Analysis

Kuan-Yu Chen (陳冠宇)

2019/02/27 @ TR-310-1, NTUST

Review

- **Algorithm:** Any well-defined computation procedure that takes some value, or set of values, as input and produces some value, or set of values, as output

- Analyzing an algorithm has come to mean predicting the **resources** that the algorithm requires
  - Most often we want to measure the computational time

- For insertion sort
  - The best case

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8) \times n - (c_2 + c_4 + c_5 + c_8)$$

  - The worst case

$$T(n) = \left(\frac{c_5 + c_6 + c_7}{2}\right) \times n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5 - c_6 - c_7}{2} + c_8\right) \times n - (c_2 + c_4 + c_5 + c_8)$$

Merge Sort.

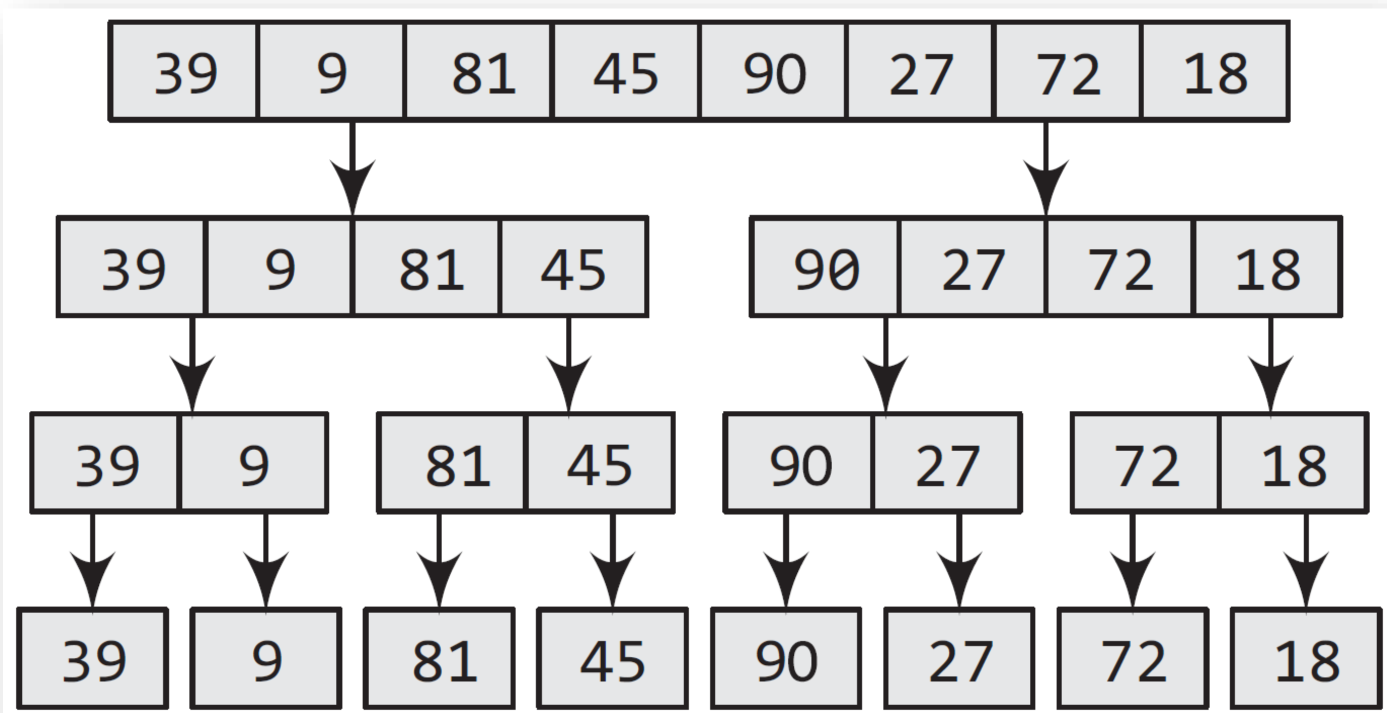
- Merge sort is a sorting algorithm that uses the **divide**, **conquer**, and **combine** algorithmic paradigm
  - ***Divide*** means partitioning the n-element array to be sorted into two sub-arrays
    - Divide the problem into a number of subproblems that are smaller instances of the same problem
  - ***Conquer*** means sorting the two sub-arrays recursively
    - Conquer the subproblems by solving them recursively
    - If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner
  - ***Combine*** means merging the two sorted sub-arrays
    - Combine the solutions to the subproblems into the solution for the original problem

Example.

- Sort the given array using merge sort

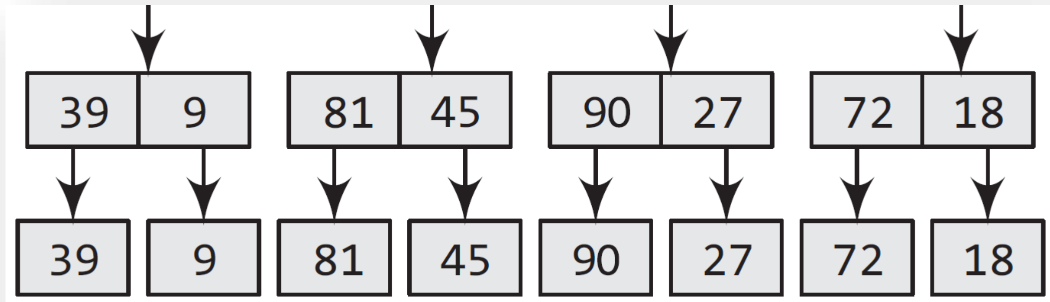
39	9	81	45	90	27	72	18

  - Divide and Conquer

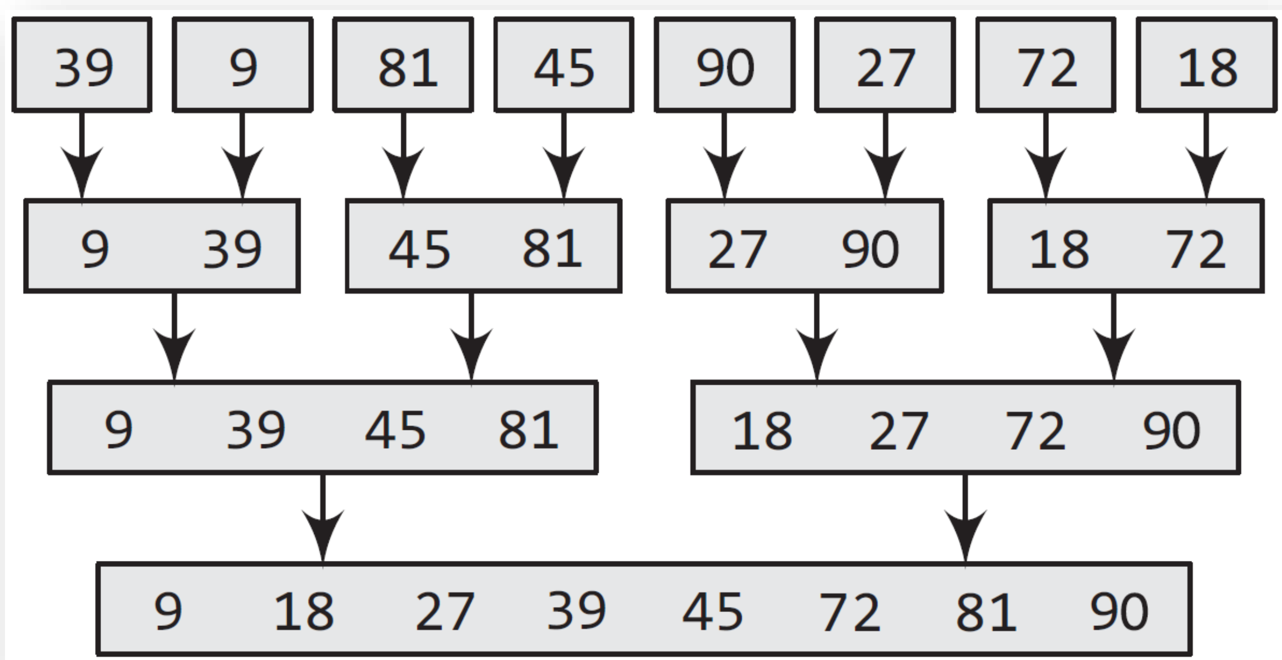


Example..

- Divide and Conquer

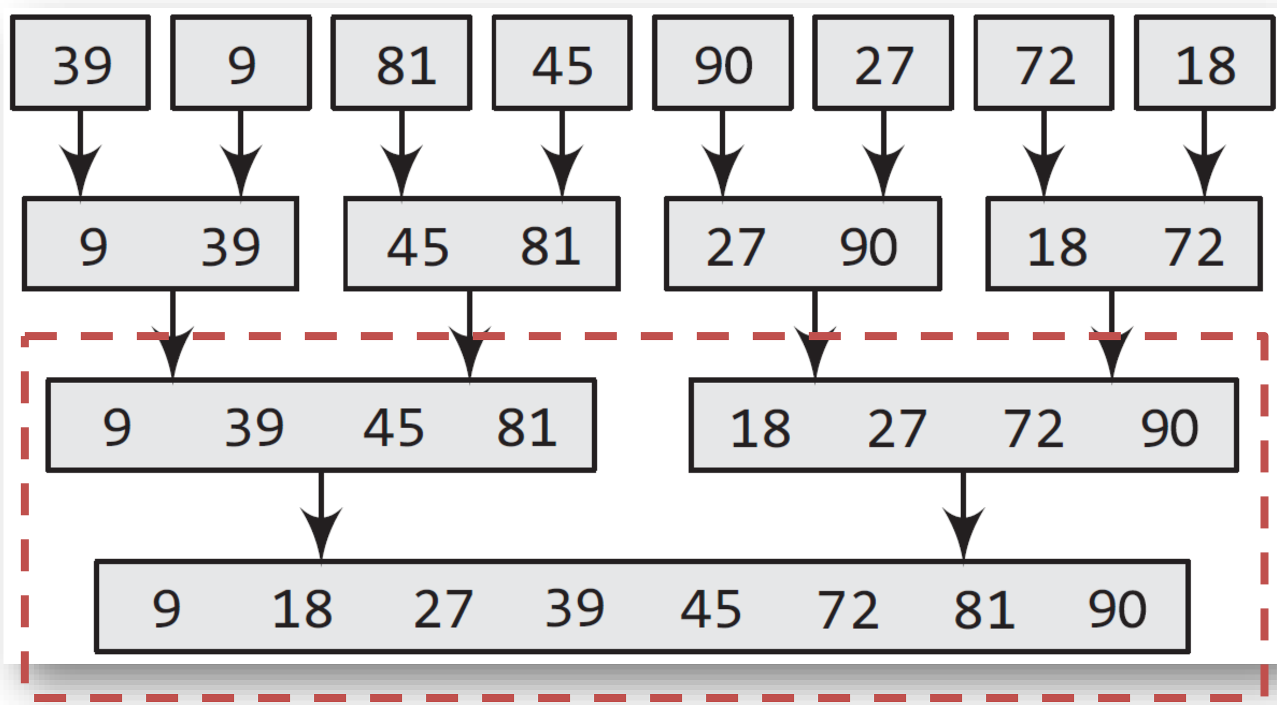


- Combine



Merge Sort..

- The concept of the merge function is to compare two sub-arrays (ARR[I] and ARR[J]), the smaller of the two is placed in a temp array (TEMP) at the location specified by a index (INDEX) and subsequently the index value (I or J) is incremented
  - Example for the merge function



Merge Sort....

								TEMP							
9	39	45	81	18	27	72	90	9							
BEG, I			MID	J			END	INDEX							

								TEMP							
9	39	45	81	18	27	72	90	9	18						
BEG	I		MID	J			END		INDEX						

9	39	45	81	18	27	72	90	9	18	27					
BEG	I		MID		J		END			INDEX					

9	39	45	81	18	27	72	90	9	18	27	39				
BEG	I		MID			J	END				INDEX				

9	39	45	81	18	27	72	90	9	18	27	39	45			
BEG		I	MID			J	END					INDEX			

9	39	45	81	18	27	72	90	9	18	27	39	45	72		
BEG			I, MID			J	END						INDEX		

9	39	45	81	18	27	72	90	9	18	27	39	45	72	81	
BEG			I, MID				J END							INDEX	

9	39	45	81	18	27	72	90	9	18	27	39	45	72	81	90
BEG			MID	I			J END								INDEX

Merge Sort...

```
MERGE_SORT(ARR, BEG, END)
Step 1: IF BEG < END
            SET MID = (BEG + END)/2
            CALL MERGE_SORT (ARR, BEG, MID)
            CALL MERGE_SORT (ARR, MID + 1, END)
            MERGE (ARR, BEG, MID, END)
        [END OF IF]
Step 2: END
```

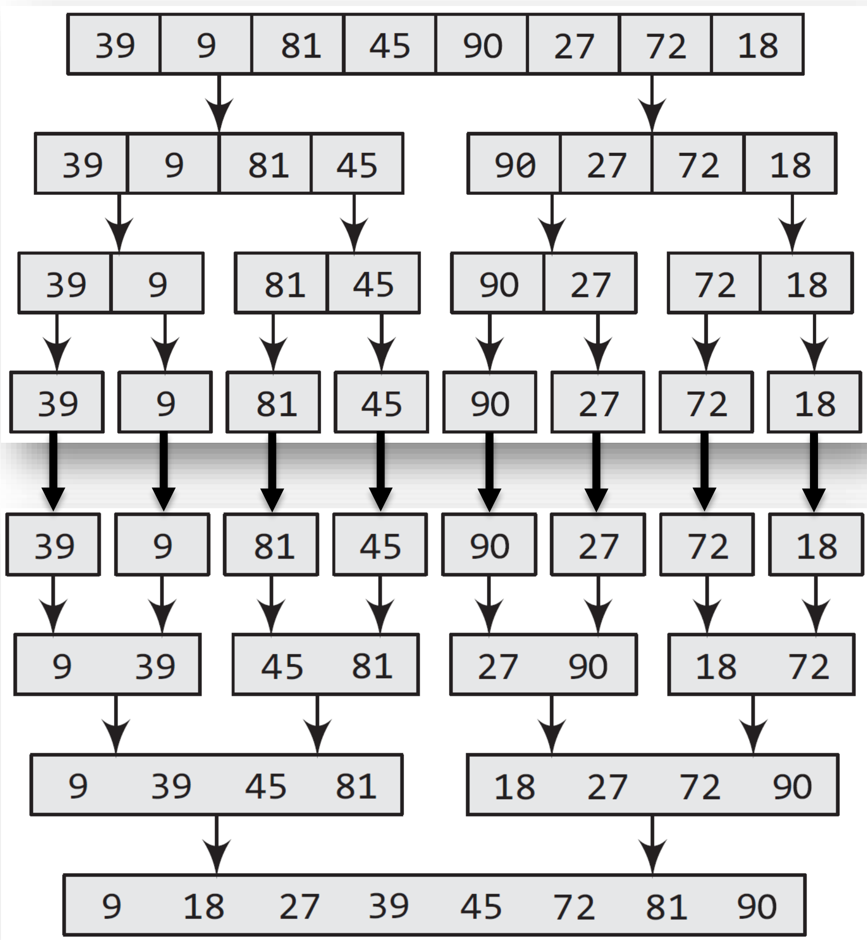
Merge Sort.....

```
MERGE (ARR, BEG, MID, END)

Step 1: [INITIALIZE] SET I = BEG, J = MID + 1, INDEX = 0
Step 2: Repeat while (I <= MID) AND (J<=END)
            IF ARR[I] < ARR[J]
                  SET TEMP[INDEX] = ARR[I]
                  SET I = I + 1
            ELSE
                  SET TEMP[INDEX] = ARR[J]
                  SET J = J + 1
            [END OF IF]
            SET INDEX = INDEX + 1
      [END OF LOOP]
```

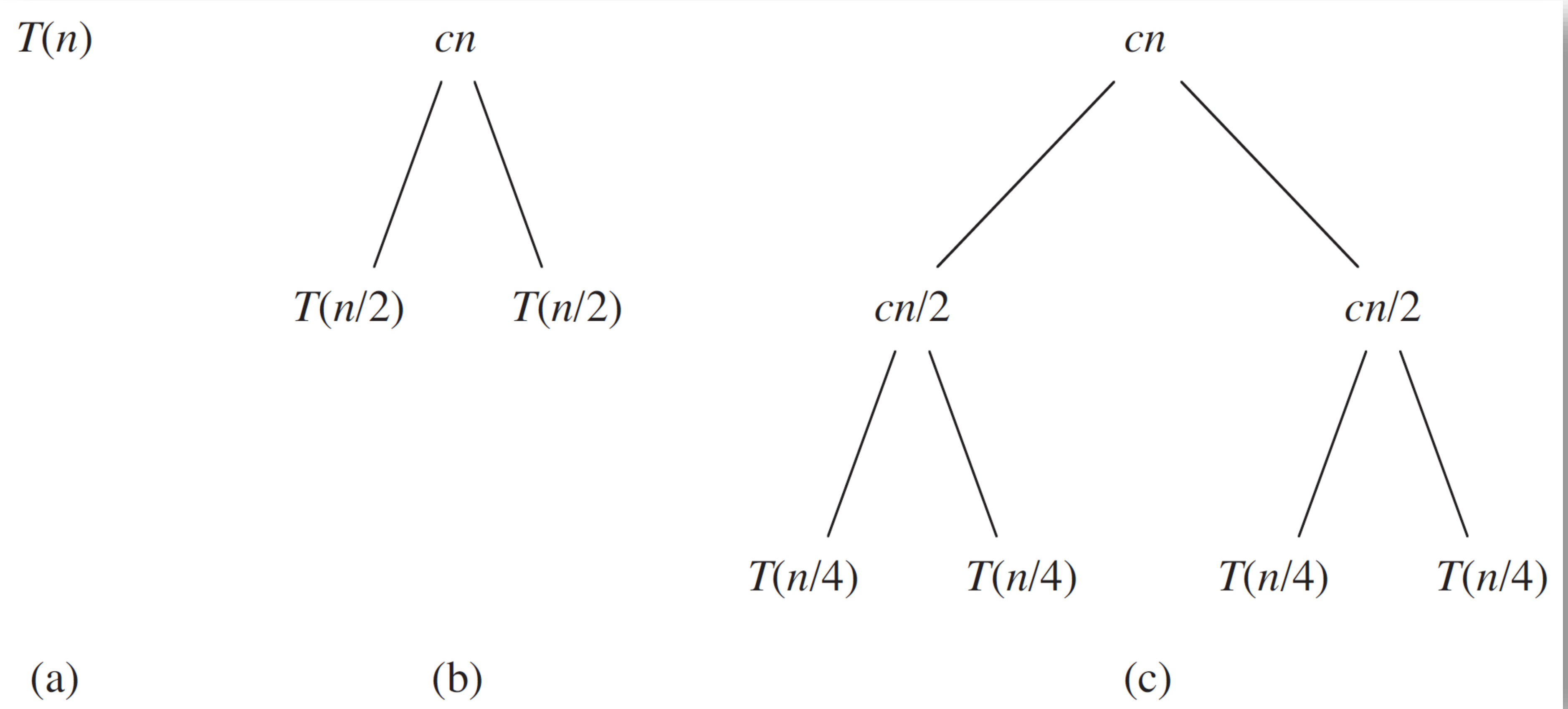
Analysis.

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

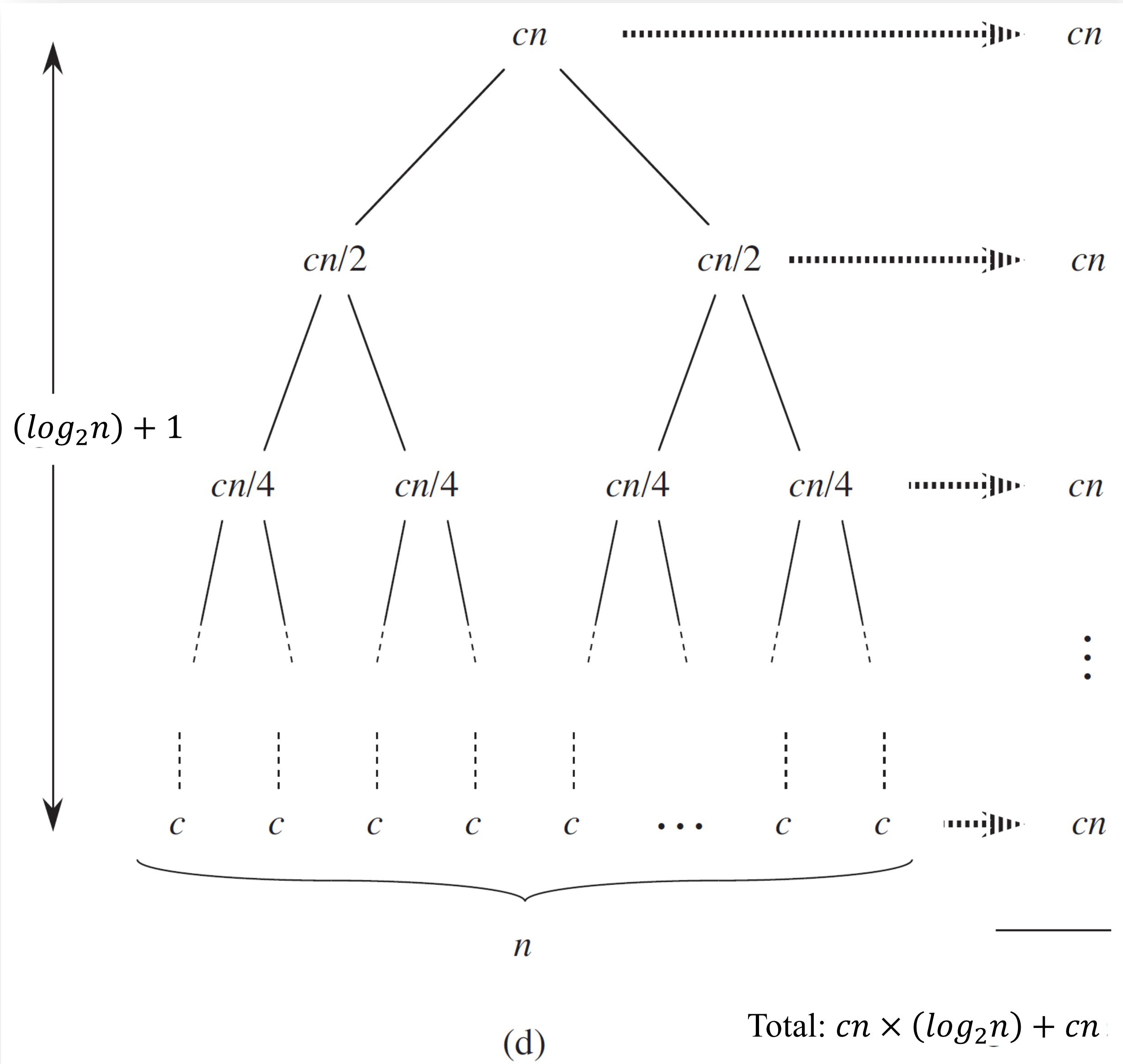


- Divide
  - The divide step just computes the middle of the subarray, which takes constant time $D(n)$
- Conquer
  - We recursively solve two subproblems $T(n) = 2 \times T(\frac{n}{2})$
- Combine
  - We have already noted that the Merge procedure on an n-element subarray takes time $C(n)$

Analysis..



Analysis...



Faster than insertion sort!

Design an Algorithm

- We can choose from a wide range of algorithm design techniques
  - Incremental Approach
    - Insertion Sort
  - Divide-and-conquer Approach
    - Merge Sort
    - One advantage of divide-and-conquer algorithms is that their running times are often easily determined

Questions?

kychen@mail.ntust.edu.tw