Analysis of Algorithms: the Analysis Framework

- it is logical to investigate an algorithm’s efficiency as “a function of some parameter ***n*** indicating **the algorithm’s input size**”

let n be an input size

$$\left.\begin{aligned} t(n) &= 2n \\ t(n) &= n^2+2n+1 \end{aligned}\right\}$$

เขียนอธิบาย running time ของ algm ในรูปแบบ ฟังก์ชันของ n

Analysis of Algorithms: the Analysis Framework

Time Efficiency (2)

Example: the problem of searching for the maximum element in a list

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

max = A[0];

For (i = 1; i < n; i++) do

    if (**max < A[i]**) max = A[i];

$$\sum_{i=1}^{n-1} (1) = ?$$

- basic operation: key comparison inside loop for

$$\sum_{i=1}^{n-1} (1) = \underbrace{1 + 1 + 1 + \dots + 1}_{(n - 1 - \cancel{1} + \cancel{1}) \text{ ครั้ง}}$$

$$= (n-1) \times 1 = n - 1$$

$$C(n) = \sum_{i=1}^{n-1} (1) = n - 1 \quad f(n)$$

Analysis of Algorithms: the Analysis Framework

Time Efficiency (3)

Example: the problem of sorting all elements in a list

```
Algorithm BubbleSort(A[0..n-1])
For (i = 0; i < n-1; i++)
        For (j = 0; j < n – i – 1; j++)
                if (A[j] > A[j+1])
                        Swap(A[j], A[j+1]);
```

$$\sum_{i=0}^{n-2} \sum_{j=0}^{n-i-1-1} (1) = ?$$

- basic operations: key comparison inside the inner loop for

  swap operation inside the inner loop for

$$T(n) \simeq C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-i-1-1} (1)$$

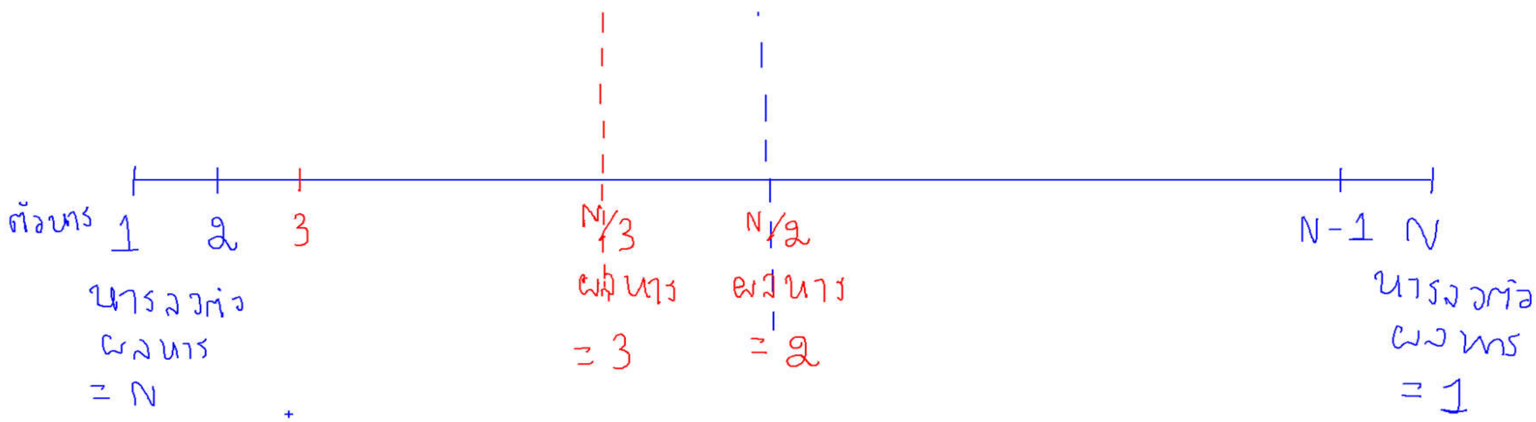
$$\sum_{i=0}^{n-2} \sum_{j=0}^{n-i-2} (1) = \sum_{i=0}^{n-2} \underbrace{1 + 1 + \ldots + 1}_{\text{มี } (n-i-2-0+1) \text{ ตัว} \; = n-i-1} \qquad \in O(n^2)$$

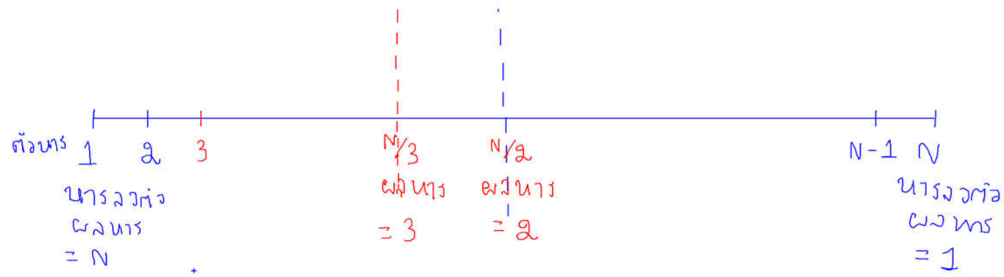
$$= \sum_{i=0}^{n-2} (n-i-1)$$

$$= \underset{i=0}{(n-1)} + \underset{i=1}{(n-2)} + \underset{i=2}{(n-3)} + \ldots + \underset{i=n-2}{(\cancel{n} - \cancel{n} + \cancel{2} - \cancel{1})}^{\;1}$$

$$C(n) = 1 + 2 + 3 + \ldots + n - 1$$

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


ตัวหาร
1
2
3
N/3
N/2
N-1
N
หารลงตัว
ผลหาร
= N
ผลหาร
= 3
ผลหาร
= 2
หารลงตัว
ผลหาร
= 1



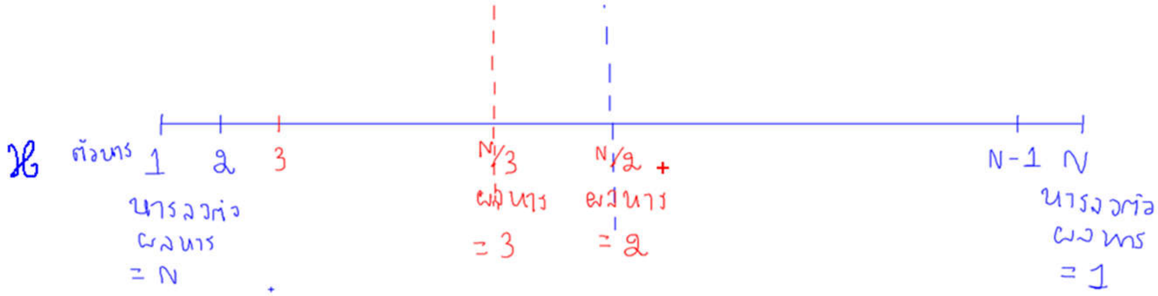
N หาร 3 ไม่ลงตัว bound ของตัวหาร ไม่เกิน $N/3$

N หาร 4 ไม่ลงตัว bound ของตัวหาร ไม่เกิน $N/4$

⋮

$\boxed{\text{N หาร } x \text{ ไม่ลงตัว bound ของตัวหาร ไม่เกิน } N/x}$

ถ้ามีตัวหาร เกิน N/x ซึ่งหาร N ลงตัว ผลลัพธ์ก็จะน้อยกว่า x



N หาร 3 ไม่ลงตัว bound ของตัวหาร ไม่เกิน $N/3$

N หาร 4 ไม่ลงตัว bound ของตัวหาร ไม่เกิน $N/4$

⋮

N หาร x ไม่ลงตัว bound ของตัวหาร ไม่เกิน N/x

ถ้ามีตัวหาร เกิน N/x ซึ่งหาร N ลงตัว ผลลัพธ์ที่ได้จะน้อยกว่า x

$$\therefore \quad x \le N/x$$

$$x^2 \le N$$

$$x \le \sqrt{N}$$

```
for (x = 2; x <= √N ; x++)
// ลองหาร N/x ลงตัว?
```

กล่องนักเก็ต 6 ชิ้น, 9 ชิ้น, 20 ชิ้น

Let 'Nugget' be the set of Nugget Numbers.

Basis step : $6, 9, 20 \in \text{Nugget}$

Recursive step : if $x \in \text{Nugget}$

then $x + 6 \in \text{Nugget}$

$x + 9 \in \text{Nugget}$

$x + 20 \in \text{Nugget}$

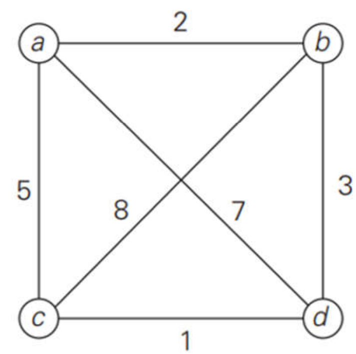
Exhaustive Search: Traveling Salesman Problem (2)

$G = (V, E)$

$V = \{a, b, c, d\}$

$v_1 \overset{w}{\text{———}} v_2$: มีถนนเชื่อมเมือง v_1 กับ v_2

w : distance ระหว่าง เมือง v_1 กับ v_2



Tour	Length	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	optimal
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	optimal
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

Perm ("abc")

- a + perm("bc")
  - b + perm("c") → abc
  - c + perm("b") → acb
- b + Perm("ac")
  - a + perm("c") → bac
  - c + perm("a") → bca
- c + perm("ba")
  - b + perm("a") → cba
  - a + perm("b") → cab

$3! = 3 \times 2 \times 1 = 6$ permutations

Exhaustive Search: Assignment Problem

- There are *n* people who need to be assigned to execute *n* jobs, one person per job. (That is, each person is assigned to exactly one job and each job is assigned to exactly one person.)
- The cost that would accrue if the i^{th} person is assigned to the j^{th} job is a known quantity $C[i, j]$ for each pair $i, j = 1, 2, \ldots, n$.
- The problem is to find an assignment with the minimum total cost.

	Job 1	Job 2	Job 3	Job 4	total cost (฿)
Permutation #1	P1	P2	P3	P4	
#2	P1	P2	P4	P3	
#3	P1	P3	P2	P4	
⋮		⋮			

Subset	Total weight	Total value
$\varnothing$	0	$ 0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1, 2}	10	$54
{1, 3}	11	not feasible
{1, 4}	12	not feasible
{2, 3}	7	$52
{2, 4}	8	$37
{3, 4}	**9**	**$65**
{1, 2, 3}	14	not feasible
{1, 2, 4}	15	not feasible
{1, 3, 4}	16	not feasible
{2, 3, 4}	12	not feasible
{1, 2, 3, 4}	19	not feasible

	#4	#3	#2	#1	**Subset**	**Total weight**	**Total value**
0	0	0	0	0	∅	0	$ 0
1	0	0	0	1	{1}	7	$42
2	0	0	1	0	{2}	3	$12
4	0	1	0	0	{3}	4	$40
8	1	0	0	0	{4}	5	$25
3	0	0	1	1	{1, 2}	10	$54
5	0	1	0	1	{1, 3}	11	not feasible
9	1	0	0	1	{1, 4}	12	not feasible
6	0	1	1	0	{2, 3}	7	$52
10	1	0	1	0	{2, 4}	8	$37
12	1	1	0	0	**{3, 4}**	**9**	**$65**
7	0	1	1	1	{1, 2, 3}	14	not feasible
11	1	0	1	1	{1, 2, 4}	15	not feasible
13	1	1	0	1	{1, 3, 4}	16	not feasible
14	1	1	1	0	{2, 3, 4}	12	not feasible
15	1	1	1	1	{1, 2, 3, 4}	19	not feasible

```
using namespace std;

#define N 4  // number of items

int main() {

    //for (int i=0; i<16; i++)
    for (int i=0; i < (1 << N); i++)b
        add_bitstring_to_vector(i);

    return 0;
} // end main()
```

left shift

				1	1
			1	0	2
		1	0	0	4
	1	0	0	0	8
1	0	0	0	0	16

```
void add_bitstring_to_vector(int value) {
    char buf[N+1];
    char *p = buf;

    for (int i=0; i<N; i++) {
        if (value & (1 << i))
            *p = '1';
        else
            *p = '0';
        p++;
    } //end for
    *p = '\0';
    result.push_back(string(buf));
} // end add_bitstring_to_vector()
```

buf $\boxed{0|0|0|0}$

value = 0 = 0000

$$\begin{array}{r} 0000 \\ \& \ 0001 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0000 \\ \& \ 0010 \\ \hline 0000 \end{array} \qquad \begin{array}{r} 0000 \\ \& \ 0100 \\ \hline 0000 \end{array} \qquad \begin{array}{r} 0000 \\ 1000 \\ \hline 0000 \end{array}$$

```c
void add_bitstring_to_vector(int value) {
    char buf[N+1];
    char *p = buf;

    for (int i=0; i<N; i++) {
        if (value & (1 << i))
            *p = '1';
        else
            *p = '0';
        p++;
    } //end for
    *p = '\0';
```

buf $\boxed{1|0|0|0}$

value = 1 = 0001

0001 &	0001 &	0001	0001
0001	0010	0100	1000
0001	0000	0000	0000