CS483-06 Brute Force & Divide and Conquer

Instructor: Fei Li

Room 443 ST II

Office hours: Tue. & Thur. 1:30pm - 2:30pm & 4:30pm - 5:30pm or by appointments

`lifei@cs.gmu.edu` with subject: CS483

`http://www.cs.gmu.edu/~ lifei/teaching/cs483_fall07/`

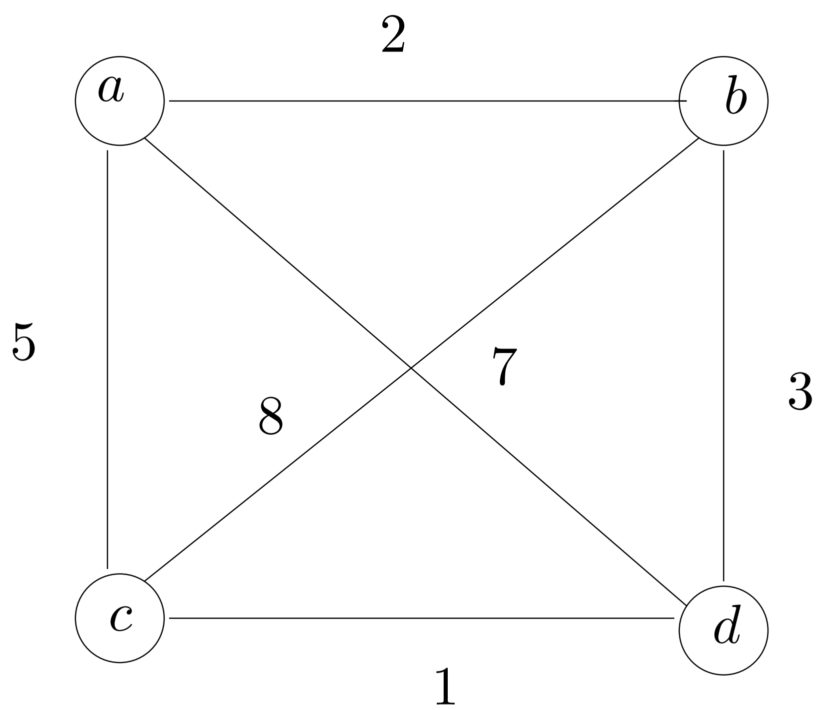
This lecture note is based on notes by Anany Levitin and Jyh-Ming Lian.

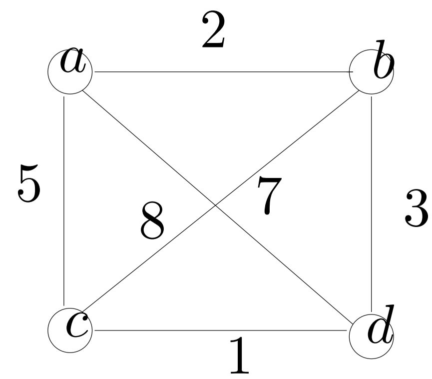
Outline

- Brute Force
  - Examples: Exhaustive Search
- Divide and conquer
  - Ideas
  - Analysis: Master Theorem
  - Examples: Mergesort

Traveling Salesman Problem

➢ **TSP**: Find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it starts.





Tour	Cost
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$2 + 3 + 7 + 5 = 17$
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$2 + 4 + 7 + 8 = 21$
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$8 + 3 + 4 + 5 = 20$
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$8 + 7 + 4 + 2 = 21$
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$5 + 4 + 3 + 8 = 20$
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$5 + 7 + 3 + 2 = 17$

Traveling Salesman Problem

Analysis

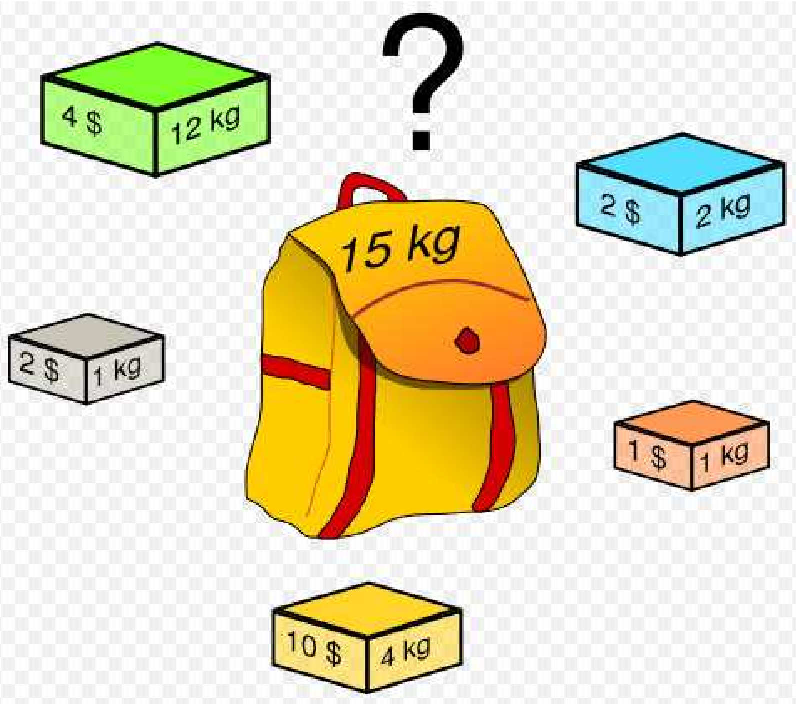
- Input size: $n + n \cdot (n-1)/2 = n \cdot (n-1)/2$.
- Running time:

$$T(n) = (n-1)!.$$

Knapsack Problem

➢ **Knapsack Problem**: Given n objects, each object i has weight w_i and value v_i, and a knapsack of capacity W (in terms of weight), find most valuable items that fit into the knapsack

Items are not splittable



http://en.wikipedia.org/wiki/Knapsack_problem

Example: Knapsack capacity $W = 16$

Item	Weight	Value
1	2	$\$20$
2	5	$\$30$
3	10	$\$50$
4	5	$\$10$

Subset	Total weight	Total value
$\{1\}$	2	$\$20$
$\{2\}$	5	$\$30$
$\{3\}$	10	$\$50$
$\{4\}$	5	$\$10$
$\{1,2\}$	7	$\$50$
$\{1,3\}$	12	$\$70$
$\{1,4\}$	7	$\$30$
$\{2,3\}$	15	$\$80$
$\{2,4\}$	10	$\$40$
$\{3,4\}$	15	$\$60$
$\{1,2,3\}$	17	not feasible
$\{1,2,4\}$	12	$\$60$
$\{1,3,4\}$	17	not feasible
$\{2,3,4\}$	20	not feasible
$\{1,2,3,4\}$	22	not feasible

Knapsack Problem

Analysis

- Input size: n (items).
- Running time:

  The number of subsets of an n-element set is 2^n, including $\emptyset$.

$$T(n) = \Omega(2^n).$$

Assignment Problem

- **Assignment Problem**: There are n people to execute n jobs, one person per job. If ith person is assigned the jth job, the cost is $C[i, j]$, $i, j = 1, \ldots, n$. Find the assignment with the minimum total cost.

	Job 1	**Job 2**	**Job 3**	**Job 4**
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

Assignment Problem

Analysis

- Input size: n.
- Running time:

$$T(n) = n!.$$

Summary for Brute Force

➢ Strengths

1. Wide applicability
2. Simplicity
3. Yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)
4. In many cases, exhaustive search or its variation is the only known way to get exact solution

➢ Weaknesses

1. Rarely yields efficient algorithms. Some brute-force algorithms are unacceptably slow
2. Not as constructive as some other design techniques
3. Exhaustive-search algorithms run in a realistic amount of time only on very small instances

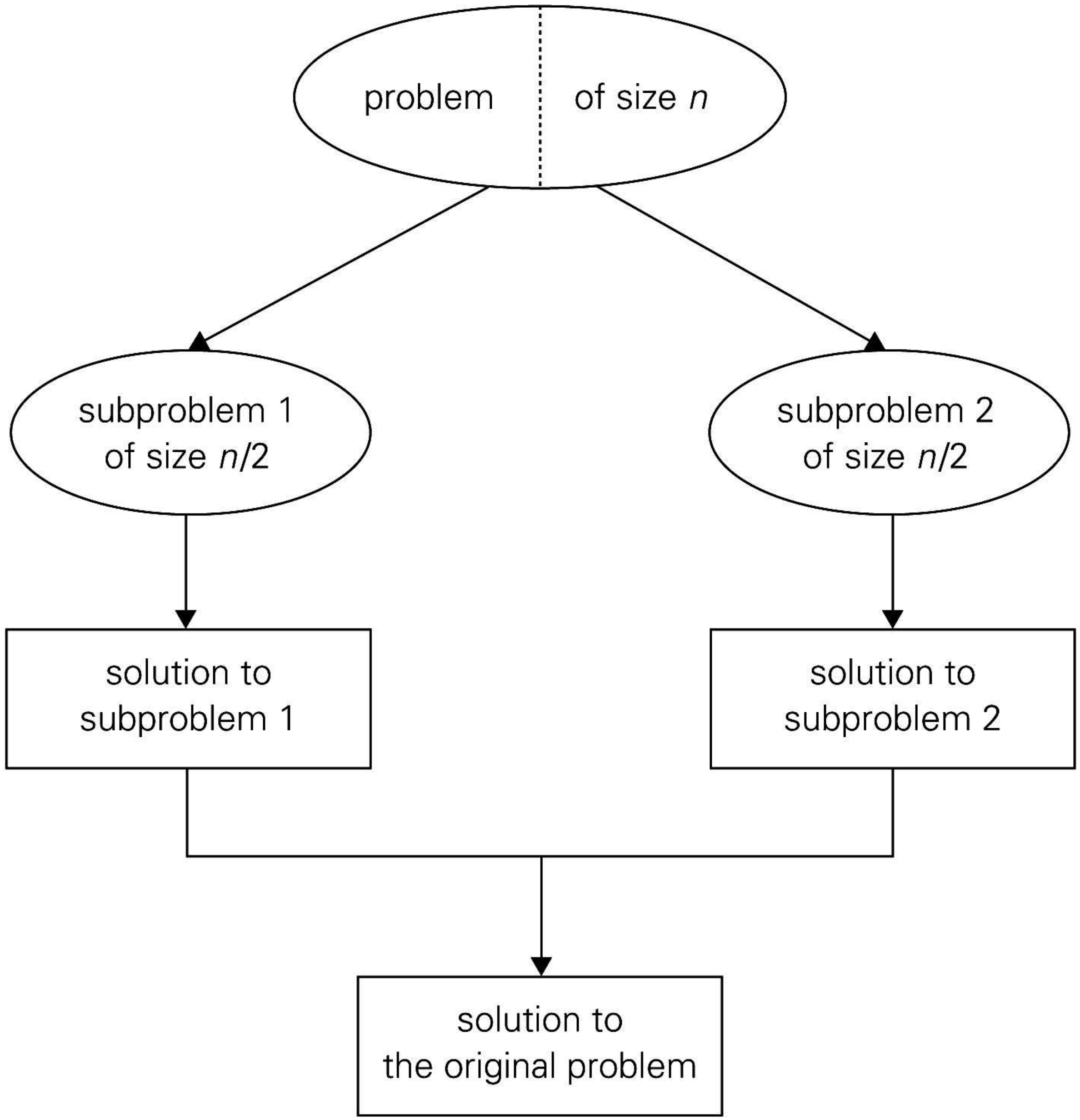
Outline

- Brute Force
  - Examples: Exhaustive Search
- Divide and conquer
  - Ideas
  - Analysis: Master Theorem
  - Examples: Mergesort

Divide and Conquer

The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances
2. Solve smaller instances recursively
3. Obtain solution to original (larger) instance by combining these solutions


problem
of size n
subproblem 1
of size $n/2$
subproblem 2
of size $n/2$
solution to
subproblem 1
solution to
subproblem 2
solution to
the original problem

Outline

- Brute Force
  - Examples: Exhaustive Search
- Divide and conquer
  - Ideas
  - Analysis: Master Theorem
  - Examples: Mergesort

General Divide-and-Conquer Recurrence

- Problem size: n. Divide the problems into b smaller instances; a of them need to be solved. $f(n)$ is the time spent on dividing and merging.
- **Master Theorem**: If $f(n) \in \Theta(n^d)$, where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

- Examples:
  1. $T(n) = 4T(n/2) + n \Rightarrow T(n) =$
  2. $T(n) = 4T(n/2) + n^2 \Rightarrow T(n) =$
  3. $T(n) = 4T(n/2) + n^3 \Rightarrow T(n) =$

Summary: Algorithm Analysis

- Recursive algorithms
  a. The iteration method
  b. The substitution method
  c. Master Theorem ($T(n) = aT(n/b) + f(n)$.)

Outline

- Brute Force
  - Examples: Exhaustive Search
- Divide and conquer
  - Ideas
  - Analysis: Master Theorem
  - Examples: Mergesort

Sorting Problem

- Given an array of n numbers, sort the elements in non-decreasing order.
- Input: An array $A[1, \ldots, n]$ of numbers
- Output: An array $A[1, \ldots, n]$ of sorted numbers

Mergesort - Algorithm

- Given an array of n numbers, sort the elements in non-decreasing order.

Algorithm 0.1: MERGESORT($A[1, \ldots n]$)

if $n = 1$

then return (A)

else $\begin{cases} B \leftarrow A[1 \cdots \lfloor \frac{n}{2} \rfloor] \\ C \leftarrow A[\lceil \frac{n}{2} \rceil \cdots n] \\ MergeSort(B) \\ MergeSort(C) \\ Merge(B, C, A) \end{cases}$

- Is this algorithm complete?

Mergesort - Algorithm

- Merge two sorted arrays, B and C and put the result in A

Algorithm 0.2: MERGE($B[1, \ldots p], C[1, \ldots q], A[1, \cdots p+q]$)

$i \leftarrow 1; j \leftarrow 1$

for $k \in \{1, 2, \ldots p+q-1\}$

do $\begin{cases} \textbf{if } B[i] < C[j] \\ \quad \textbf{then } A[k] = B[i]; i \leftarrow i+1 \\ \quad \textbf{else } A[k] = C[j]; j \leftarrow j+1 \end{cases}$

- Example: 24, 11, 91, 10, 22, 32, 22, 3, 7, 99

Mergesort - Analysis

- All cases have same time efficiency: $\Theta(n \log_2 n)$

  $T_{\text{merge}}(n) = n - 1.$

$$T(n) = 2T(n/2) + n - 1, \quad \forall n > 1, \quad T(1) = 0$$

- Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting: $\lceil \log_2 n! \rceil \approx n \log_2 n - 1.44n$
- Space requirement: $\Theta(n)$ (not *in-place*) (In-place: The number are rearranged within the array.)
- Can be implemented without recursion?
- Is this algorithm `Mergesort` stable?