# 01 Important Problem Types

## I. Why Study Algorithms?
- Learn standard algorithms
- Design new algorithms
- Analyze algorithm efficiency

Important problem types
- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

## II. Sorting
- To arrange items in either ascending or descending order
- Why is sorting important in CS?
  - Efficient searching
  - Foundation for other algorithms
    - Merging, duplicate removal, etc.
  - Real-World Applications
    - Crucial for databases, file systems, and improving user experience (e.g. ranked results)
- Sorting: Time Efficiency
  - $O(n \cdot \log n)$ is considered fastest time complexity for sorting

- Stable Sorting
  - A sorting algorithm is stable if the algorithm keeps the relative order of equal elements in input
  - Ex:

    Input: 5, 4, 3, 4, 12, 4

    Output: 3, 4, 4, 4, 5, 12

    Three 4's in both input & output

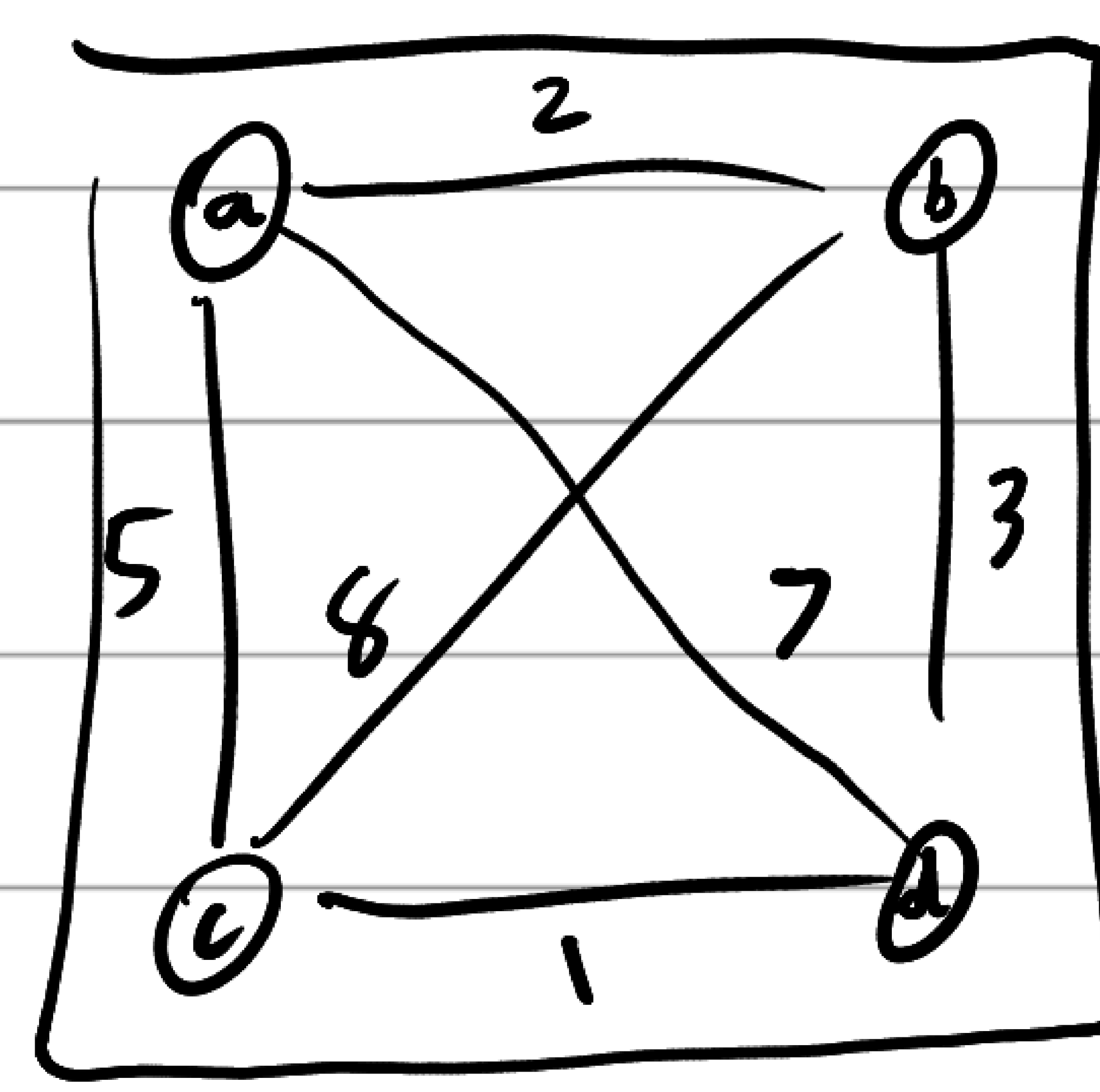
  - Stable Sorting Algorithms
    - Merge, Insertion, Bubble, etc
  - Unstable Sorting Algorithms
    - Quick, Heap, Selection, etc
- In-Place Sorting Algorithm
  - An algorithm is in-place if it doesn't require extra memory, except a few memory units
    - Bubble, Selection, Insertion, Heap, Quick, etc
  - Tradeoff between space for speed (often)

## III. Searching

- Finding a target element, known as a <u>search key</u>, among the elements of a list
- Sequence Search
- Binary Search
- Hashing

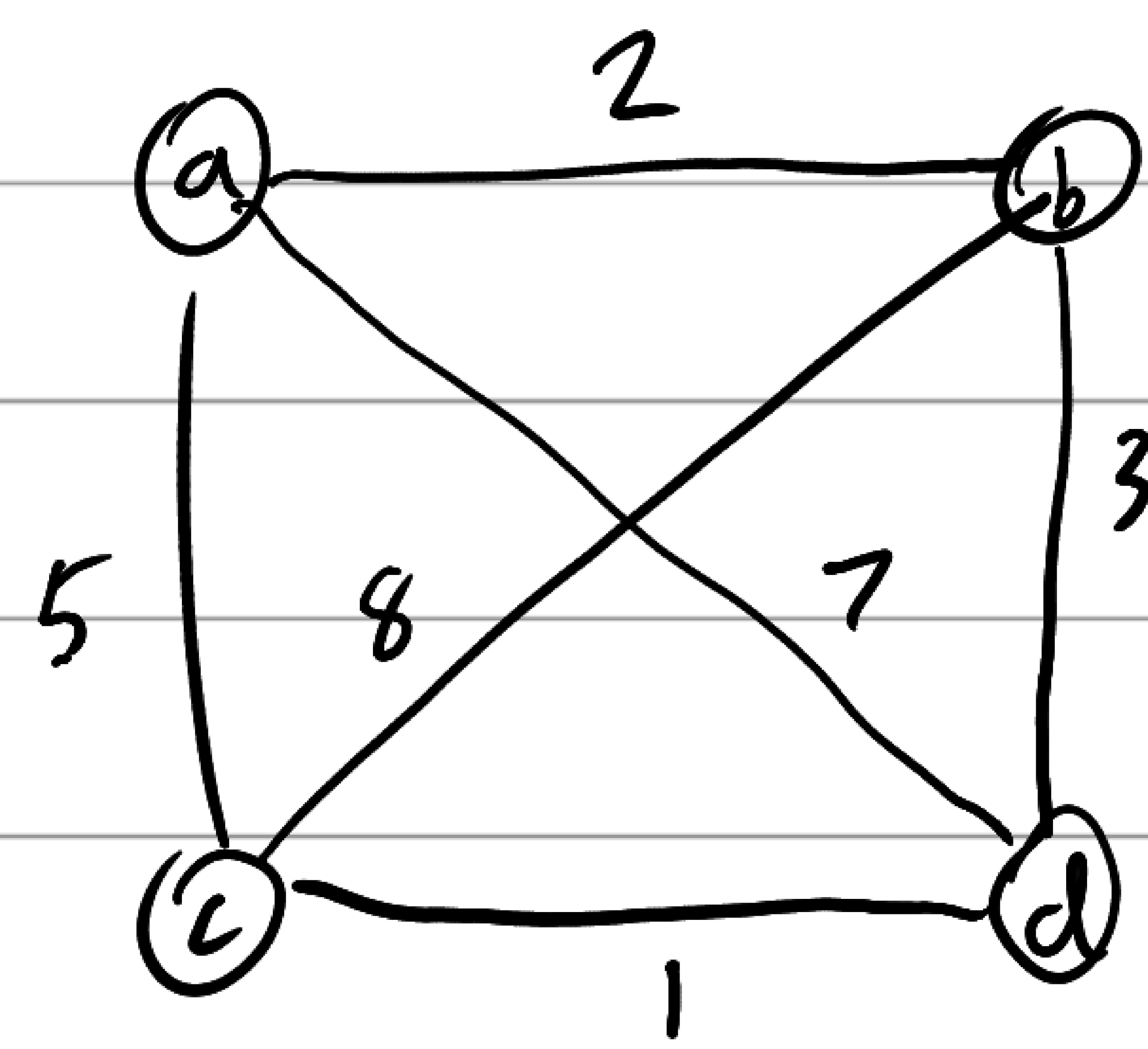
## IV. Graph Problems

- A graph in computer science is composed of vertices (nodes) & edges
  - Ex:


- Used for modeling many real-life applications
  - transportation, communication networks, project scheduling
- Graph traversal problem
- Shortest path problem

- Travelling Salesman Problem (TSP)
    - "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once & returns to origin city?"
    - Ex:




Solution:

$a \to b \to d \to c \to a$ or $a \to c \to d \to b \to a$ → 11

- Brute force – Check all permutations
    - Δ Layout all paths & compare length
    - Δ $(n-1)!$ possibilities

# V. Combonitorial Problems

- Finding a combinatorial object (permutation, combination or subset that satisfies condition or desired property)
    - TSP, Knapsack problem
- Most difficult problems in computing

- Number of objects grows very fast

- Knapsack Problem
  - Given $n$ items
    - Weights: $w_1, w_2, \ldots, w_n$
    - Values: $v_1, v_2, \ldots, v_n$
  - Find most valuable subset of items that fit into knapsack w/ capacity $W$
  - One quantity per item
  - Ex:

    Capacity: 10 ($W=10$)
    Item 1: $w_1 = 7$, $v_1 = \$42$
    Item 2: $w_2 = 3$, $v_2 = \$12$
    Item 3: $w_3 = 4$, $v_3 = \$40$
    Item 4: $w_4 = 5$, $v_4 = \$25$

    Answer: (3, 4)

  - Brute force (Exhaustive Search)
    - Check all possible subsets
    - $2^n$ possibilities