  - O(log n)
  - Example B.1 & B.18

▶QUIZ?

- **T(n) = T(n/2) + 1**
  - Using Master Theorem
  - By Substitution
  - O(log n)

Binary Search - Recursive via D&C

- The **simplest kind** of divide-and-conquer algorithm
  - The instance is broken down into only one smaller instance.
  - There is no combination of outputs.
  - The solution of the original instance is the solution to the smaller instance.

▶QUIZ?

- **Algorithm 2.1**
  - Binary Search (**Recursive**)
  - Recurrence Equation?

$$W(n) = 1 \quad \text{if } n=1$$
$$W(n) = W(n/2) + 1 \quad \text{if } n>1$$

  - **O(log N)**

▶QUIZ?

- How about **Ternary Search**?
  - Write a recursive algorithm that **searches** a sorted array of n items by dividing it into **three sorted subarrays of n/3 items**.
  - Recurrence Equation?
    - $W(n) = W(n/3) + 2$
    - $O(\log n)$
- Exercise 6

▶QUIZ?

- **Linear Search (Recursive)?**
  - Recurrence Equation?

$$W(n) = 1 \quad \text{if } n=1$$
$$W(n) = W(n-1) + 1 \quad \text{if } n>1$$

  - **O(N)**

2. Merge Sort – Sorting via D&C

- A simple sorting algorithm **ExchangeSort**
  - **$O(n^2)$** worst-case time

- Can we do better?
  - Apply **D&C**!
  - **O(n log n)** worst-case time

  - Sorting worst-case lower bound is **Ω(n logn)**.

Merge Sort – Sorting via D&C

- A divide-and-conquer-based sorting algorithm.

Idea:

Sort by Merging!

Mergesort

- Three steps.
  - Step 1: Divide the array into 2 sub-arrays each of n/2.
  - Step 2: Solve each sub-array by sorting it (use recursion till array is sufficiently small).
  - Step 3: Combine solutions to the sub-arrays by merging them into a single sorted array.

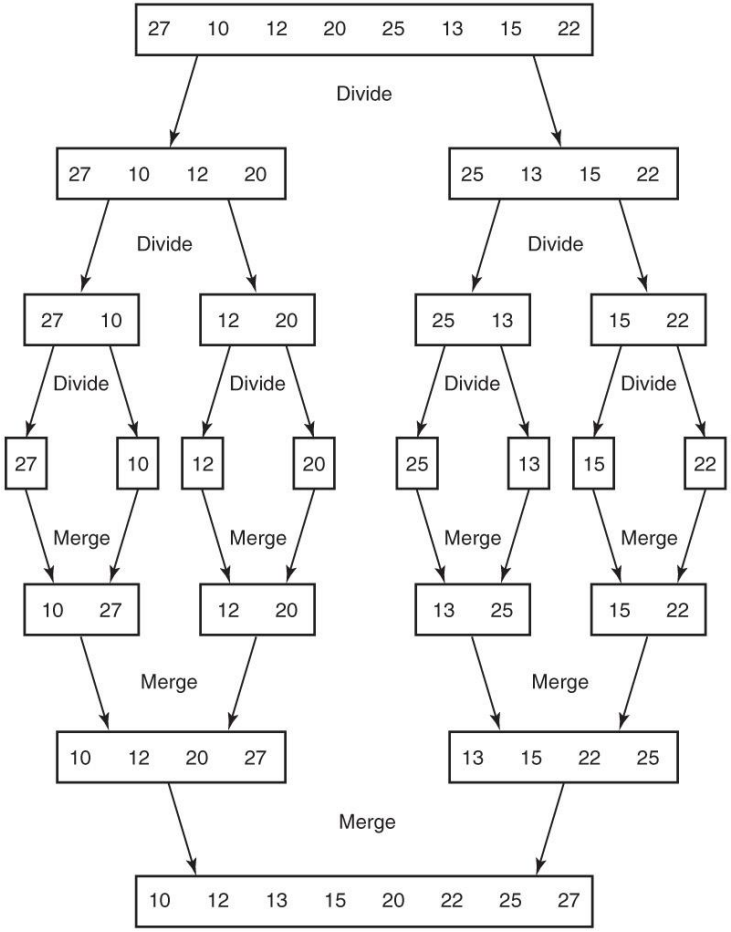
Merge

- Using **two-way merging**, we can combine **two sorted** arrays into **one** array.
- By repeatedly applying the merging procedure, we can sort an array.

MergeSort

- Example:
- MergeSort **[27 10 12 20 25 13 15 22]**
  - Example 2.2
  - Figure 2.2

MergeSort


27 10 12 20 25 13 15 22
Divide
27 10 12 20
25 13 15 22
Divide
Divide
27 10
12 20
25 13
15 22
Divide
Divide
Divide
Divide
27
10
12
20
25
13
15
22
Merge
Merge
Merge
Merge
10 27
12 20
13 25
15 22
Merge
Merge
10 12 20 27
13 15 22 25
Merge
10 12 13 15 20 22 25 27

Merging

- Merging is a combination of two or more sorted sequences into a single sorted sequence.

Merge

- Merge **[10 12 20 27]** and **[13 15 22 25]**!
  - Table 2.1

- Analysis of Algorithm 2.3 **Merge**
  - W(h,m) = h + m + -1
  - **O(n) time**
  - **O(n) auxiliary space**

Merge Sort - Recursive

- If there is more than one item in the array
  - Cut the array in half.
  - Merge Sort the left half (into an auxiliary array) recursively.
  - Merge Sort the right half (into an auxiliary array) recursively.
  - Merge the two sorted halves back into one sorted array in the original array.
- If there is only one item in the array
  - Do nothing.

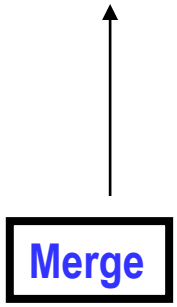
▶QUIZ? Mergesort

- **MergeSort [27 10 12 20 25 13 15 22]?**

Merge Sort - Analysis

- Analysis of Algorithm 2.2 Mergesort
  - Example B.19
- T(n) = The running time for Merge Sort

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + O(n) \quad \text{if } n>1$$



Merge Sort - Analysis

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + n \quad \text{if } n>1$$

$$\begin{aligned}
\mathbf{T(n)} &= 2\,T(n/2^1) + n \\
&= 2\,[2\,T(n/2^2) + n/2] + n \\
&= 2^2\,T(n/2^2) + n + n \\
&= 2^2\,[2\,T(n/2^3) + n/2^2] + n + n \\
&= 2^3\,T(n/2^3) + n + n + n \\
&\;\;\vdots \\
&= 2^{\log n}\,T(n/2^{\log n}) + n + \ldots + n + n + n \\
&= n\,T(1) + (\log n)\,n \\
&= n + n \log n \\
&= \mathbf{O(n \log n)}
\end{aligned}$$

▶QUIZ?

- **T(n) = 2 T(n/2) + n T(1)= Q(1)**
  - Using Master Theorem
  - By Substitution
  - O(n log n)

Merge Sort - Analysis

- **Algorithm 2.4 Merge Sort**
  - Worst-Case Time Complexity= O(nlogn)
  - Average-Case Time Complexity = O(nlogn)
  - Worst-Case Space Complexity= O(n)

- **Algorithm 2.5 Merge**
  - Worst-Case Time Complexity= O(n)
  - Worst-Case Space Complexity= O(n)

▶QUIZ?

- **Algorithm 2.4 Merge Sort**
  - Recurrence Equation?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + O(n) \quad \text{if } n>1$$

  - Worst-Case Time Complexity= O(n log n)
  - Average-Case Time Complexity = O(n log n)
  - Worst-Case **Space** Complexity= **O(n)**

▶QUIZ?

- How about **3-way MergeSort**?
  - Write an algorithm that **sorts** a list of n items by dividing it into **three sublists of about n/3 items**, sorting each sublist recursively and **merging** the three sorted sublists.
  - Recurrence Relation?
    - **W(n) = 3 W(n/3) + O(n)**
    - **O(n log n)**
- Exercise 13

▶QUIZ?

- How about **MergeSort** using **$O(n^2)$** time & **O(1)** auxiliary space **Merge**?

- Recurrence Relation?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + O(n^2) \quad \text{if } n>1$$

- Worst-Case Time Complexity?
  - **$O(n^2)$**

Merge Sort Algorithm via D&C Visualization

- ***Merge Sort Algorithm via D&C Visualization***

3. Quick Sort – Sorting via D&C

- A simple sorting algorithm **ExchangeSort**
  - **$O(n^2)$** worst-case & average-case time
- Can we do better at average-case?
  - Apply **D&C**!
  - **O(n log n)** average-case time
  - Sorting average-case lower bound is **Ω(n logn)**.

Quick Sort – Sorting via D&C

- A divide-and-conquer-based sorting algorithm.

Idea:
Sort by Partitioning!

Quick Sort (Partition Exchange Sort)

- Quicksort is similar to Mergesort
  - The sort is accomplished by dividing the array into two partitions and then sorting each partition recursively.
- In quicksort, the array is partitioned by placing all items smaller (larger) than some pivot item before (after).
  - The pivot item can be any item – **Random selection**!
  - **Randomized Algorithm!**

Good average-case time complexity!

Partition

- The array is partitioned by placing all items smaller (larger) than some pivot item before (after).
- After the partitioning,
  - Items smaller than the pivot item are to the left of the item, and all items larger than the pivot item are to the right of it.
  - The order of the items in the subarrays is unspecified and is a result of how the partitioning is implemented.

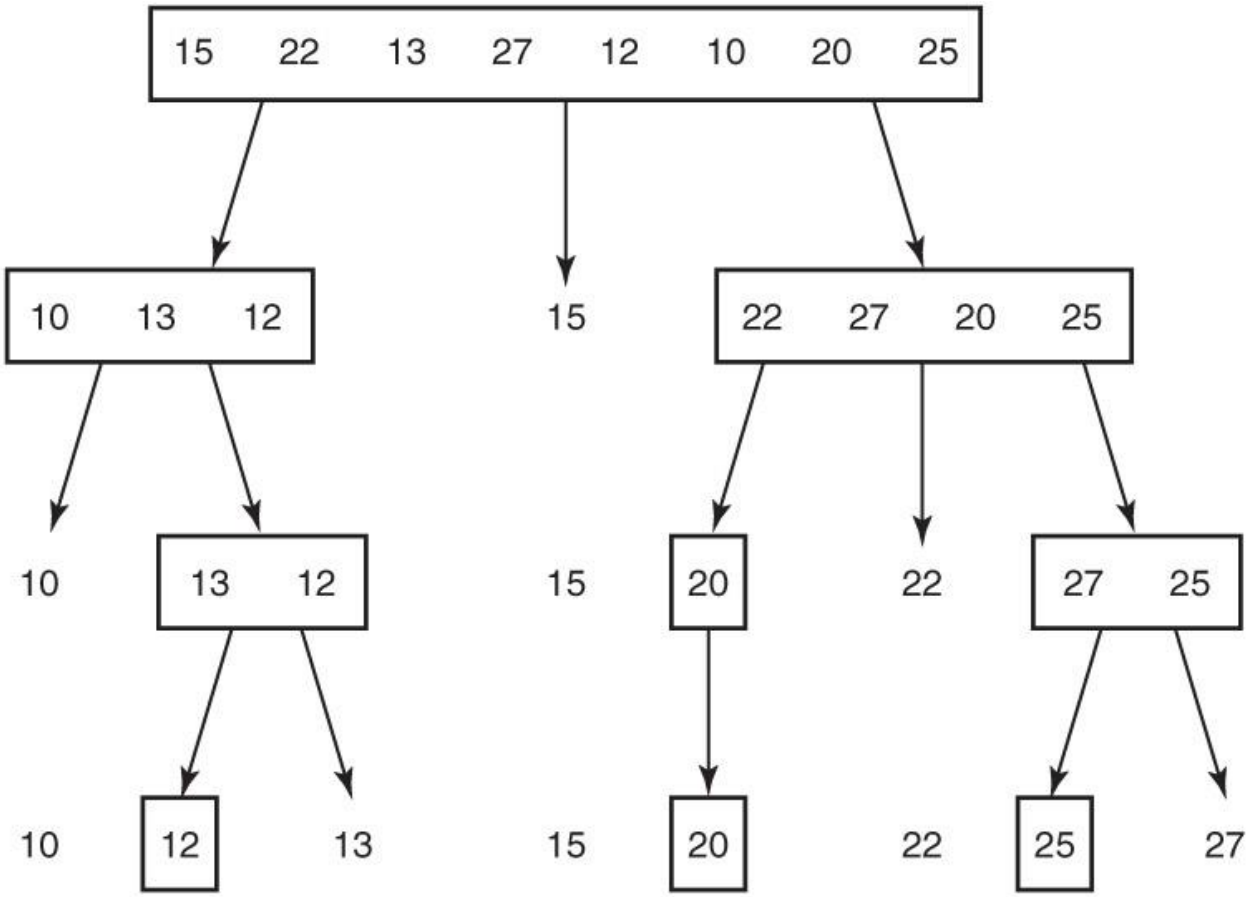
Quick Sort (Partition Exchange Sort)

- Quicksort is then called recursively to sort each of the two subarrays.
- They are partitioned, and the procedure is continued until an array with one item is reached.
  - This array with one item is trivially sorted.

Quick Sort

- Example:
- QuickSort **[15 22 13 27 12 10 20 25]**
  - Example 2.3
  - Figure 2.3

Quick Sort


15 22 13 27 12 10 20 25
10 13 12
15
22 27 20 25
10
13 12
15
20
22
27 25
10
12
13
15
20
22
25
27

Quick Sort - Recursive

- **Algorithm 2.6** Quicksort
  - If there is more than one item in the array
    - Select a splitting (pivot) value.
    - Split (partition) the array according to the splitting value into two parts
      - The left part of the array $\leq$ splitting value.
      - The right part of the array $>$ splitting value.
    - Quicksort the left part of the array recursively.
    - Quicksort the right part of the array recursively.
  - If there is only one item in the array
    - Do nothing.

Partitioning

- Splits (partitions) the array according to the splitting (pivot) value into two parts
  - The left part of the array $\leq$ The splitting (pivot) value.
  - The right part of the array $>$ The splitting (pivot) value.
- Places the pivot value in its correct position within the array.

Partition

- **Algorithm 2.7** Partition
- Partition [**15 22 13 27 12 10 20 25**]!
  - Table 2.2

Partition

- Analysis of Algorithm 2.7 **Partition**
  - T(n) = n-1
  - **O(n)** worst-case time

▶QUIZ? Quick Sort

- QuickSort [15 22 13 27 12 10 20 25]?

A Pivot Value

- The efficiency of Quick Sort depends on the choice of the pivot value!
  - Least efficient:
    - When the value chosen as the pivot is either the smallest or the largest value in the array.
  - Most efficient:
    - When the value chosen as the pivot partitions the array into two parts with about the same number of values.

How to Select the Pivot Value?

- The value in the first of the array.
- The value in the middle of the array.
- The value in the last of the array.
- The median of first, middle and last of the array.
- The randomly chosen value in the array.
- …

Good average-case time complexity!

Quick Sort - Analysis

- T(n) = The running time for Quick Sort when the pivot value is the p_th smallest:

p= 1,2, 3, …, n-1, n

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(p-1) + T(n-p) + O(n) \quad \text{if } n>1,\ 1 =< p =< n$$

Partition

Quick Sort - Analysis

- T(n) = The running time for Quick Sort when the pivot value is the p_th smallest:

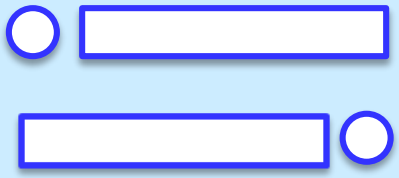
p= 1,2, 3, …, n-1, n

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(p-1) + T(n-p) + O(n) \quad \text{if } n>1,\ 1 =< p =< n$$

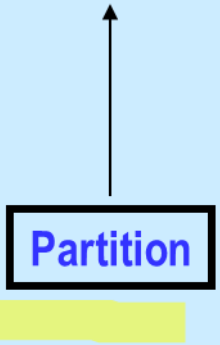
- **Worst-case?**
- **Best-case?**
- **Average-case?**

Worst Case: i = 0

- Worst-case?
  - **p=1**
  - **p=n**



$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(n-1) + O(n) \quad \text{if } n>1$$



Worst Case: i = o

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(n-1) + n \quad \text{if } n>1$$

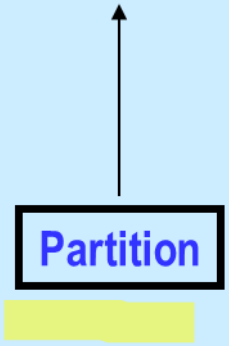
Example B.16

$$\begin{aligned}
\mathbf{T(n)} &= T(n-1) + n \\
&= T(n-2) + (n-1)+ n \\
&= T(n-3) + (n-2)+ (n-1)+ n \\
&= T(n-4) + (n-3) + (n-2)+ (n-1)+ n \\
&\ . \\
&\ . \\
&= T(n-(n-1)) + 2+3+ \ldots + (n-3) + (n-2)+ (n-1)+ n \\
&= T(1) + 2+3+ \ldots + (n-3) + (n-2)+ (n-1)+ n \\
&= \mathbf{O(n^2)}
\end{aligned}$$

Best Case: i = n/2

- Best Case?
  - p= n/2

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + O(n) \quad \text{if } n>1$$



Best Case: i = n/2

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2\,T(n/2) + n \quad \text{if } n>1$$

$$\begin{aligned}
T(n) &= 2\,T(n/2^1) + n \\
&= 2\,[2\,T(n/2^2) + n/2] + n \\
&= 2^2\,T(n/2^2) + n + n \\
&= 2^2\,[2\,T(n/2^3) + n/2^2] + n + n \\
&= 2^3\,T(n/2^3) + n + n + n \\
&\;\;\vdots \\
&= 2^{\log n}\,T(n/2^{\log n}) + n + \ldots + n + n + n \\
&= n\,T(1) + (\log n)\,n \\
&= n + n \log n \\
&= \mathbf{O(n \log n)}
\end{aligned}$$

▶QUIZ?

- **T(n) = 2 T(n/2) + n T(1)= Q(1)**
  - Using Master Theorem
  - By Substitution
  - O(n log n)

Average (Expected) Case: i =o .. n

- T(n) = The running time for Quick Sort when the pivot value is the p_th smallest:

p= 1,2, 3, …, n-1, n

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(p-1) + T(n-p) + O(n) \quad \text{if } n>1,\ 1 =< p =< n$$

Partition

Average (Expected) Case: i =o .. n

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = \sum_{p=1,n} [T(p-1) + T(n-p)]\ 1/n + O(n) \quad \text{if } n>1,\ 1 =< p =< n$$

$$T(n) = O(n \log n)$$

Good average-case time complexity!

Quick Sort - Analysis

- Analysis of Algorithm 2.6 Quicksort
  - Worst-Case Time Complexity= $O(n^2)$
  - Average (Expected) Case Time Complexity = O(nlogn)
  - Obtaining a good running time requires the choice of a good splitting (pivot) element!

▶QUIZ?

- **Algorithm 2.6 Quicksort**
- **Worst-Case?**
  - Recurrence Equation?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = T(n-1) + O(n) \quad \text{if } n>1$$

  - Worst-Case Time Complexity= $O(n^2)$

▶QUIZ?

- **Algorithm 2.6 Quicksort**
- **Best-Case?**
  - Recurrence Equation?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 2T(n/2) + O(n) \quad \text{if } n>1$$

  - Worst-Case Time Complexity= O(n log n)

▶QUIZ?

- **Algorithm 2.6 Quicksort**
- **Average (Expected) Case?**

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = \sum_{p=1,n} [T(p-1) + T(n-p)]\ 1/n + O(n) \quad \text{if } n>1,\ 1 \leq p \leq n$$

- Average (Expected) Case Time Complexity =

  O(n log n)

▶QUIZ? Selection via D&C!

- **Selection via D&C!**
- **Quick Select**?
- **O(n) average-case** time

The *Selection* Problem & Algorithms *(Chapter 8)*

▶QUIZ? Selection via D&C!

- **Selection via D&C!**
- **Median-of-Medians Select**?
- **O(n) worst-case** time

- **We can find the median of an unsorted array of n items in O(n) at worstocas!**

The *Selection* Problem & Algorithms *(Chapter 8)*

▶QUIZ?

- Can we achieve **wost-case O(n log n) QuickSort**?
  - Yes! Find & use the median as pivot!
- What will be the **worst case time complexity** of this modified **QuickSort**?
  - **T(n) = O(1) if n=1**
  - **T(n) = 2T(n/2) + O(n) if n>1**
  - **O(n log n)**

▶QUIZ?

- Compare **MergeSort** with **QuickSort**?

Quick Sort Algorithm via D&C Visualization

- *Quick Sort Algorithm via D&C Visualization*

4. Matrix Multiplication via D&C

Matrix Multiplication - Standard

- Standard Matrix Multiplication
- **Algorithm 1.4**
- How many *? +/-?
  - 8 *
  - 4 +

Matrix Multiplication - Standard

Algorithm 1.4 Standard Matrix Multiplication via D&C

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

$$= \begin{bmatrix} A_{00} * B_{00} + A_{01} * B_{10} & A_{00} * B_{01} + A_{01} * B_{11} \\ A_{10} * B_{00} + A_{11} * B_{10} & A_{10} * B_{01} + A_{11} * B_{11} \end{bmatrix}$$

A, B: N x N matrices;

$\mathbf{A_{ij}}$, $\mathbf{B_{ij}}$: N/2 x N/2 matrices, where i, j $\in$ {0, 1}

Matrix Multiplication - Standard

- Analysis of Algorithm 1.4 Matrix Multiplication (Standard) – multiplications?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 8T(n/2) \quad \text{if } n>1$$

Solving Recurrences by Substitution

- $T(n) = 8T(n/2)$ if $n>1$
- $T(1) = 1$
  - $T(n) = 8\ T(n/2^1)$
  - $= 8\ \ 8\ T(n/2^2)$
  - $= 8\ 8\ 8\ T(n/2^3)$
  - $= 8\ 8\ 8\ 8\ T(n/2^4)$
  - .
  - .
  - $= 8\ 8\ 8 \ldots 8\ T(n/2^{\log n})$
  - $= 8^{\log_2 n}\ 1$
  - $= n^{\log_2 8}$
  - $= n^3$
  - $O(n^3)$

$$x^{\log_a y} = y^{\log_a x}$$

Example A.8

Matrix Multiplication - Standard

- **Algorithm 1.4** Matrix Multiplication (Standard) – multiplications?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 8T(n/2) \quad \text{if } n>1$$

- $T(n) = \mathbf{O(n^3)}$

- Can we do better?

Strassen's Matrix Multiplication via D&C

- Strassen's Matrix Multiplication
- Idea?
- Example 2.4
- Example 2.5
- How many *? +/-?
  - 7 *
  - 18 + & -

Strassen's Matrix Multiplication via D&C

Algorithm 2.8 Strassen's Matrix Multiplication via D&C

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

$$= \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

$M_1=(A_{00}+A_{11})*(B_{00}+B_{11})$
$M_2=(A_{10}+A_{11})*B_{00}$
$M_3=A_{00}*(B_{01}-B_{11})$
$M_4=A_{11}*(B_{10}-B_{00})$
$M_5=(A_{00}+A_{01})*B_{11}$
$M_6=(A_{10}-A_{00})*(B_{00}+B_{01})$
$M_7=(A_{01}-A_{11})*(B_{10}+B_{11})$

Strassen's Matrix Multiplication via D&C - Analysis

- Analysis of **Algorithm 2.8** Strassen's Matrix Multiplication Algorithm – multiplications?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 7T(n/2) \quad \text{if } n>1$$

$$T(n) = O(n^{\log 7}) \sim O(n^{2.81})$$

- Example B.2

$$x^{\log_a y} = y^{\log_a x}$$

Example A.8

▶QUIZ?

- Solving Recurrences by Substitution?

$$T(n) = O(1) \quad \text{if } n=1$$
$$T(n) = 7T(n/2) \quad \text{if } n>1$$

$$T(n) = O(n^{\log 7}) \sim O(n^{2.81})$$

Matrix Multiplication

- Standard Matrix Multiplication
  - $O(n^3)$
- Strassen's Matrix Multiplication
  - $O(n^{\log 7}) \sim O(n^{2.81})$

- Matrix multiplication lower bound is $\Omega(n^2)$.

✓ Divide-and-Conquer (D&C)-based Algorithm Analysis

The D&C Approach

- Three steps:
  - **Step 1:** Divide & Decrease an instance of a problem into one or more smaller instances.
  - **Step 2:** Conquer solve each of the smaller instances (use recursion until the instance is sufficiently small).
  - **Step 3:** Combine the solutions of the smaller instances to obtain the solution of the original instance.

D&C Algorithm Analysis

- The **running time of D&C algorithms** is mainly affected by 3 criteria:
  - The **number of sub-instances** (***a***) into which a problem is split.
  - The **ratio of initial problem size to sub-problem size** (***b***).
  - The **number of steps** required **to divide the initial instance and to combine sub-solutions**, expressed as **a function of the input size, n^k**.

D&C Algorithm Recurrence Equation

- A divide-and-conquer algorithm that instantiates ***a*** **sub-instances, each of size n/*b***:
  - **$T(n) = a\,T(n/b) + O(n^k)$** where k is constant
  - **$T(n_o) = d$** (constant)
    - **$T(n) = O(n^k)$ if $a < b^k$**
    - **$T(n) = O(n^k \log n)$ if $a = b^k$**
    - **$T(n) = O(n^{\log_b a})$ if $a > b^k$**

D&C Algorithm Recurrence Equation

- **Theorem B.5**
  - Example B.26
  - Example B.27

D&C Algorithm Recurrence Equation

- Examples:
  - $T(n) = T(n/2) + O(1)$
    - $T(n) = O(\log n)$
  - $T(n) = 2T(n/2) + O(1)$
    - $T(n) = O(n)$
  - $T(n) = 2\,T(n/2) + O(n)$
    - $T(n) = O(n \log n)$

▶QUIZ?

- Write a **divide-and-conquer-based recursive** algorithm for the **Towers of Hanoi** problem with n disks.
- Recurrence relation?
  - $T(1) = 1$
  - $T(n) = 2\,T(n-1) + 1$
  - $O(2^n)$
- Exercise 17

✓ When Not to Use D&C?

When Not to Use D&C?

- D&C algorithms should be avoided in the following two cases:
  - An instance of size *n* is divided into two or more instances each **almost of size *n***.
  - An instance of size *n* is divided into **almost *n* instances** of size *n/c*, where *c* is a constant.

Computing N-th Fibonacci Term via D&C

- (0,) 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
- **Algorithm 1.6** N-th Fibonacci Term (Recursive)
  - **Divide-and-Conquer**
  - **Top-down**
  - **Exponential time $O(2^N)$**
- Fib(n) = Fib(n – 1) + Fib(n – 2) for n >= 2;
- Fib(0) =0
- Fib(1) = 1
- **The, what?**
  - **DP**

✓ Divide-and-Conquer (D&C)-based Algorithms Summary

1. Binary Search – Searching via D&C
2. Merge Sort – Sorting via D&C
3. Quick Sort – Sorting via D&C
4. Matrix Multiplication via D&C

5. Quick Select & MM Select – Selection via D&C

Homework Assignment

Homework Assignment?

- Chapter 2: Exercise #9 and #20.
- What is the worst-case time complexity of MergeSort using O(n2) time & O(1) space merge? Explain.
- What is the average-case time complexity of Quicksort? Explain

✓ Textbook Readings

- Chapter 2:
  - 2.1
  - 2.2
  - 2.3
  - 2.4
  - 2.5
  - 2.8

the right majors, minors &

ucation?

or students' academic and career success

ging for many students - *Many students change their*
uring college!

e prediction of student success in MMC could
dual students
d their right MMC
chieve their academic goals

END

Prof. Young Park