OOP & data struct

9. Big O notation

BY SOMSIN THONGKRAIRAT


N LOG N
COMPUTER
N^2
imgflip.com

Select number game (เกมเลือกเลข)

Given n positive integer (sorted), select individual a b c d that maximize (((a*b)+c)-d)

กำหนดเลขจำนวนเต็มบวก ให้ n จำนวน (เรียงมาให้แล้ว) เลือก a b c d (ห้ามเป็นตัวเดียวกัน)ที่ทำให้สมการ (((a*b)+c)-d) มีค่ามากที่สุด

$\{12,11,10,4,2,1,0\} = ?$

$\{9,8,7,6\} = ?$

Method 1 Brute force

find all combination / ค้นหาทุกกรณี

Running time = ?

a=12 b=11 c=10 d=4
a=12 b=11 c=10 d=2
a=12 b=11 c=10 d=1
a=12 b=11 c=10 d=0
a=12 b=11 c=4 d=10
a=12 b=11 c=4 d=2
a=12 b=11 c=4 d=1
a=12 b=11 c=4 d=0
a=12 b=11 c=2 d=10
a=12 b=11 c=4 d=4
.....

```
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        for(int k=0;k<n;k++){
            for(int l=0;l<n;l++){
```

≈ N^4

More advance trick

Maximize -> (((a*b)+c)-d)

D have to be as less as possible

D ต้องเป็นตัวเลขที่ต่ำที่สุดที่จะเป็นไปได้

Method 2 Brute force (fixed minimum)

find just a,b,c using last number (minimum) as d

ค้นหา a,b,c และใช้ตัวสุดท้าย (น้อยที่สุด) เป็น d

Runnung time = ?

a=12 b=11 c=10 d=0
a=12 b=11 c=4 d=0
a=12 b=11 c=2 d=0
a=12 b=11 c=1 d=0
a=12 b=10 c=11 d=0
a=12 b=10 c=4 d=0
a=12 b=10 c=2 d=0
...

```
for(int i=0;i<n-1;i++){
    for(int j=0;j<n-1;j++){
        for(int k=0;k<n-1;k++){
```

Method 3 Greedy (reasonable?)

Select 3 left most number as a b c and last as d

เลือก 3 ตัวแรกเป็น a b c และตัวสุดท้ายให้เป็น d

Runnung time = ?

a=12 b=11 c=10 d=0

```
return ((num[0] * num[1]) + num[2]) - num[n-1];
```

Which one you like? And Why?

ชอบอันไหน? และทำไม?

Method 1 Brute force

Method 2 Brute force (fixed minimum)

Method 3 Greedy (reasonable?)

Which one fastest? / อันไหนเร็วสุด

Asymptotic Notation

- describes the limiting time (iteration) an algorithm takes with a given input, n.

- การพิจารณาขีดจำกัดด้านเวลา (จำนวนรอบ) ที่โปรแกรมนั้นๆ ทำงาน จากการป้อน n เป็น input

Asymptotic Notation

big O (O) -> worst-case running time (iteration).

big Theta -> (Θ) -> Constance running time (iteration).

big Omega (Ω) -> best-case running time (iteration).

big O (O) -> เวลา(รอบ) ที่ใช้ในกรณีที่แย่ที่สุด

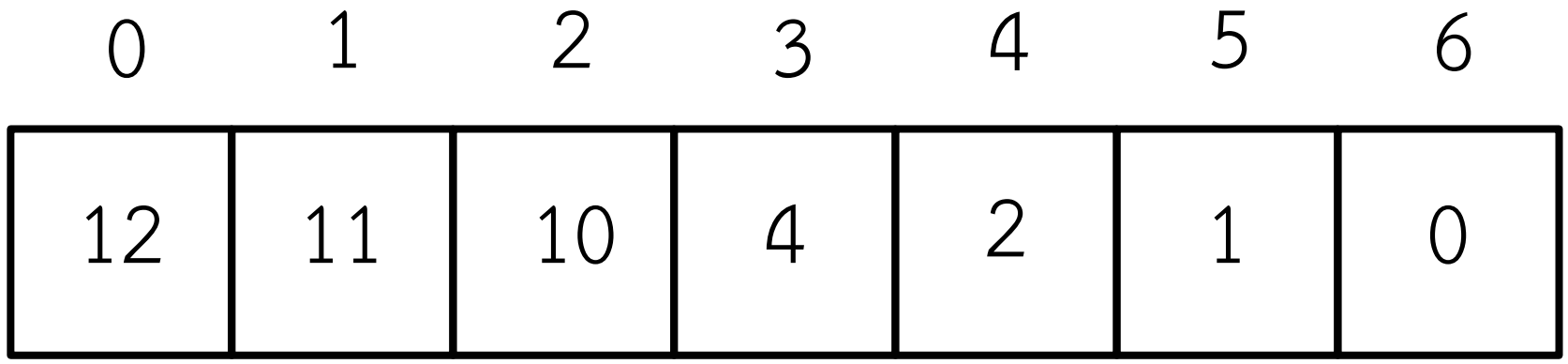
big Theta -> (Θ) -> เวลา(รอบ) ที่ใช้ในทุกกรณี

big Omega (Ω) -> เวลา(รอบ) ที่ใช้ในกรณีที่ดี

Example (linear search)

- find wanted object in array from 0 to n if found return

- หา object ที่ต้องการจาก 1 ถึง n หากพบให้ return ทันที



Example (linear search)

```
for(int i=0;i<n;i++){
    if(num[i] == target){
        return i;
    }
}
```

linear search running time ?

```
int number1[7] = {12,11,10,4,2,1,0} , n1 = 7;

 linear_search(number1,n1,1)   ?
 linear_search(number1,n1,12)  ?
 linear_search(number1,n1,999) ?
```

Iteration = ?

linear search

big O (worst-case iteration) = ?

big Theta -> (**Θ**) (Constance iteration) = ?

big Omega (**Ω**) -> (best-case iteration) = ?

In function of n

Asymptotic Notation

big O (worst-case iteration) is most common method to determine algorithm performance because it’s covered all condition

big O (กรณีที่แย่ที่สุด) นิยมที่สุดในการใช้พิจารณาประสิทธิภาพของ algorithm เนื่องจาก ครอบคลุมทุกกรณี

Back to select game (Actual running time)

Method 1 Brute force

= 6*n^4 (check redundance index) +

3*n^4 (calculate result) +

1*n^4 (compare)

= 10n^4 iteration

Actual running time

Method 1 = 10n^4 iteration

Method 2 = 7n^3

Method 3 = 3

How bad it that? / มันจะแย่ได้ขนาดไหนกัน?

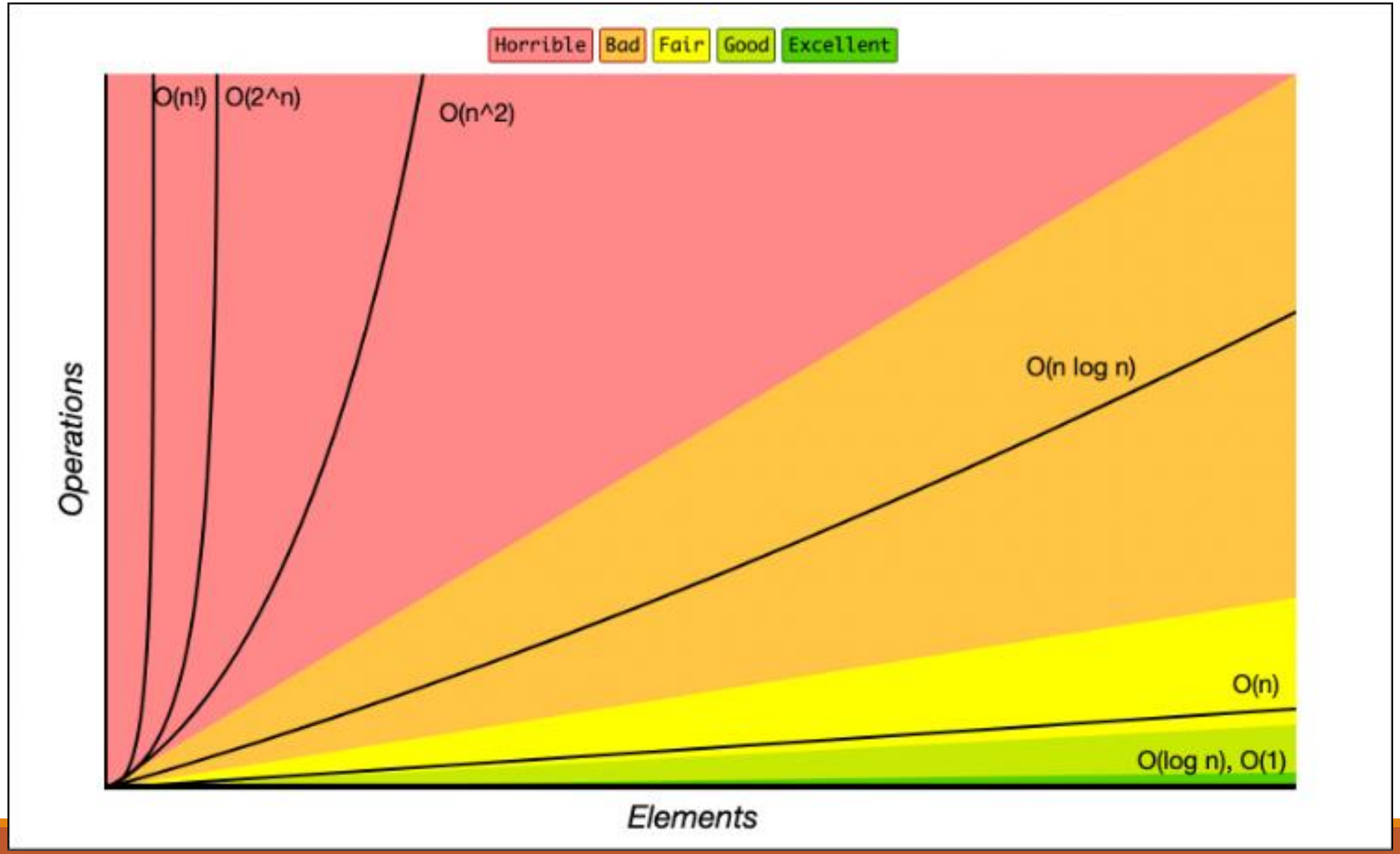
O(1) – Excellent / จัดไป

O(log n) – Good / ดี

O(n) – Fair / พอใช้

O(n log n) – Bad / ถ้าจำเป็น

O(n^2), O(2^n) and O(n!) - Horrible/Worst / no!


Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Dominating (which term are dominating)

Method 1= 10n^4 iteration

Method 2 = 7n^3

Method 3 = 3

this is enough to show domination

O(1)

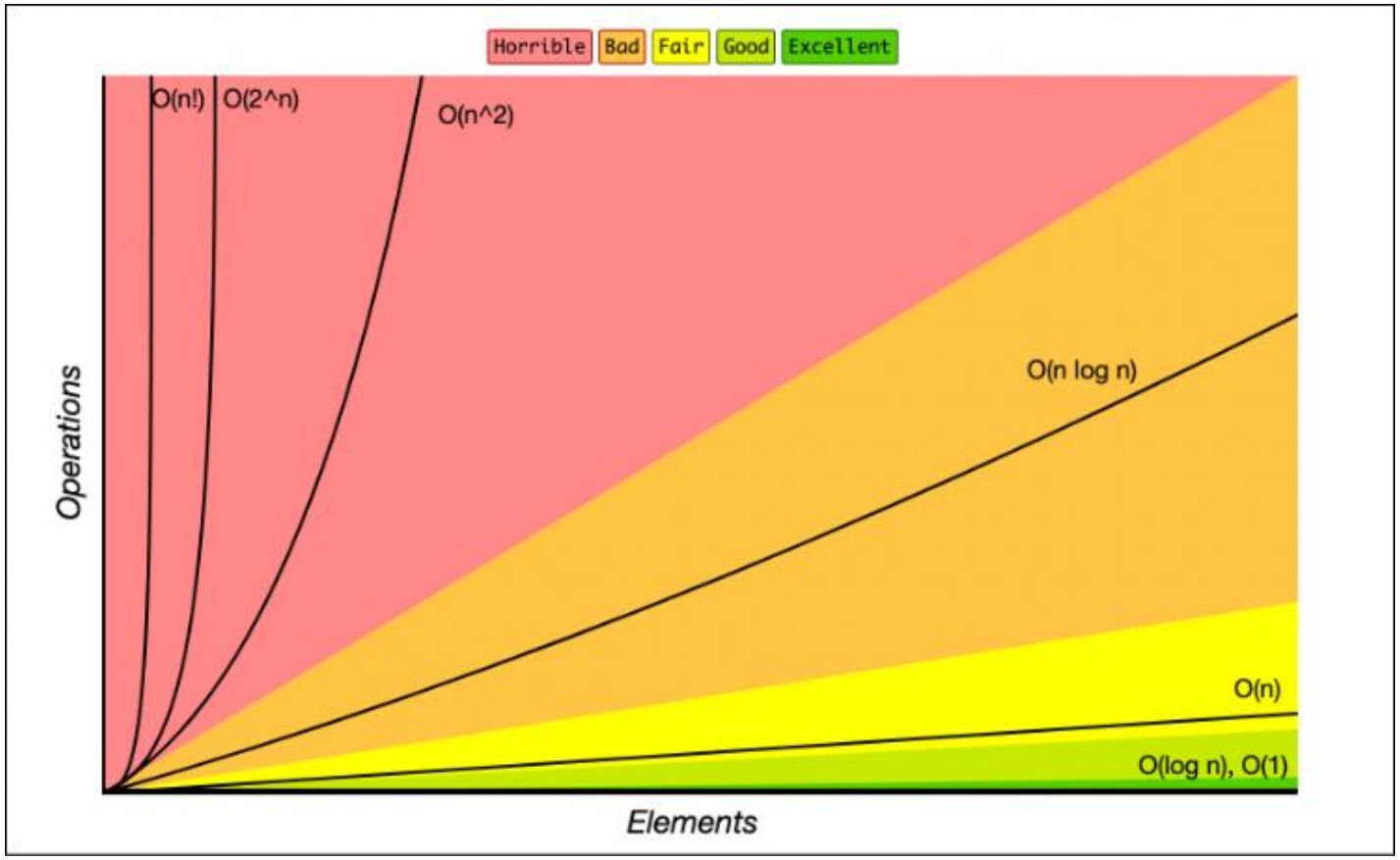
O(log n)

O(n)

O(n log n)

O(n^2)

O(2^n)



Big O

Method 1 Brute force = n^4

Method 2 Brute force (fixed minimum) = n^3

Method 3 Greedy (reasonable?) = 1

We don’t care constant and non-dominate term

ไม่สนใจค่าคงที่ และ พจน์ที่ไม่ dominate

See experiment

```
int number3[1000] = {10000,9990,9980,9970,9970...}, n3 = 1000;
```

```
greedy_select(number3,n3) = 1 iteration
fix_minmum_select(number3,n3) = 7*10^9 (7 พันล้าน)
brute_fource_select(number3,n3) = too much to wait
```

Big O feeling

Method 3 Greedy (reasonable?) = immediately ทันที

Method 2 Brute force (fixed minimum) = can wait รอได้

Method 1 Brute force = T T

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

Another example

- check prime number?

- dictionary algorithm (binary search)?

Prime number

```
bool is_prime(int n){
    for(int i=2;i<n;i++){
        if(n%i==0){
            return false;
        }
    }
    return true;
}
```

n	iteration
2	1
77	?
1024	?
179424673	?
Any prime	?
Any factor of 2	?

n	iteration	Dominate term
2	1	1
77	7	7
1024	1	1
179424673	179424673	n
Any prime	n	n
Any factor of 2	1	1

Prime number

big O (worst-case iteration) = n

big Theta -> (Θ) (Constance iteration) = ?

big Omega (Ω) -> (best-case iteration) = 1

In function of n -> f(n)

Prime number (improve)

```
bool is_prime2(int n){
    for(int i=2;i<n/2;i++){
        if(n%i==0){
            return false;
        }
    }
    return true;
}
```

n	iteration
2	1
77	7
1024	1
179424673	89712336
1021	510
2069	1034
Any prime	n/2
Any factor of 2	1

n	iteration	Dominate term
2	1	1
77	7	n
1024	1	1
179424673	89712336	n
1021	510	n
2069	1034	n
Any prime	n/2	n
Any factor of 2	1	1

Prime number (improve)

big O (worst-case iteration) = n

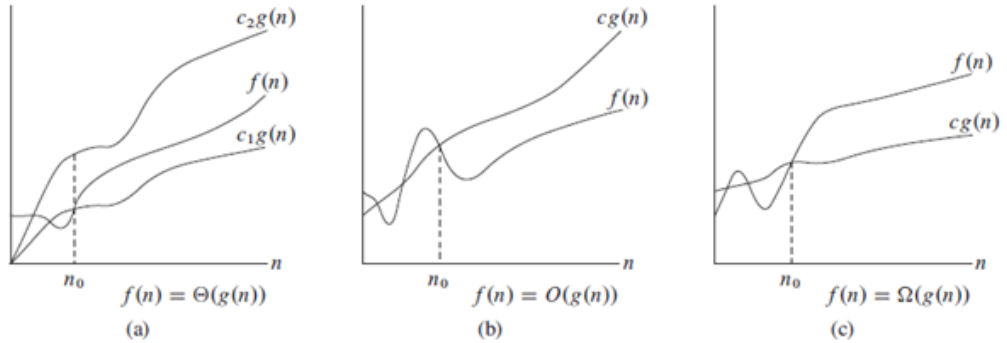
big Theta -> (Θ) (Constance iteration) = ?

big Omega (Ω) -> (best-case iteration) = 1

In function of n -> f(n)

Why O(is_prime) = O(is_prime2)

- Mathematically -> Asymptotic Notation



Boring

- Programmatically -> same performance
- Feeling -> same

Why O(is_prime) = O(is_prime2)

- long-term growth rate of functions effect running time more more than absolute magnitudes

- growth rate ของฟังก์ชันในระยะยาวส่งผลกระทบมากกว่าค่าจริงๆ ของฟังก์ชัน

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

Try running

```
cout << is_prime(179424673) << endl;
cout << is_prime(373587883) << endl;
cout << is_prime(776531401) << endl;
cout << is_prime(2038074743) << endl;
cout << is_prime(11111111111) << endl;
cout << is_prime(10000000000) << endl;
```

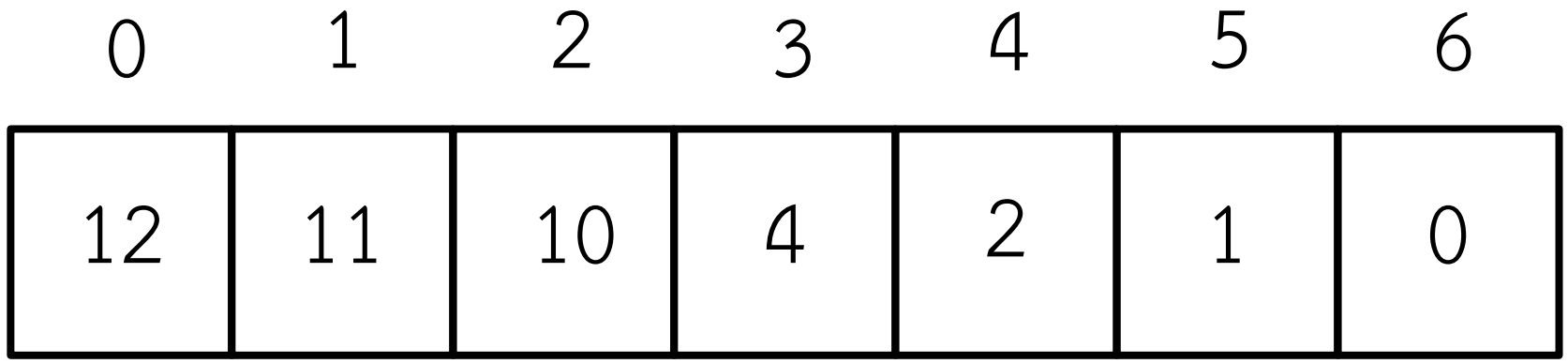
```
cout << is_prime2(179424673) << endl;
cout << is_prime2(373587883) << endl;
cout << is_prime2(776531401) << endl;
cout << is_prime2(2038074743) << endl;
cout << is_prime2(11111111111) << endl;
cout << is_prime2(10000000000) << endl;
```

Feel difference ?

Searching in dictionary (binary search)

- find wanted object in sorted array from 0 to n if found return

- หา object ที่ต้องการจาก 1 ถึง n ใน array ที่เรียงเลขไว้แล้วหากพบให้ return ทันที



Searching in dictionary (binary search)

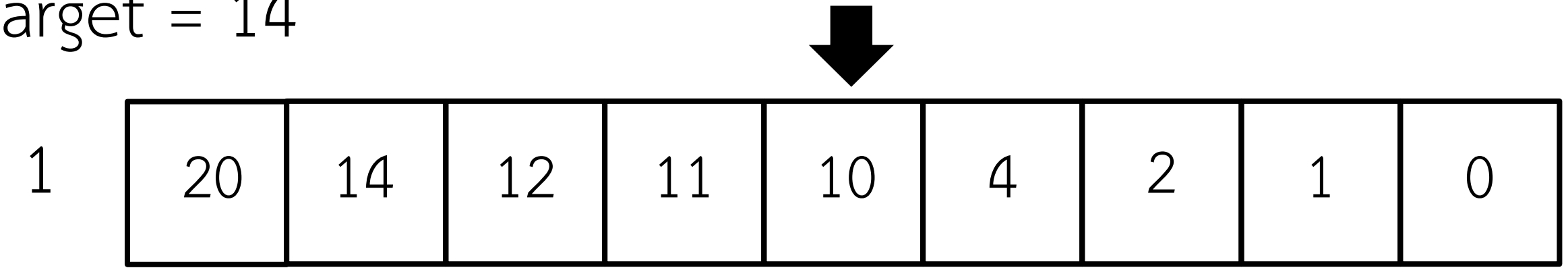
- go to middle of array / ไปยังตรงกลางของ array

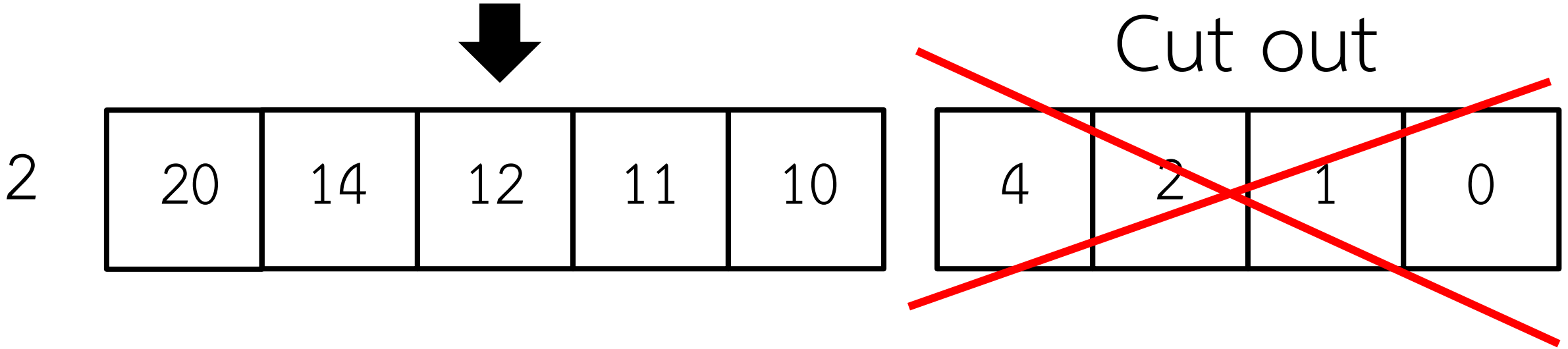
- if found target return / หากพบเป้าหมาย return

- if middle item greater than target then cut right array out otherwise cur left out

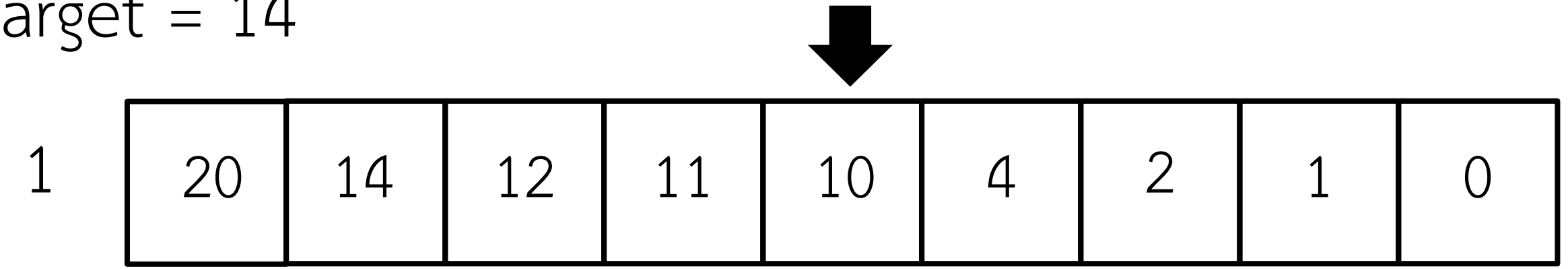
- หาก item ที่พบมากกว่าเป้าหมาย ให้ตัด array ที่อยู่ด้านขวาทิ้ง หากไม่ใช้ตัดด้านซ้ายทิ้ง

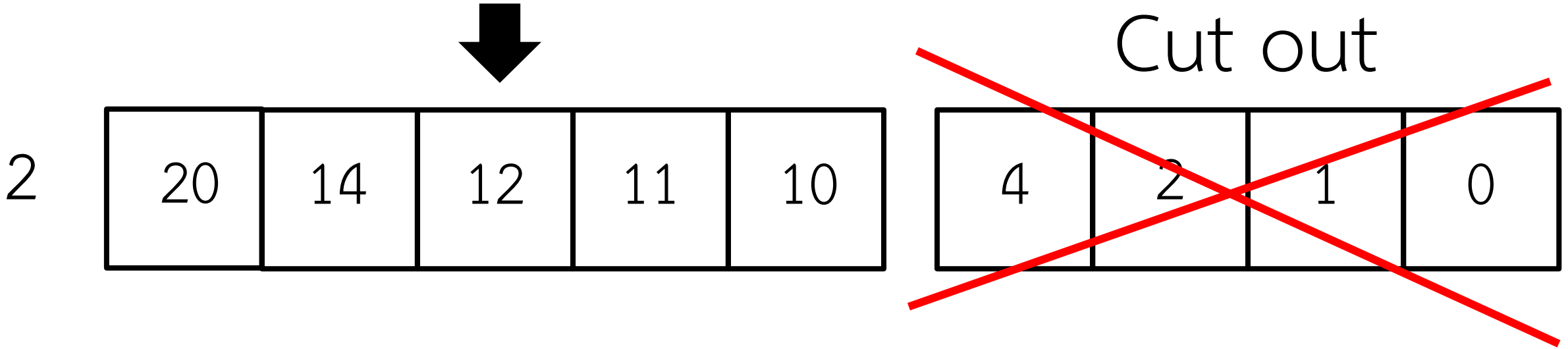
Target = 14



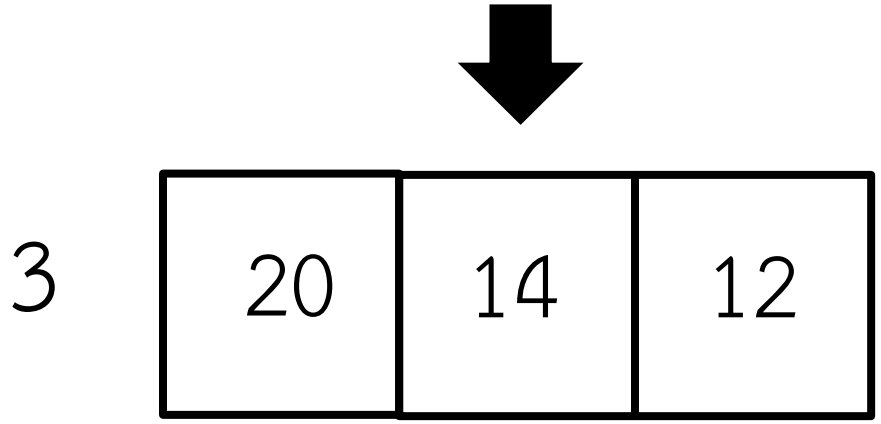


Target = 14

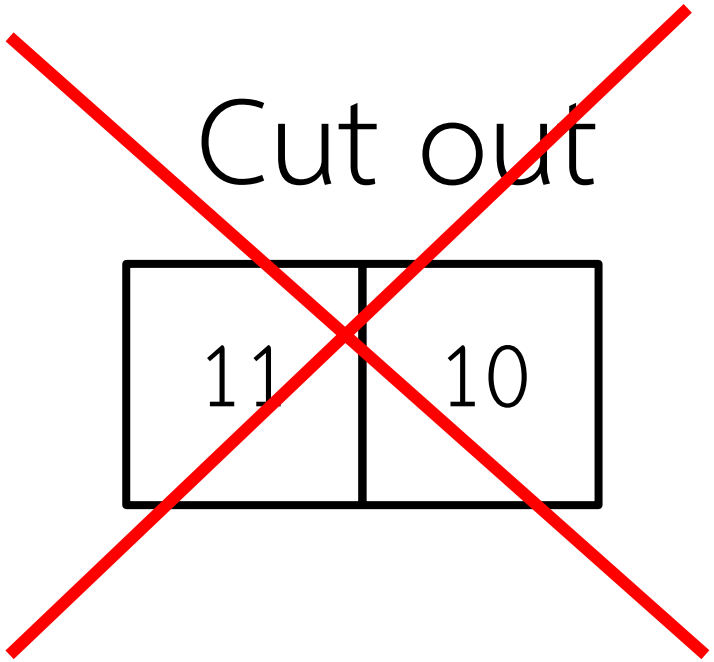




Target = 14



Found!



```
int b_search(int num[],int n,int target){
    int lo = 0, hi = n-1;

    while(hi > lo){
        int mid = (lo + hi) / 2;
        if(target == num[mid]){
            return mid;
        }
        else if(num[mid] > target){
            lo = mid + 1;
        }
        else{
            hi = mid - 1;
        }
    }
    if(target == num[lo]){
        return lo;
    }
    return -1;
}
```

```
int number1[9] = {20,14,12,11,10,4,2,1,0} , n1 = 9;
int number2[8] = {12,11,10,4,2,1} , n2 = 8;
int number3[4] = {12,11,10,4} , n3 = 4;

cout << linear_search(number1,n1,12) << endl;
cout << b_search(number1,n1,12) << endl;
```

Find 18

20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5

4 iteration

Worse case

n	Worse case	best case
2	1	1
8	3	1
32	5	1
256	8	1
1024	10	1
8192	13	1
n	$log_2 n$	1

binary search

big O (worst-case iteration) = log n

big Theta -> (Θ) (Constance iteration) = ?

big Omega (Ω) -> (best-case iteration) = 1

In function of n -> f(n)

Common word

Complexity -> Big O notation

Running time -> actual iteration

Conclude

- how to measure algorithm performance
- big O
- example for some algorithm

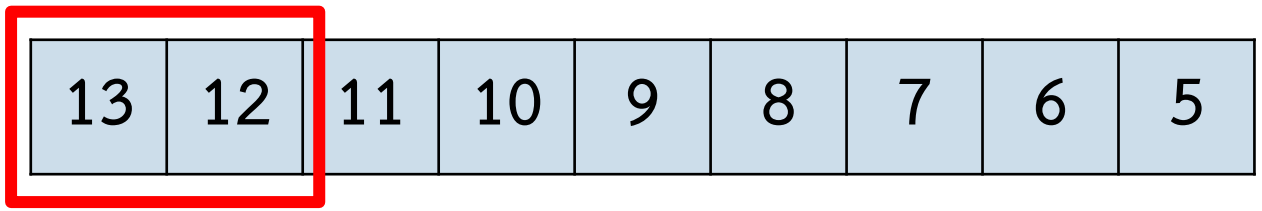
Lab

Bubble sort

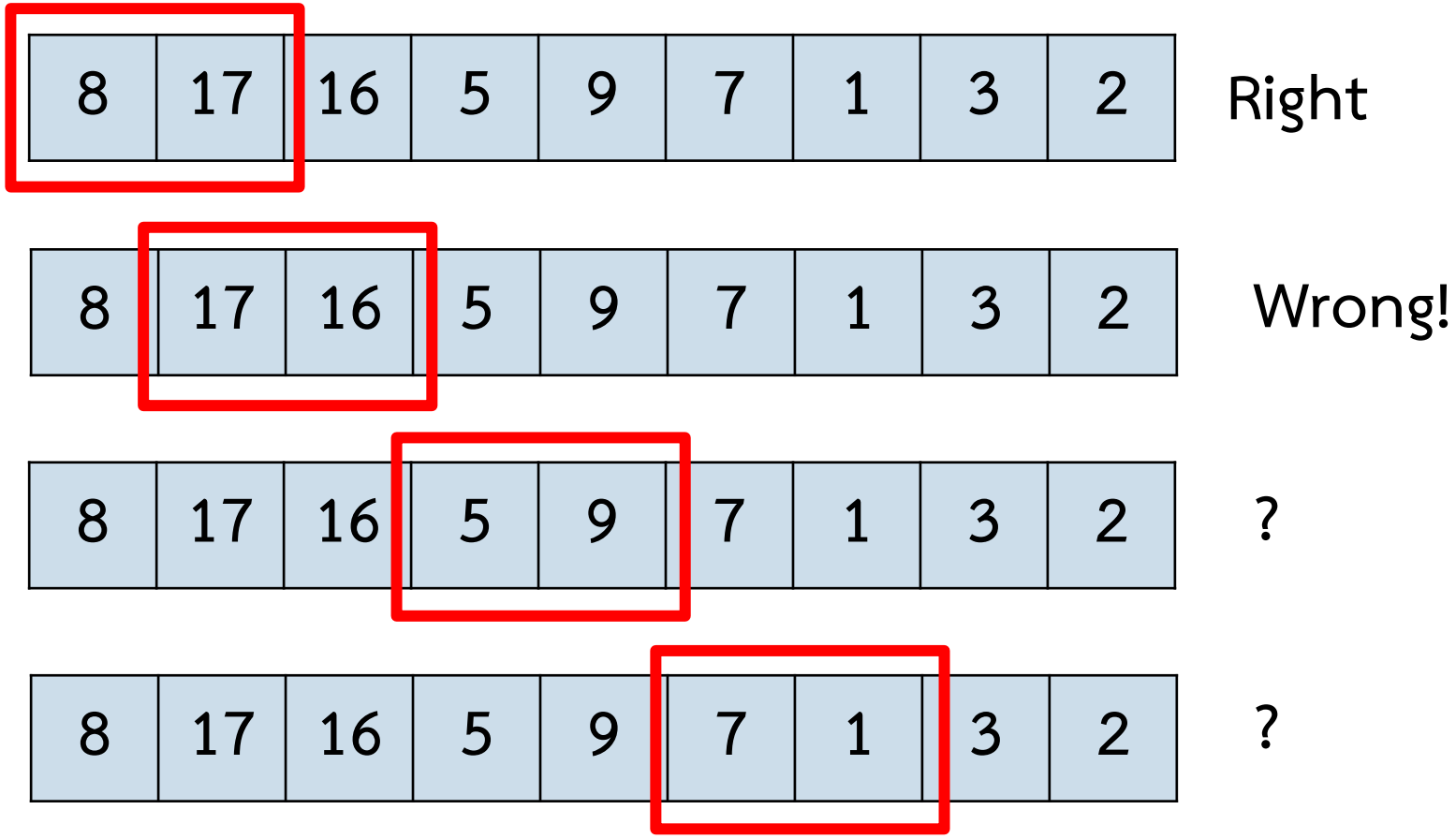
- simple sorting algorithm / วิธีการเรียงเลขอย่างง่าย

- check 2 adjacent number if they are in the wrong place swap it

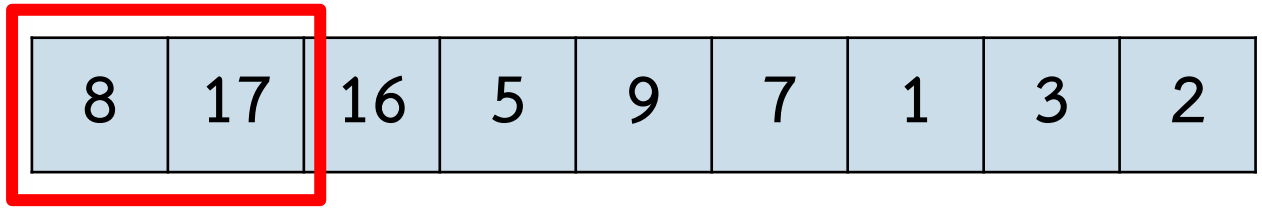
- ตรวจสอบตัวเลข 2 ตัวที่อยู่ติดกัน ถ้าอยู่ผิดตำแหน่งให้สลับกัน



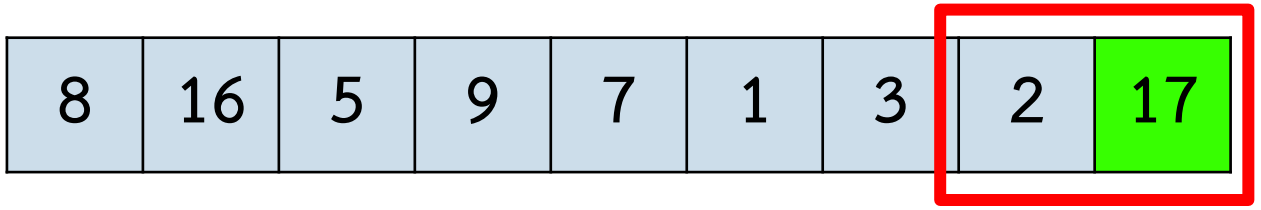
Example (increasing order) น้อยไปมาก



sequence



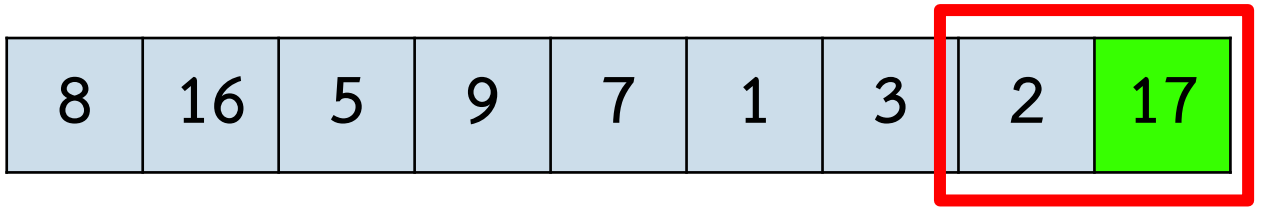
- if we check and swap all number, the highest number will be in the right place
- หากเราตรวจสอบและสลับตัวเลขทุกตัว เลขสูงที่สุดจะอยู่ถูกที่



Right?

iteration

- check all number and swap take n iteration
- เช็คและสลับทุกตัวเลขใช้ทั้งหมด n รอบ
- repeat whole process for n time / ทำทุกขั้นตอน อีก n ครั้ง




8
17
16
5
9
7
1
3
2

code

```
// normal bubble sort
for(int i=0;i<n - 1;i++){
    for(int j=0;j<(n-i)-1;j++){
        m1++;
        if(number1[j] > number1[j + 1]){
            int tmp = number1[j];
            number1[j] = number1[j+1];
            number1[j+1] = tmp;
        }
    }
}
```

Several modification of bubble sort

normal bubble sort

n*n bubble sort

limited bubble sort

- normal bubble sort run n-1 , n-2 , n-3 ... 1

- n*n bubble sort run n-1 , n-1 , n-1 , n-1

- limited bubble sort check if in 1 n iteration if no swap return (finish sorting)

- limited bubble sort เช็คว่าใน 1 รอบมีการสลับที่หรือไม่ถ้าไม่มีแปลว่าเรียงเสร็จแล้วให้จบการทำงานเลย