CPE231RC — Algorithms — Lecture2

BRUTE FORCE& EXHUASTIVE SEARCH

Dr. Prapong Prechaprapranwong

Computer Engineering

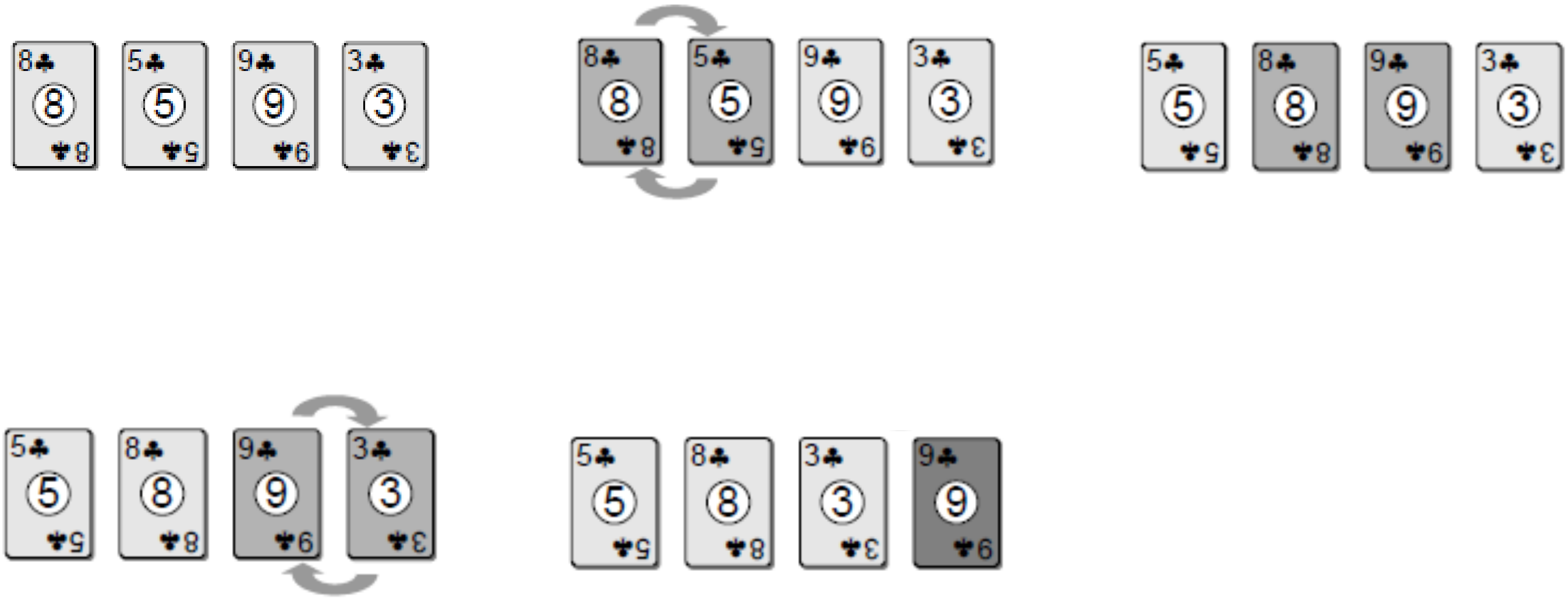
King Mongkut's University of Technology Thonburi

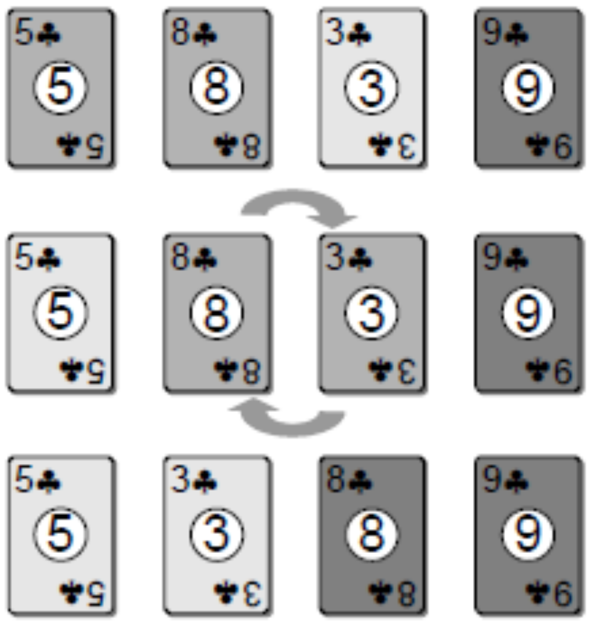
Brute-Force

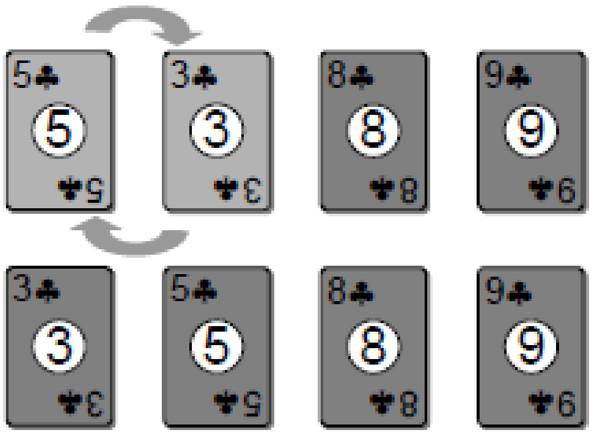
a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved

- Sorting
  - Bubble sort
  - Selection sort
- String matching
- Closest-pair
- Convex Hull

Bubble sort







Algorithm BubbleSort (A [0..n-1])

// Sorts a given array by bubble sort

Input: An array A[0..n-1] of orderable elements

Output: Array A[0..n-1] sorted in nondecreasing order

```
for i ← 0 to n-2 do
    for j ← 0 to n-2-i do
        if A[j+1] < A[j]
            swap A[j] and A[j+1]
```

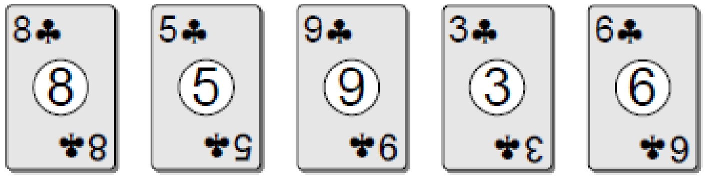
Complexity

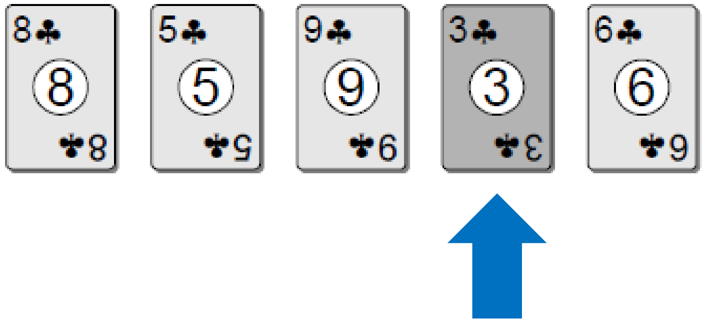
$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$

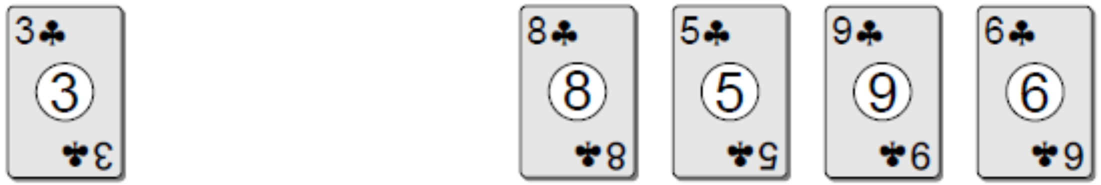
$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in O(n^2)$$

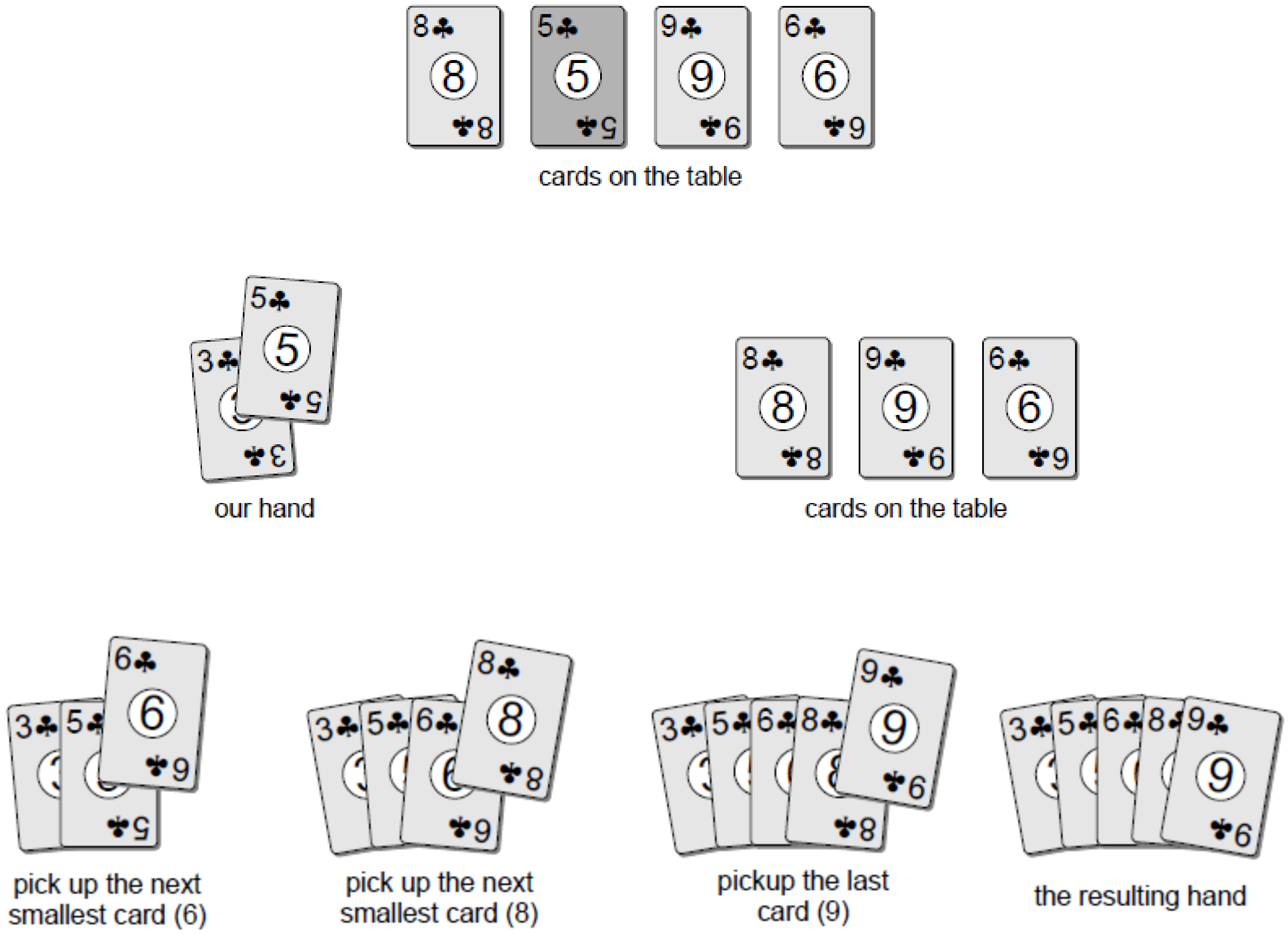
Practice1.1

Selection Sort








cards on the table
our hand
cards on the table
pick up the next smallest card (6)
pick up the next smallest card (8)
pickup the last card (9)
the resulting hand

Algorithm SelectionSort (A[0..n-1])

// Sorts a given array by selection sort

Input: An array A[0..n-1] of orderable elements

Output: Array A[0..n-1] sorted in nondecreasing order

1. **for** i ← 0 **to** n-2 do
2.   min ← i
3.   **for** j ← i +1 **to** n-1 do
4.   **if** A[j] < A[min]
5.   min ← j
6.   swap A[i] and A[min]

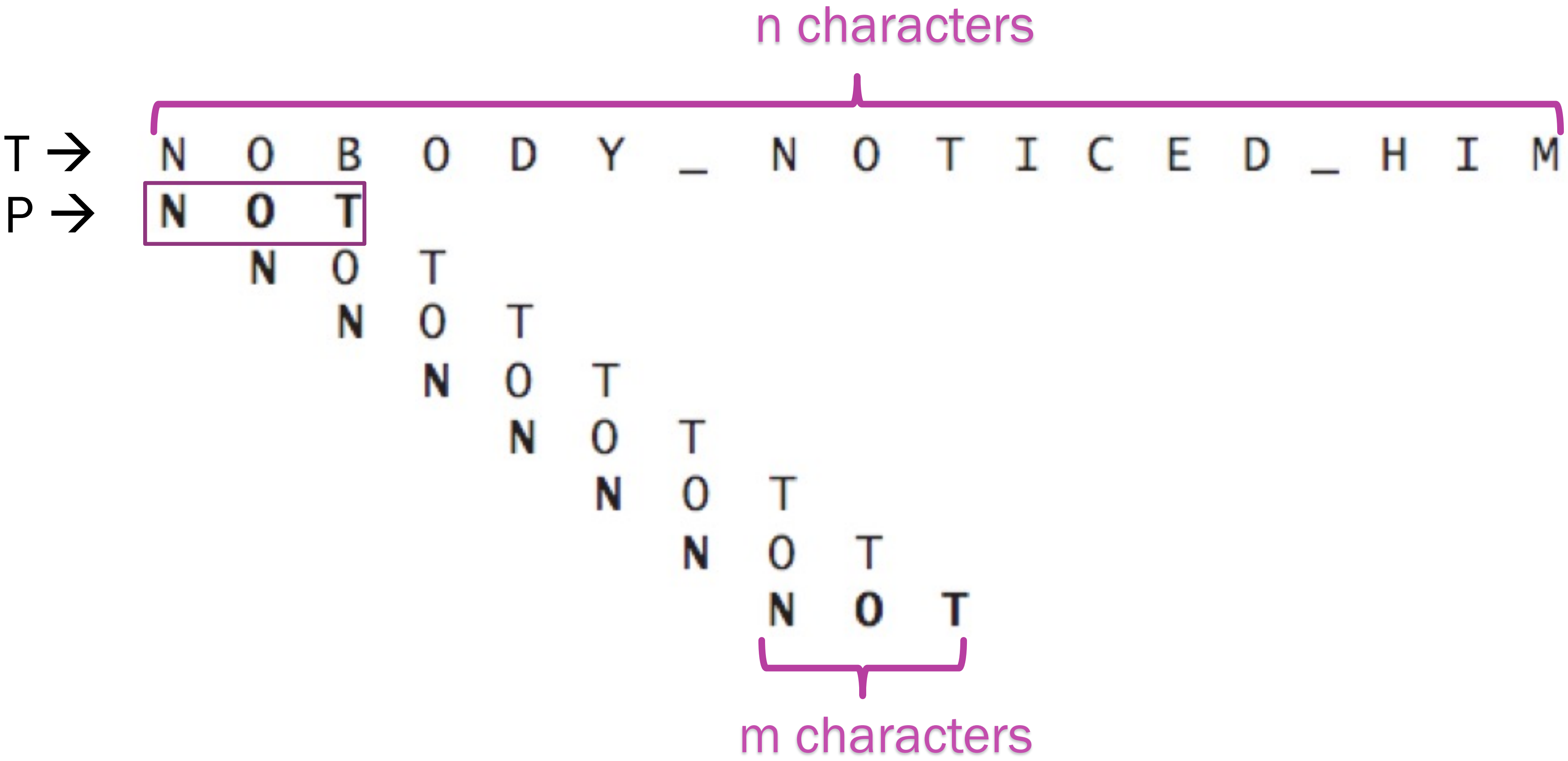
Complexity

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in O(n^2)$$

Brute-Force String Matching

Objective: Determine if a given pattern P (m characters) is in a text T (n characters, n ≥ m)



Algorithm BruteForceStringMatching

```
Input: Pattern P = p_1 p_2 p_3 ··· p_m and Text T = t_1 t_2 t_3 ··· t_n, n ≥ m
Output: Index of the 1st character in the text that starts a matching
            substring, and −1 for the unsuccessful search

for i := 1 to [ n-m+1 ] do
    j ← 1
    while j ≤ m and [ p_j == t_{i+j-1} ] do
        j ← j + 1
    if j == m + 1 return i
return −1
```

```
def brute_stringmatch (P,T):
#Return the lowest index of T at which substring P begins or -1 for not found.
    n,m = len(T),len(P)
    for i in [fill code here]
        j = 0
        while j < m and [fill code here]
            j += 1
        if j == m:
            return i
    return -1
```

Practice 2: fill the code

Closest-Pair Problem

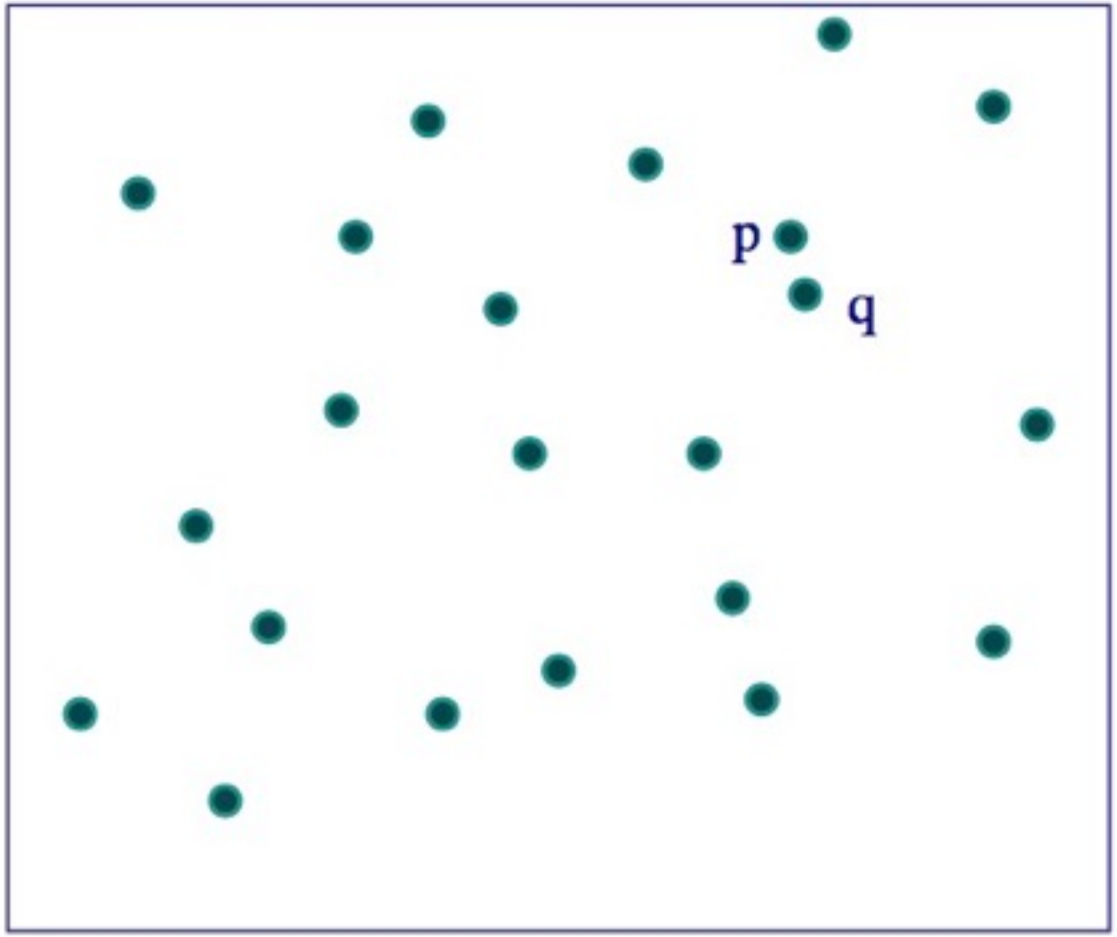
ALGORITHM *BruteForceClosestPair*(P)

//Finds distance between two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $p_1(x_1, y_1), \ldots, p_n(x_n, y_n)$
//Output: The distance between the closest pair of points

$d \leftarrow \infty$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  for $j \leftarrow i + 1$ **to** n **do**
    $d \leftarrow \min(d, sqrt((x_i - x_j)^2 + (y_i - y_j)^2))$ //*sqrt* is square root
return d

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n - i)$$

$$= 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2).$$



Practice3

Euclidean distance

$$dist(a,b) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

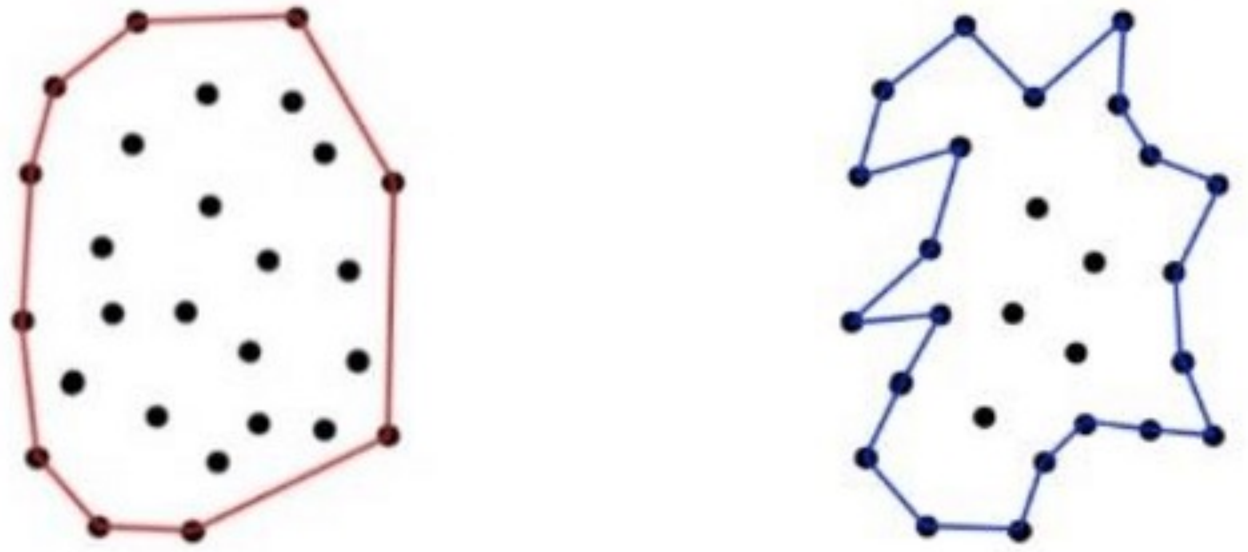
```
def dist(a,b):
    x=pow((a[0]-b[0]),2)
    y=pow((a[1]-b[1]),2)
    return sqrt(x+y)
```

⇓

```
def dist(a,b):
    x = a[0] - b[0]
    y = a[1] - b[1]
    return x*x + y*y
```

Convex-Hull Problem

The convex-hull problem is the problem of constructing the convex hull for a given set S of n points



(a) Convex hull (b) Concave hull

more information

https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/

Exhaustive Search

uses a brute-force approach to combinational problems.

Exhaustive search find the solution by follow these simple steps

1. generating every element of the problem
2. selecting those that satisfy all the constraints
3. finding a desired solution

- Traveling Salesman Problems
- Knapsack Problems
- Assignment Problems

Traveling Salesman Problem

To find the shortest tour through a given set of n cities that visits each city exactly once before return the city

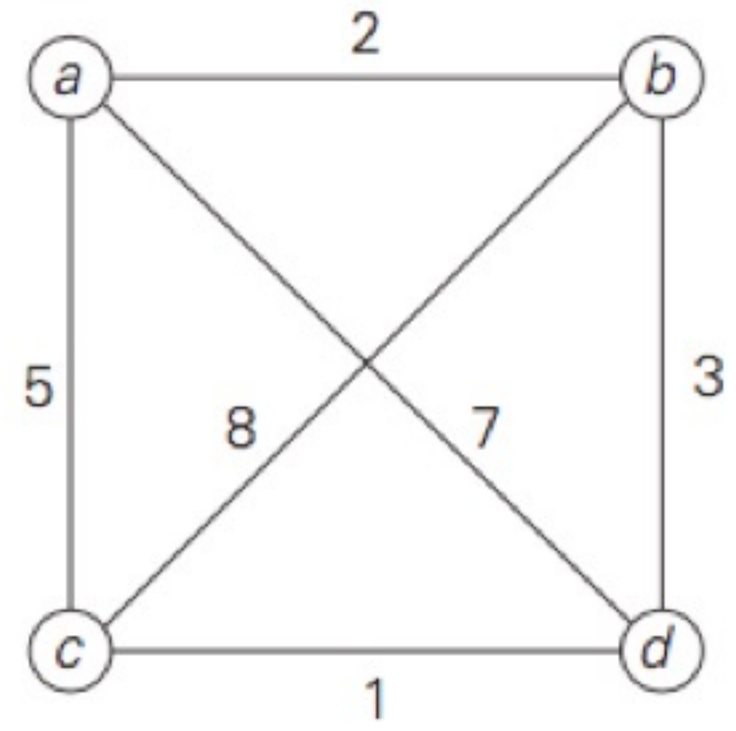
The problem can be modeled by a weight graph, which

- vertices represent the cities
- the edge's weight represent the distances

✓ finding the shortest Hamiltonian circuit (a cycle that passes through all the vertices of the graph exactly once) of the graph

is the same as "Minimum Spanning Tree" ?

Traveling Salesman Problem



Tour	Length
a → b → c → d → a	
a → b → d → c → a	
a → c → b → d → a	
a → c → d → b → a	
a → d → b → c → a	
a → d → c → b → a	

Knapsack Problem

Given n items of know weight w_1, w_2,…,w_n and values v_1, v_2, …, v_n and a knapsack of capacity W

✓ find <u>the most value</u> subset of the items that fit in to the knapsack

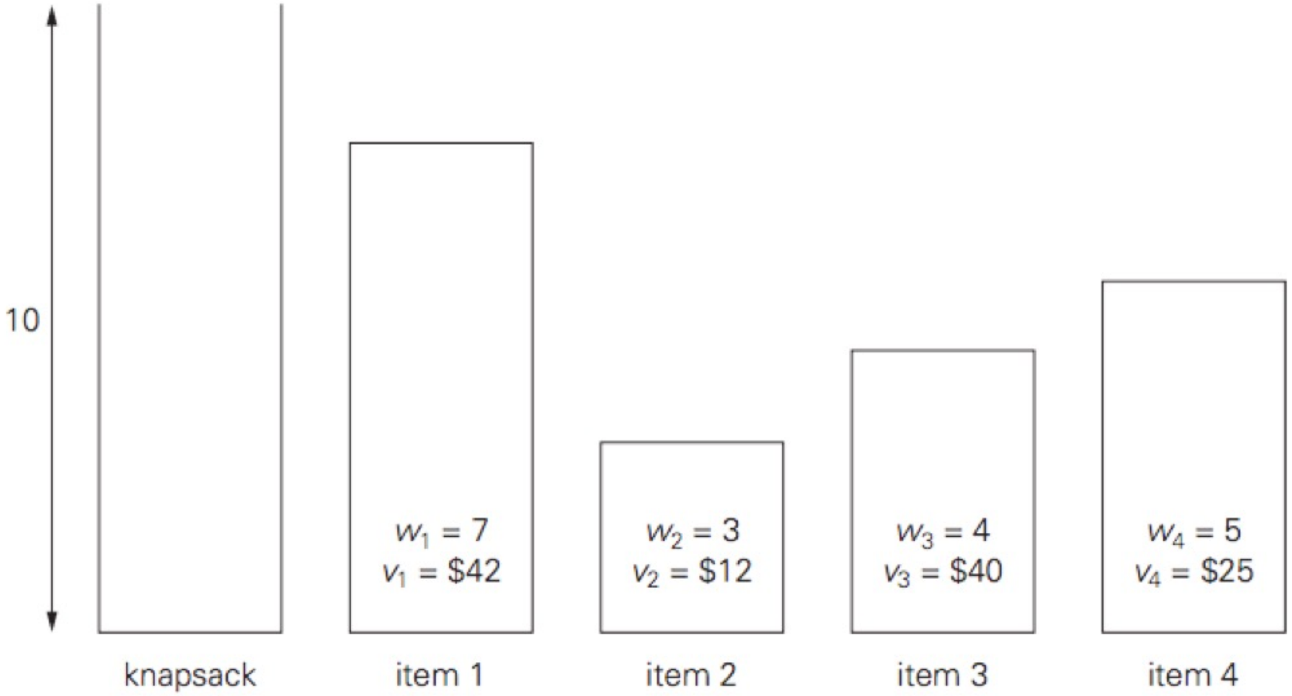
Capacity W = 15kg

$5 $2 $4

12 kg 10kg 5kg

Step to solve

1. Generating all possible subset of the items
2. Calculate all weight of those subset in knapsack
3. find the most value subset



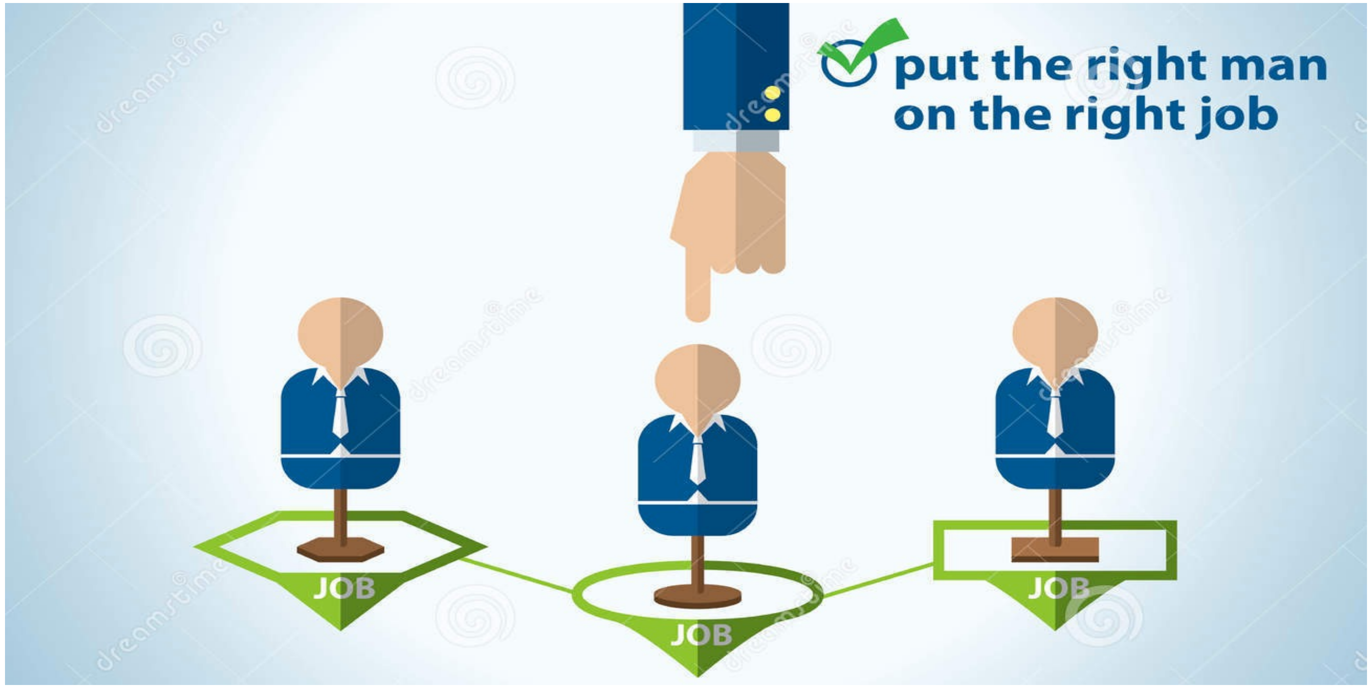
Subset	Total weight	Total value
∅	0	
{1}	7	
{2}	3	
{3}	4	
{4}	5	
{1, 2}	10	
{1, 3}	11	
{1, 4}	12	
{2, 3}	7	
{2, 4}	8	
{3, 4}	**9**	
{1, 2, 3}	14	
{1, 2, 4}	15	
{1, 3, 4}	16	
{2, 3, 4}	12	
{1, 2, 3, 4}	19	

Knapsack Problem

The Assignment Problem

- How to put the right man on the right job
- *n* people who need to be assigned to execute *n* jobs (1 person: 1 job)
- The cost (time taking or wage) of the i^{th} person when assigned to the j^{th} job is in a C[i,j] for each pair i, j = 1,2,…,n

✓ find an assignment with the <u>minimum total cost</u>



person	Design algorithm	Write/debug code	Analyze complexity	Do report & presentation
KANOKWAN	9	2	7	8
JANASPORN	6	4	3	7
NAPAT	5	8	1	8
THANAPORN	7	6	9	4

find the minimum of hours spent by each group member on each task

Amount to explore all permutations of {1,2,3,4}

<1,2,3,4> = 9+4+1+4 = 18

<1,2,4,3> = 9+4+8+9 = 30

<1,3,2,4> = 9+3+8+4 = 24

<1,3,4,2> = 9+3+8+6 = 26

: : :

: : :

Summary: Brute-force and Exhaustive search

- Applicable to a wide range of problem,
  - Easiest to design
  - Sometimes only way to some hard problems.

- Not worth time on designing efficient algorithm
  - Only a few problem instances needed to be solved.
  - Project time constraint

- Useful for small problem sizes with acceptable performance.
- Benchmark for more efficient algorithms

person	Design algorithm	Write/debug code	Analyze complexity	Do report & presentation
KANOKWAN	9	2	7	8
JANASPORN	6	4	3	7
NAPAT	5	8	1	8
THANAPORN	7	6	9	4

Practice: Find the best solution for this problem in 60 minutes

Assignment2:

for more efficient algorithm using **Hungarian method**