Data Structures and Algorithms

(ESO207)

Lecture 40

- Search data structure for integers : Hashing

Data structures for searching

in $O(1)$ time

Motivating Example

Input: a given set S of 1009 positive integers

Aim: Data structure for **searching**

Example

{

123, 579236, 1072664, 770832456778, 61784523, 100004503210023, $\boxed{19}$,
762354723763099, 579, 72664, $\boxed{977083245677001238}$, 84, 100004503210023,

...

}

Data structure : Array storing S in sorted order

Searching : Binary search

O(log $|S|$) time

Can we perform search in **O**(1) time ?

Problem Description

$\boldsymbol{U}$:

$\boldsymbol{S} \subseteq \boldsymbol{U}$,

$\boldsymbol{n} = |\boldsymbol{S}|$, $\boldsymbol{n} \ll \boldsymbol{m}$

A search query:

Aim: A data structure for a given set $\boldsymbol{S}$

that can facilitate search in **O**($\mathbf{1}$) time

A trivial data structure for O(1) search time

Build a 0-1 array **A** of size m such that

A[i] = 1 **if** $i \in S$.

A[i] = 0 **if** $i \notin S$.

Time complexity for searching an element in set S **: O(1).**

	0	**1**	**2**	**3**	**4**	**…**				**…**			**m-1**
A	0	0	1	0	0	…	0	1	0	…	0	0	1

This is a totally Impractical data structure because $n \ll m$!

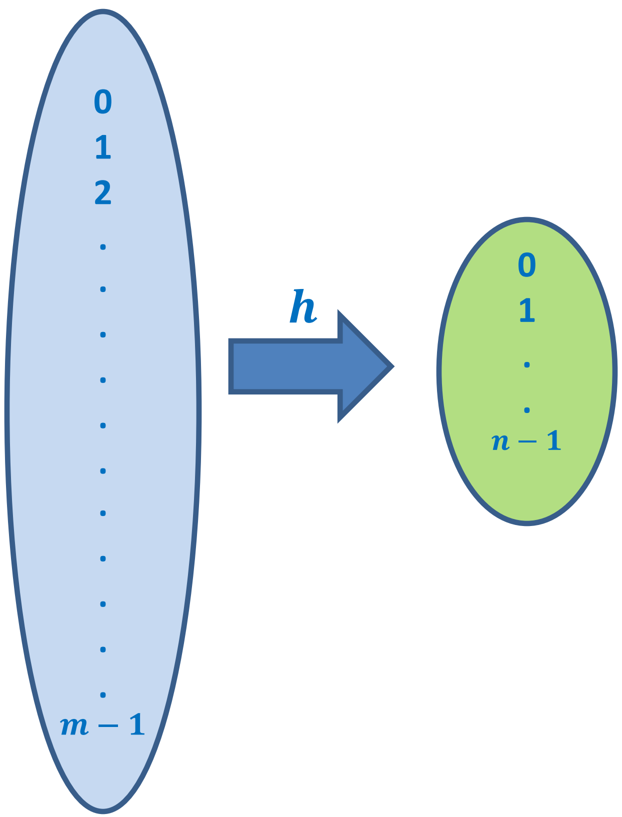
Example: n = few thousands, m = few trillions.

Question:

Can we have a data structure of **O**(n) size that can answer a search query in **O**(**1**) time ?

Answer: Hashing

Hash function



Hash function:

h is a mapping from U to $\{0,1,\ldots,n-1\}$ with the following characteristics.

- **Space** required for h
- $h(i)$ computable in

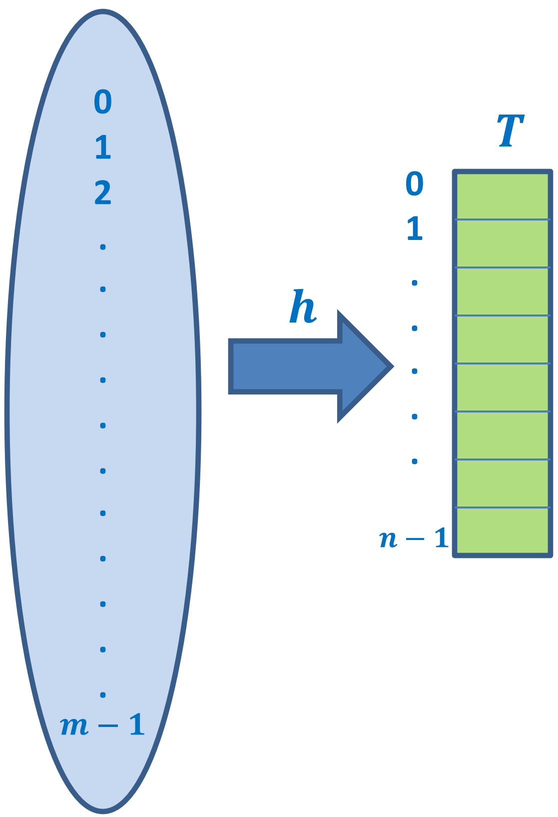
Example:

Hash value:

For a given hash function h, and $i \in U$.

$h(i)$ is called hash value of i

Hash function, hash value



Hash function:

h is a mapping from U to $\{0,1,\ldots,n-1\}$ with the following characteristics.

- **Space** required for h : a few **words**.
- $h(i)$ computable in **O(1) time** in **word RAM.**

Example: $h(i)$ = i **mod** n

Hash value:

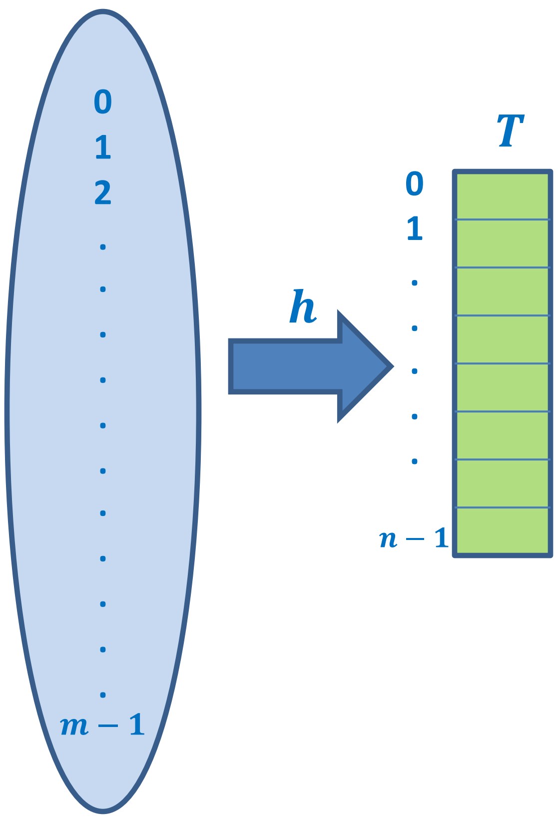
For a given hash function h, and $i \in U$.

$h(i)$ is called hash value of i

Hash Table:

An array $T[0 \ldots n-1]$

Hash function, hash value, hash table



Hash function:

h is a mapping from U to $\{0,1,\ldots,n-1\}$ with the following characteristics.

- **Space** required for h : a few **words**.
- $h(i)$ computable in **O(1) time** in **word RAM.**

Example: $h(i)$ = i **mod** n

Hash value:

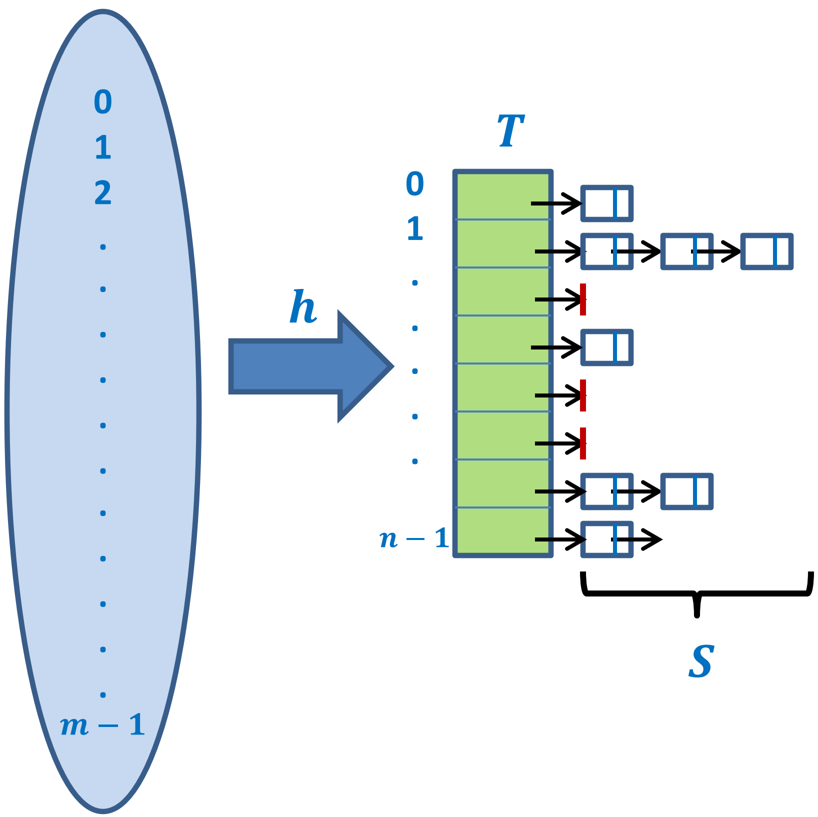
For a given hash function h, and $i \in U$.

$h(i)$ is called hash value of i

Hash Table:

An array $T[0 \ldots n-1]$

Hash function, hash value, hash table



Hash function:

$\boldsymbol{h}$ is a mapping from $\boldsymbol{U}$ to $\{0,1,\ldots,\boldsymbol{n}-\mathbf{1}\}$ with the following characteristics.

- **Space** required for $\boldsymbol{h}$: a few **words**.
- $\boldsymbol{h}(\boldsymbol{i})$ computable in **O(1) time** in **word RAM.**

Example: $\boldsymbol{h}(\boldsymbol{i})$ = $\boldsymbol{i}$ **mod** $\boldsymbol{n}$

Hash value:

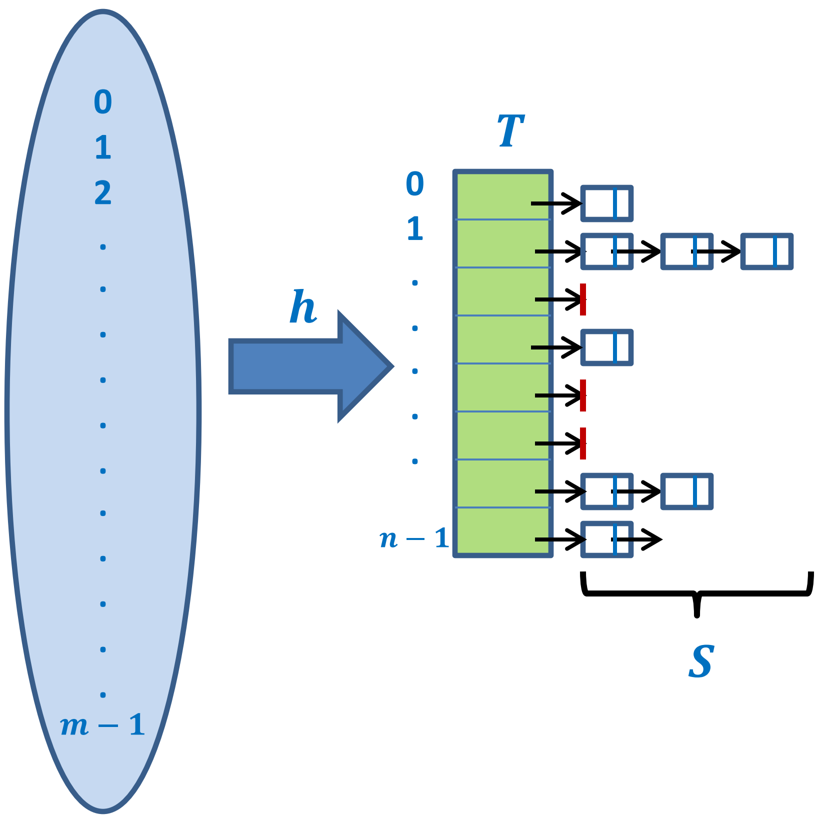
For a given hash function $\boldsymbol{h}$, and $\boldsymbol{i} \in \boldsymbol{U}$.

$\boldsymbol{h}(\boldsymbol{i})$ is called hash value of $\boldsymbol{i}$

Hash Table:

An array $\boldsymbol{T}[0 \ldots \boldsymbol{n}-\mathbf{1}]$ of pointers storing $\boldsymbol{S}$.

Hash function, hash value, hash table



Question:

How to use ($\boldsymbol{h}$,$\boldsymbol{T}$) for searching an element $\boldsymbol{i} \in \boldsymbol{U}$?

Answer:

$\boldsymbol{k} \leftarrow \boldsymbol{h}(\boldsymbol{i})$;

Search element $\boldsymbol{i}$ in the list $\boldsymbol{T}[\boldsymbol{k}]$.

Time complexity for searching:

O(length of the **longest** list in $\boldsymbol{T}$).

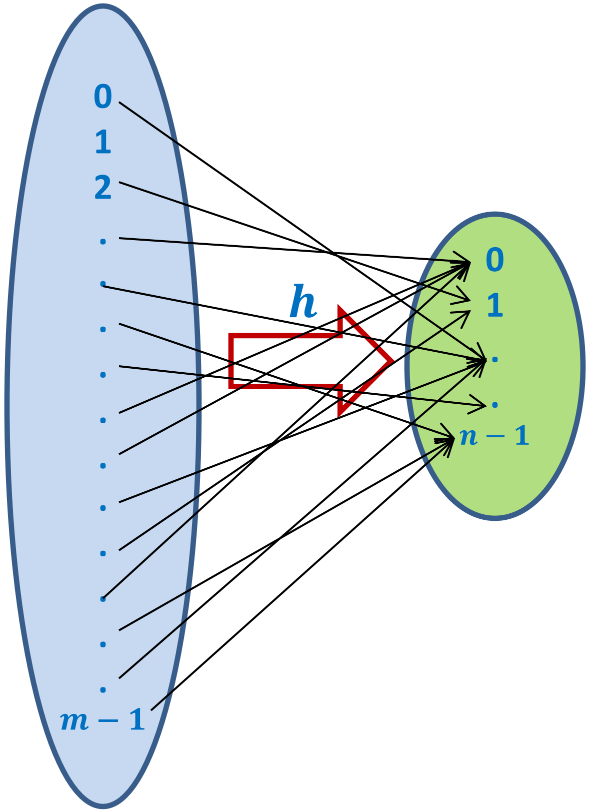
Efficiency of Hashing depends upon hash function

A hash function h is good if it can **evenly** distributes S.

Aim: To search for a good hash function for a given set S.

There **can not be** any hash function h which is good for **every** S.

Hash function, hash value, hash table



For every $\boldsymbol{h}$, there exists a subset of $\left\lceil \frac{m}{n} \right\rceil$ elements from $\boldsymbol{U}$ which are hashed to same value under $\boldsymbol{h}$.

So we can always construct a subset $\boldsymbol{S}$ for which all elements have same hash value

➔ All elements of this set $\boldsymbol{S}$ are present in a single list of the hah table $\boldsymbol{T}$ associated with $\boldsymbol{h}$.

➔ **O**(n) worst case search time.

Why does hashing work so well in Practice ?

$$\boldsymbol{h}(i) = i \ \mathbf{mod} \ n$$

Because the set $\boldsymbol{S}$ is usually a **uniformly random** subset of $\boldsymbol{U}$.

Let us do a **theoretical analysis to prove** this fact.

Why does hashing work so well in Practice ?

$$\boxed{\boldsymbol{h}(x) = \boldsymbol{x}\ \mathbf{mod}\ n}$$

Let $y_1, y_2, \ldots, y_n$ denote n elements selected <u>randomly uniformly</u> from $\boldsymbol{U}$ to form $\boldsymbol{S}$.

Question:

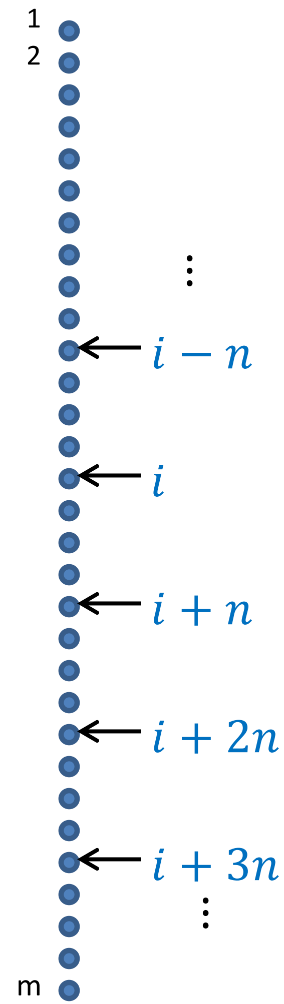
What is expected number of elements of $\boldsymbol{S}$ colliding with y_1?

Answer: Let y_1 takes value i.

P(y_j collides with y_1) = ??

How many possible values can y_j take ? $m - 1$

How many possible values can collide with i ?



Why does hashing work so well in Practice ?

$$\boxed{h(x) = x \bmod n}$$

Let $y_1, y_2, \ldots, y_n$ denote n elements selected randomly uniformly from U to form S.

Question:

What is expected number of elements of S colliding with y_1?

Answer: Let y_1 takes value i.

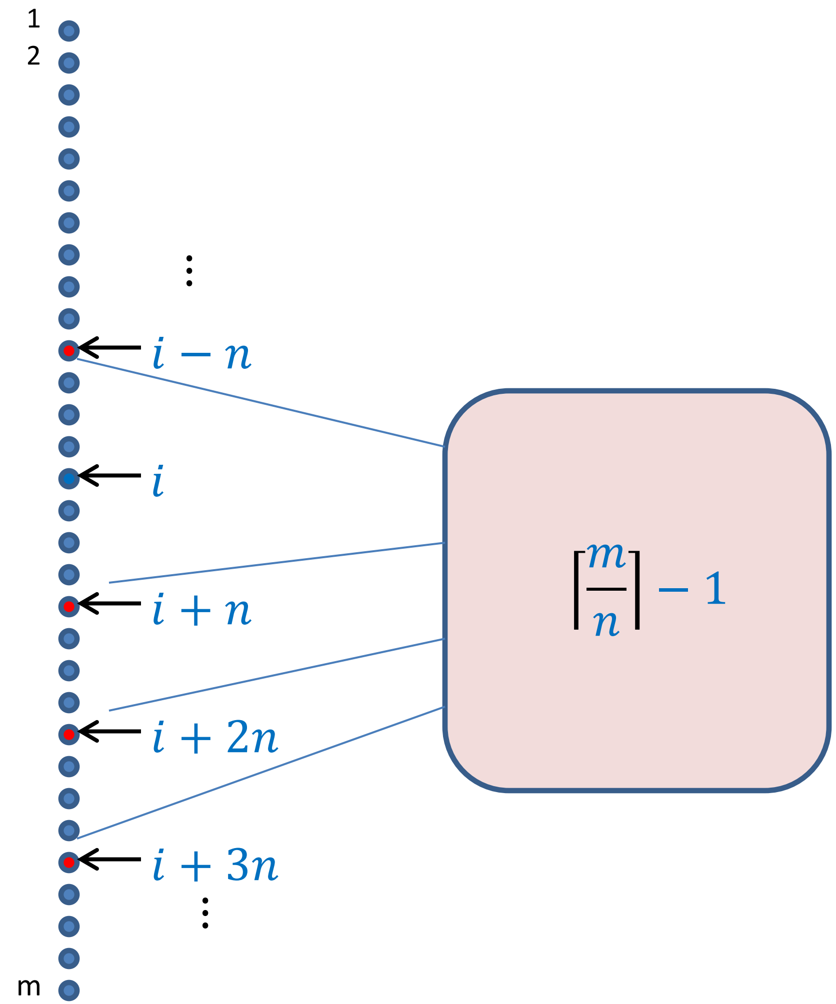
P(y_j collides with y_1) =

$$\frac{\lceil \frac{m}{n} \rceil - 1}{m-1}$$

Expected number of elements of S colliding with y_1 =

$$= \frac{\lceil \frac{m}{n} \rceil - 1}{m-1}(n-1)$$

$$= O(1)$$



Why does hashing work so well in Practice ?

Conclusion

1. $\boldsymbol{h}(i) = i \ \mathbf{mod} \ n$ works so well because
   for a uniformly random subset of $\boldsymbol{U}$,
   the **expected** number of collision at an index of $\boldsymbol{T}$ is **O**(1).

It is easy to fool this hash function such that it achieves **O**(s) search time.

(do it as a simple exercise).

This makes us think:

"How can we achieve worst case **O**(1) search time for a given set $\boldsymbol{S}$."

Hashing: theory

1953

1984

U : $\{0,1,\ldots,m-1\}$

$S \subseteq U$,

$n = |S|$,

Theorem [FKS, 1984]:

A hash table and hash function can be computed in average **O**(n) **time** for a **given** S s.t.

Space : **O**(n)

Query time: worst case **O**(1)

Ingredients :

- elementary knowledge of **prime numbers**.
- The algorithms use **simple randomization**.

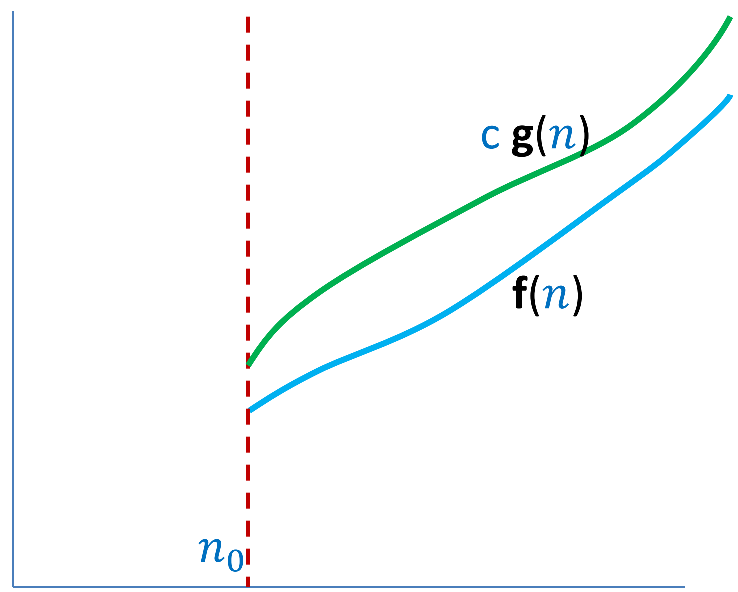
Order notation

Definition: Let **f**(n) and **g**(n) be any two increasing functions of n.

f(n) is said to be

if there exist constants c and n_0 such that

$$f(n) \le c\, g(n) \quad \text{for all } n > n_0$$



f(n) = O(g(n))

Order notation extended

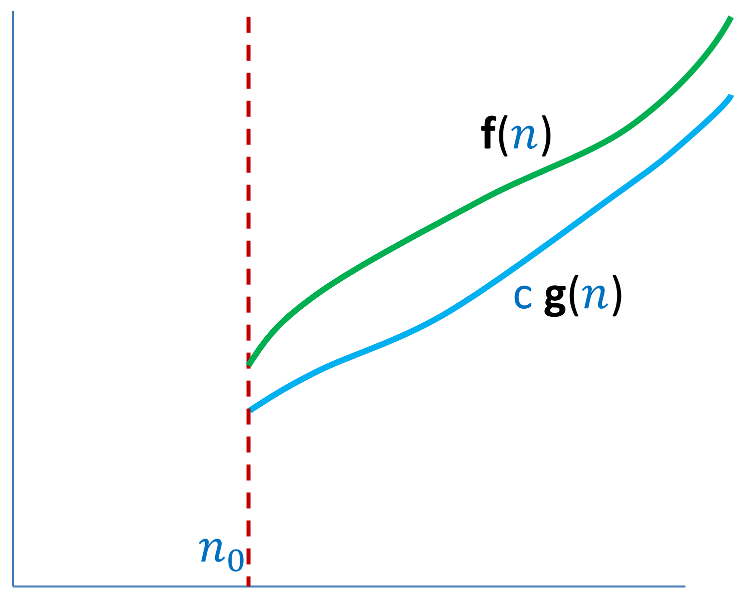
Definition: Let **f**(n) and **g**(n) be any two increasing functions of n.

f(n) is said to be

if there exist constants c and n_0 such that

$$f(n) \geq c\, g(n) \quad \text{for all } n > n_0$$

f(n) = Ω(g(n))



$$\frac{n^2}{100} = \Omega(10000\, n \log n)$$

Order notation extended

Observations:

- $\mathbf{f}(n) = \mathbf{O}(\mathbf{g}(n))$ **if and only if** $\mathbf{g}(n) = \boldsymbol{\Omega}(\mathbf{f}(n))$

One more Notation:

If $\mathbf{f}(n) = \mathbf{O}(\mathbf{g}(n))$ **and** $\mathbf{g}(n) = \mathbf{O}(\mathbf{f}(n))$, then

$$\mathbf{g}(n) = \boldsymbol{\Theta}(\mathbf{f}(n))$$

Examples:

- $\frac{n^2}{100} = \boldsymbol{\Theta}(10000\, n^2)$

- Time complexity of Quick Sort is $\boldsymbol{\Omega}(n \log n)$

- Time complexity of Merge sort is $\boldsymbol{\Theta}(n \log n)$