Brute Force & Exhuastive Search

- **Brute Force** : แก้ปัญหาโดย " ลองทุกกรณี " ตรงไปตรงมาตามนิยาม

Ex. • Sorting

↳ Bubble Sort : เปรียบเทียบและสลับค่าที่ติดกัน ; time → $O(n^2)$

Algorithm BubbleSort (A [0..n-1]) → สมาชิกตั้งแต่ 0 ถึง n-1 (A → Array)
// 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

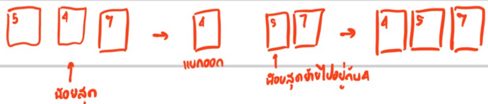
```
for i ← 0 to n-2 do                 ← จน.รอบที่จะเท่า (n-1)
    for j ← 0 to n-2-i do           ← ลูปไล่เปรียบเทียบ ; n-2-i → ขอบเขต
        if A[j+1] < A[j]            ถ้าค่าแหน่งหลัง < หน้า
            swap A[j] and A[j+1]    ให้สลับตำแหน่ง
```

Complexity

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

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

↳ Selection Sort : เลือกค่าน้อยที่สุดไว้ด้านหน้า ; time → $O(n^2)$



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

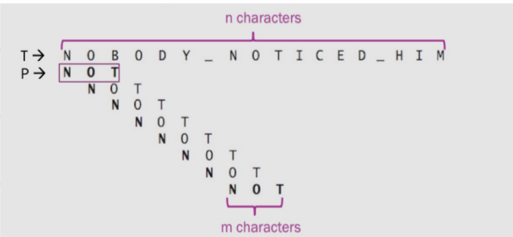
```
for i ← 0 to n-2 do          ตำแหน่ง i เอาไว้ใส่ค่าน้อยที่สุด
    min ← i                  ให้ i เป็นค่าน้อยที่สุดก่อน
    for j ← i +1 to n-1 do   ไล่ตรวจค่าน้อยที่สุดทีละตำแหน่ง
        if A[j] < A[min]     } เปรียบเทียบค่า เมื่อเจอค่าที่
            min ← j          } น้อยกว่า → อัพเดท min
    swap A[i] and A[min]     เอาค่าที่น้อยที่สุดสลับไว้ที่ i
```

Complexity

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

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

- **String Matching** : เลื่อน pattern เปรียบเทียบ text ทีละตำแหน่ง ; time → $O(nm)$



Algorithm BruteForceStringMatching

1: Input: Pattern $P = p_1p_2p_3\cdots p_m$ and Text $T = t_1t_2t_3\cdots t_n$, $n \geq m$
2: Output: Index of the 1st character in the text that starts a matching
3: substring, and −1 for the unsuccessful search
4:
5: **for** $i := 1$ to [n-m+1] **do** — n-(m-1) = n-m+1 ; i = ตำแหน่งเริ่มต้น text
6: $j \leftarrow 1$ — ไล่ pattern
7: **while** $j \leq m$ and [$p_j == t_{\{i+j-1\}}$] **do** — เปรียบเทียบตัวอักษร pattern กับ text ว่าตัวเดียวกันไหม ; ถ้ายังมีตัวอักษรให้เปรียบเทียบ
8: $j \leftarrow j+1$ — ไล่ตัวถัดไป
9: **if** $j == m+1$ **return** i — ถ้า j วิ่งเกิน m → pattern ✓
10: **return** −1 — ไม่พบ pattern

```
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] range(n-m+1)
        j = 0
        while j < m and [fill code here] P[j] == T[i+j]:
            j += 1
        if j == m:
            return i
    return -1
```

- ตำแหน่งเริ่ม Text (i)
- range(n-m+1) ; ใช้ได้แค่ n-m ตัว เพราะ Pattern m ตัว
- P[j] == T[i+j]: ; เทียบทีละตัวอักษรทีละตัว
  - P[j] → ตัวที่ j ของ Pattern
  - T[i+j] → ตัวใน text เริ่มจาก i

- **Closest-Pair Problem** : คำนวณระยะห่างทุกคู่ของจุด เพื่อหาคู่ที่ใกล้ที่สุด

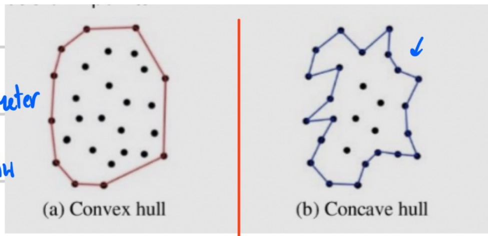
ALGORITHM *BruteForceClosestPair(P)*
//Finds distance between two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $p_1(x_1, y_1), \ldots, p_n(x_n, y_n)$
//Output: The distance between the closest pair of points
$d \leftarrow \infty$ — ตั้งค่าเริ่มต้นเป็น ∞
for $i \leftarrow 1$ **to** $n-1$ **do** — เลือกจุดที่ i ไปเรื่อยถึง (n-1)
  for $j \leftarrow i+1$ **to** n **do** — เทียบจุดต่อไป
    $d \leftarrow \min(d, sqrt((x_i - x_j)^2 + (y_i - y_j)^2))$ //*sqrt* is square root — เปรียบเทียบค่าที่น้อยที่สุด เก็บใน d
return d

↳ สูตรคำนวณระยะทาง Euclidean

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

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

- Convex - Hull Problem : สร้างขอบนอกสุดของกลุ่มจุด



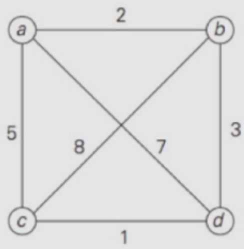
	Convex hull	Concave hull
ลักษณะ	ไม่มีส่วนเว้า	มีส่วนเว้า
	ไม่ขึ้นกับ parameter	รูปร่างละเอียดกว่า
	เป็นเส้นตรงต่อกัน	ขึ้นกับ parameter
ข้อดี	คำนวณง่าย, เร็ว, มีคำตอบเดียว	แสดงรูปทรงได้ดีกว่า
ข้อจำกัด	ข้อมูลมี Error	ซับซ้อน, มีหลายคำตอบ

- Exhaustive (เหมือน) Search : ไล่ตรวจทุกความเป็นไปได้ของปัญหา → จัดเรียง → เลือกคำตอบที่ถูกที่สุด

Ex. • Traveling Salesman Problem : ลองทุกทางที่เป็นไปได้ → เลือกทางที่สุด

! ไม่เหมือน minimum spanning tree! เพราะเป็นการหาแบบ Hamiltonian circuit โดยจะต้องครบวงจร!

Traveling Salesman Problem



Tour	Length
a → b → c → d → a	18
a → b → d → c → a	11
a → c → b → d → a	23
a → c → d → b → a	11
a → d → b → c → a	23
a → d → c → b → a	18

(11, 11) } Answer!

- Knapsack Problem : สร้างทุก subset ของสิ่งของ → เลือกชุด นน. ไม่เกิน + มูลค่า ↑

Subset	Total weight	Total value
∅	0	0
{1}	7	42
{2}	3	12
{3}	4	40
{4}	5	25
{1, 2}	10	54
{1, 3}	11	-
{1, 4}	12	-
{2, 3}	7	52
{2, 4}	8	37
{3, 4}	9	65
{1, 2, 3}	14	-
{1, 2, 4}	15	-
{1, 3, 4}	16	-
{2, 3, 4}	12	-
{1, 2, 3, 4}	19	-

Knapsack Problem

- Assignment Problem : ลองทุก permutation → เลือกผลรวมต้นทุนต่ำที่สุด

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 hour 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
: : :
: : :

ข้อดี : ใช้ได้กับปัญหาหลากหลาย
เป็น baseline

ข้อจำกัด : ช้า เมื่อปัญหาขนาดใหญ่