



# Data Structure Training

## Algorithm & Complexity

Department of CSE, CS & IT  
ABES Engineering College, Ghaziabad

# Data Structure Training(Algo&Complexity)

## Algorithm-

- 1-Finite set of steps to solve a problem is called algorithm.
- 2-Here steps means instruction which contain fundamental operators.

invalid symbol

$$\begin{array}{l} 2 = 2 \oplus 1 ; \\ \hline 2 = 2 \oplus 1 ; \end{array}$$

## Characteristics Of fundamental Instruction-

### 1-Definiteness-

Every fundamental instruction must be definite without any ambiguity.

$i = i + 1$  ✗  
 $i = i + 1;$  ✓

### 2-Finiteness-

Every instruction must be terminate within finite amount of time.

$i = 1;$  ✓  
while(1)  
{  
     $i = i + 1;$   
}

Infinite loop

break;

$i = i + 1;$  //not allowed in algorithm since executing infinitely. ✓

- 3- Every instruction must accept atleast 0 input and provides atmost 1 output.

# Data Structure Training(Algo&Complexity)

Steps For Solving Any Problem-

1-Identifying Problem Statement ✓

2-Identifying Constraints ✓✓

4 Queen Problem

Q <sub>1</sub>	×	×	×
×	×		
×		×	
×			×

Q<sub>1</sub>  
Q<sub>2</sub>  
Q<sub>3</sub>  
Q<sub>4</sub>  
→

# Data Structure Training(Algo&Complexity)

## Steps For Solving Any Problem-

### 3-Design Logic- ✓✓

Depends on the characteristics of the problem we can choose any one of the following design strategy for design logic.

- a) Devide & Conquer ✓
- b) Greedy Method ✓
- c) Dynamic Programming ✓
- d) Branch & Bound ✓
- e) Backtracking etc..... ✓

X	Q <sub>1</sub>	X	X
X	X	X	Q <sub>2</sub>
Q <sub>3</sub>	X	X	X
X	X	Q <sub>4</sub>	X

Q<sub>4</sub> X

✓✓

### 4-Validation-

Most of the algorithm validated by mathematical induction.

### 5-Analysis-

Process of comparing two algorithms with respect to time, space, number of register, network bandwidth etc is called analysis. ✓✓✓✓

# Data Structure Training(Algo&Complexity)

## Types Of Analysis-

### Priory Analysis- ✓

1-Analysis done before executing

$X=x+1$ ;

### 2-Principle-✓

Frequency count of fundamental instruction.

Since  $x=x+1$  being carried out only one time so its complexity is  $O(1)$  order of 1  
 $\hookrightarrow O(1)$

3-It provides estimated values.

4-It provides uniform values.

5-It is independent of CPU, O/S & system architecture

type  $\Rightarrow$  logic

(10)

(12)

hashing

64  
8413

32  
4413

# Data Structure Training(Algo&Complexity)

## Types Of Analysis-

### Posterior Analysis-✓

1-Analysis done after executing

X=x+1;

3-It provides exact values.

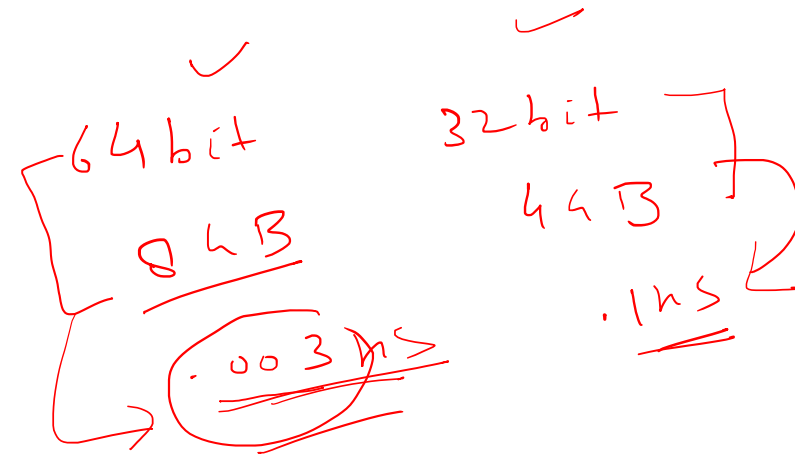
4-It provides non uniform values.

5-It is dependent of CPU,O/S & system architecture

6- Implementation →

c, c++, java, python

7-Testing And Debugging



# Data Structure Training(Algo&Complexity)

## Time Complexity

- 1-Simple for loop ✓
- 2-Nested for loop ✓
- 3-if-else ✓
- 4-Recursive Algorithm ✓



# Data Structure Training(Algo&Complexity)

## Simple for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0, i;
```

```
    for(i=1; i<=n; i++)
```

```
    {
```

```
        sum=sum+i;
```

```
    }
```

```
    return 0;
```

```
}
```

$$\frac{1+2+\dots+n}{2} = \frac{n(n+1)}{2}$$

$$1+1+1+\dots+1 = n$$

$$1+1+n+1+h+h+1 \Rightarrow 3n+4 \Rightarrow O(n)$$

$$\frac{\text{sum}}{10} \quad \frac{1}{11}$$

$$h = \text{large} + \text{const}$$

$$3n + 4 \Rightarrow \text{const}$$

$$\Rightarrow O(n)$$



# Data Structure Training(Algo&Complexity)

## Simple for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0,i; x
```

```
    for(i=1; i<=n; i=i+2) →
```

```
    {
```

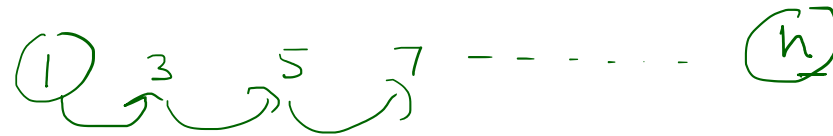
```
        sum=sum+i;
```

```
    }
```

```
    return 0; x
```

```
}
```

AP



$$t_k = a + (k-1)d$$

$$h = 1 + (k-1)2$$

$$h = 1 + 2k - 2$$

$$\frac{h+1}{2} = k$$

$$k = \frac{10+1}{2} \Rightarrow 5.5 \Rightarrow 6$$

$$\underline{\underline{O(n)}}$$

# Data Structure Training(Algo&Complexity)

## Simple for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
int main(void)
{
    int sum=0,i;
    for(i=1;i<=n;i=i*2)
    {
        sum=sum+i;
    }
    return 0;
}
```

Handwritten analysis of the code's time complexity:

Given  $n = 10$ , the sequence of values for  $i$  is  $1, 2, 4, 8, 16, \dots$ . The loop terminates when  $i > n$ .

The number of iterations is  $k$ , where  $i = 2^{k-1}$ . The total number of comparisons is  $1 + \log_2 n$ .

The time complexity is  $O(\log_2 n)$ .

Derivation of the time complexity:

$$h = 1 + \log_2 n$$
$$\log_2 h = \log_2 (1 + \log_2 n)$$
$$\log_2 h = (1 + \log_2 n) \log_2 2$$

Final result:  $O(\log_2 n)$

# Data Structure Training(Algo&Complexity)

## Simple for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
int main(void)
{
    int sum=0,i;
    for(i=n;i>0;i=i/2)
    {
        sum=sum+i;
    }
    return 0;
}
```

$$k = 1 + \log_2 n$$
$$= 5$$

$$n=10$$
$$5 \mid 10 \quad 5 \mid 5 \quad 2 \mid 5 \quad 1 \mid 2 \quad 0$$

$$\textcircled{n} \quad n/2 \quad n/4 \quad n/8 \quad \dots \quad 1$$

$$t_k = a r^{k-1}$$

$$1 = h \left( \frac{1}{2} \right)^{k-1}$$

$$1 = \frac{n}{2^{k-1}} \Rightarrow$$

$$2^{k-1} = n$$
$$\log_2 2^{k-1} = \log_2 n$$
$$(k-1) = \log_2 n$$
$$k = 1 + \log_2 n$$

# Data Structure Training(Algo&Complexity)

## Simple for loop

Find total comparison and time complexity of following code and let val(j) denote value stored in variable j. after termination of loop what is the value of val(j)

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0,i,j;
```

```
    for(i=n; j=0; i>0; i=i/2, j=j+i)
```

```
    {
```

```
        sum=sum+i;
```

```
    }
```

```
    return 0;
```

```
}
```

assignment

$$\frac{1}{\sqrt{0}} \times$$

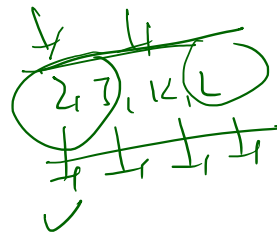
n

n/2

n/4

...

1



$$O(\log_2 n)$$

# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0,i,j;
```

```
    for(i=1;i<=n;i++)
```

```
    {
```

```
        for(j=1;j<=n;j++)
```

```
        {
```

```
            sum=sum+j;
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

$$n = 10$$

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

$$n^2 + n$$
$$O(n^2 + n)$$

1 2 3 4 ... 10

1 1 1 1 1 1 1 1 1 1

$$111$$
$$110 + 1$$
$$10(11) + 1$$
$$n(n+1) + 1$$

$$15$$
$$12$$
$$8$$
$$16$$
$$h$$
$$10$$
$$11$$
$$h+1$$

$$110 + 1$$

$$= 111$$

$$O(n^2)$$

$$100$$

# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
    int sum=0,i,j;
    for(i=1;i<=n;i++)
```

```
{
    for(j=1; j<=n; j=j+2)
    {
        sum=sum+j;
    }
}
```

```
}  
return 0;
```

}

2

1

2

$$1 - h$$

7

$$\frac{h+1}{2}$$
$$\frac{h+1}{2}$$
$$\frac{n+1}{2}$$
$$n(n+1)/2$$
$$h^2/2 + w/2$$

$O(n^2)$

# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
int main(void)
{
    int sum=0,i,j;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j=j*3)
        {
            sum=sum+j;
        }
    }
    return 0;
}
```

Handwritten notes for complexity analysis:

- $\log_2 n \Rightarrow 10$
- $n \log_2 n$
- $n \log_3 n$
- $\log_2 n$
- $\log_3 n$
- $\log_3 n$
- $-n$

Handwritten notes for complexity analysis:

1 3 9 -27 ... n

---

$T_n = a r^{n-1}$

$n = 1 \cdot 3^{n-1}$

$\log_3 n = \log_3 3^{n-1}$

$\Rightarrow n-1$

$n = 1 + \log_3 n$

# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0,i,j;
```

```
    for(i=1;i<=n;i=i*2)
```

```
    {
```

```
        for(j=1;j<=n;j=j+2)
```

```
        {
```

```
            sum=sum+j;
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

→ 60 sec

1 2

$1 + \log_2 n$

x

$\frac{n+1}{2}$

$\frac{n+1}{2}$

$O(n \log_2 n)$

$\frac{n+1}{2} + \frac{n+1}{2} \log_2 n$

$\frac{n}{2} + \left(\frac{1}{2}\right) + \frac{n}{2} \log_2 n + \frac{1}{2} \log_2 n$



# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int sum=0,i,j;
```

```
    for(i=1;i<=n;i=i+1)
```

```
    {
```

```
        for(j=1;j<=i;j=j+2) →
```

```
        {
```

```
            for(k=1;k<=n;k=k*2)
```

```
            {
```

```
                sum=sum+k;
```

```
            }
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

$$O(n^2 \log_2 n)$$

$$\begin{matrix} 1 \\ 2 \\ \vdots \\ n \end{matrix}$$

1

2

h

$$\begin{matrix} h^2+1 \\ h^2 \log_2 h + 1 \\ 2 \\ 0 \end{matrix} \quad \begin{matrix} 1 \\ 1+\log_2 h \\ 1+\log_2 h \\ 1+\log_2 h \end{matrix}$$

$$h \times \left( \frac{h+1}{2} \right) (1 + \log_2 h),$$

$$= \frac{n^2}{2} + \frac{n}{2} (1 + \log_2 h) + \frac{n}{2} + \frac{n}{2} \log_2 h$$

# Data Structure Training(Algo&Complexity)

## Nested for loop

Find total comparison and time complexity of following code

```
#include <stdio.h>
int main(void)
{
    int sum=0,i,j;
    for(i=0;i<=n;i=i+1)
    {
        for(j=1;j<=i;j=j+1)
        {
            if(j%i==0)
            {
                for(k=1;k<=n;k=k+1)
                {
                    sum=sum+k;
                }
            }
        }
    }
    return 0;
}
```

# Data Structure Training(Algo&Complexity)

## Recursive Algorithm-

Find total comparison and time complexity of following code

```
int fact(int n)
{
    if(n==0 || n==1)
        return 1;
    else
        return n*fact(n-1);
}
```

# Data Structure Training(Algo&Complexity)

## Recursive Algorithm-

Find total comparison and time complexity of following code

```
int fib(int n)
{
    if(n==0)
        return 0;
    if(n==1)
        return 1;
    else
        return fib(n-1) +fib(n-2);
}
```

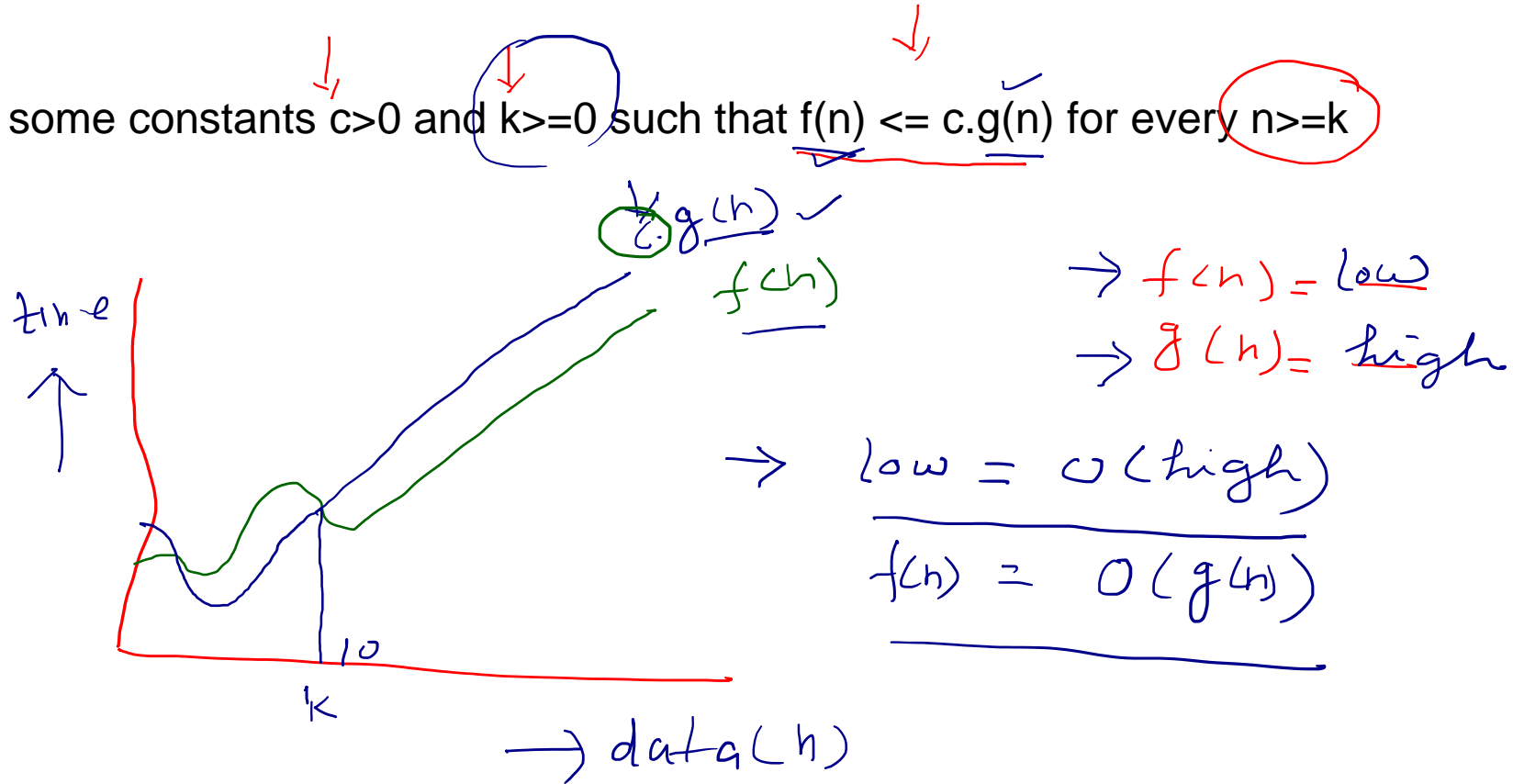
# Data Structure Training(Algo&Complexity)

Asymptotic Notation-

to compare two algorithms rate of growth with respect to time and space, need asymptotic notation.

Big-Oh (O)

$f(n)$  is  $O(g(n))$  iff there exist some constants  $c > 0$  and  $k \geq 0$  such that  $f(n) \leq c \cdot g(n)$  for every  $n \geq k$



# Data Structure Training(Algo&Complexity)

$$\textcircled{1} \rightarrow f(n) = a_0 + a_1n + a_2n^2 + \dots \rightarrow \downarrow \textcircled{a_m n^m}$$

$$\frac{f(n) \times}{g(n) \times}$$

$$\underline{O(n^m)} \checkmark$$

$$\underline{f(n) = O(g(n))}$$

$$= O(n^2)$$

$\textcircled{2}$

$$\underline{\text{low} \checkmark = O(\text{high} \checkmark)} \rightarrow$$

$$\underline{f(n) = n^2 + n + 1}$$

$$n^2 \leq n^2$$

$$n^2 + 1 \leq n^2 + n^2 \checkmark$$

$$n^2 + n + 1 \leq \underline{n^2 + n^2 + n^2} \checkmark$$

$$\downarrow ?$$

$$O(\underline{g(n)})$$

$$\frac{n^2 + n + 1}{f(n)}$$

$$\downarrow \textcircled{3n^2}$$

$$\downarrow c \quad \downarrow g(n)$$

# Data Structure Training(Algo&Complexity)

$$n^2 \leq n^3$$

$$\underline{n^2 + n + 1} \leq n^3 + n^3 + n^3$$

$$f(n) \leq 3n^3$$

$\downarrow \quad \downarrow$   
 $\subset \quad \subset \quad g(n)$

$$\underline{O(n^3)}$$

$$f(n) \leq 3n^4$$

$\downarrow$

$$O(n^4) \checkmark$$

$\downarrow$

~~$n^2$~~   $n^3$   $n^4$  ...  $n^3$

$$\vdots$$
$$n^n$$

$\Rightarrow$  ! Ub = least upper bound

# Data Structure Training(Algo&Complexity)

$$\downarrow$$

$$f(n) = n^2 \log n \quad g(n) = n (\log n)^{10}$$

$$\rightarrow \underline{f(n)} = \underline{o(g(n))}, \quad \underline{g(n)} \neq o(\underline{f(n)}) \quad \checkmark$$

$$\rightarrow f(n) = o(g(n)), \quad g(n) = o(f(n)) \quad \checkmark$$

$$\Rightarrow g(n) = o(f(n)), \quad \underline{f(n)} \neq o(g(n)) \quad \checkmark$$

$$\underline{g(n) = o(f(n))}$$

$$\begin{aligned} n &> (\log n)^2 \\ 2 &> (\log^2 n) \rightarrow n^2 \log n \\ &\downarrow \\ &\rightarrow n \cdot n \log n \end{aligned}$$

$$\begin{aligned} &n (\log n)^{10} \\ &\underline{n \log n} \cdot \underline{(\log n)^9} \end{aligned}$$

$$\begin{aligned} f(n) &> g(n) \\ \uparrow &\quad \quad \uparrow \end{aligned}$$



# Data Structure Training(Algo&Complexity)

$$\begin{matrix} f(n) \\ \underline{n^2 \log n} \end{matrix}$$

$$n * n \log n$$

$$\begin{matrix} \checkmark \\ h * \underbrace{n \log n}_{\text{high}} \end{matrix}$$

$$\underbrace{h}_{\text{low}}$$

$$\textcircled{2}$$

$$2$$

$$2$$

$$\begin{matrix} g(n) \\ \underline{n(\log n)^{10}} \end{matrix}$$

$$h(\log n)^1(\log n)^9$$

$$\begin{matrix} \checkmark \\ \underbrace{h \log n}_{\text{low}} (\log n)^9 \end{matrix}$$

$$\underbrace{(\log n)^h}_{\text{high}}$$

$$> (\log_2 2)^2$$

$$> (1)^2$$

$$\text{low} = O(\text{high})$$

$$\underline{g(n) = O(f(n))}$$

.3010

# Data Structure Training(Algo&Complexity)

$$\underline{f(n) = n!} \quad O(?)$$

$$= n(n-1)(n-2) \dots 1$$

$$= \underbrace{n}_{\downarrow} \underbrace{n(1-\frac{1}{n})}_{\downarrow} \underbrace{n(1-\frac{2}{n})}_{\downarrow} \dots \underbrace{n(1-\frac{1}{n})}_{\downarrow}$$

$$= \underbrace{(n^n)}_{\text{circled}} \left[ 1 \cdot \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \dots \frac{1}{n} \right]$$

$$f(n) = a_0 + a_1 n + \dots + a_m \underbrace{(n^m)}_{\text{circled}}$$

$$= \underline{O(n^{\check{m}})}$$

# Data Structure Training(Algo&Complexity)

$$f(n) = \log(n!)$$

$$= \log(O(n^h))$$

$$\leq C \log n^h$$

$$= \underline{O(n \log n)} \checkmark$$

$$f(n) = O(?)$$

$$\begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}$$

# Data Structure Training(Algo&Complexity)

$$\begin{array}{ccccccccccc} & & & & f(n) & & & & g(n) & & \\ & & & & \downarrow \checkmark & & & & \downarrow & & \\ 2^{2^n} & \geq & n! & \geq & 4^n & \geq & 2^n & \geq & n^3 & \geq & n^2 & \geq & n \log n & \geq & n \\ & & & & & & & & & & & & & & \} \\ \rightarrow & & & & & & \geq & \log \log n & \geq & \frac{\log n}{\log \log n} & \geq & 1 & & \end{array}$$

Dominance Ranking

$$\underline{g(n) = O(f(n))}$$

# Data Structure Training(Algo&Complexity)

$\Omega$  notation

$\Theta \rightarrow$  ✓

$$\underline{f(n) = O(g(n))}$$

$$C > 0$$

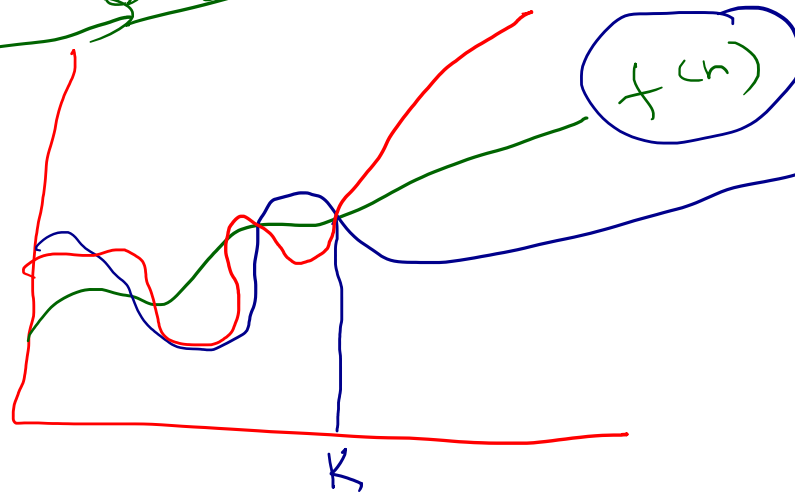
$$K \geq 0$$

$$\rightarrow f(n) \leq C g(n) \rightarrow O \quad \checkmark$$

$$\rightarrow \underline{f(n) \geq C \cdot g(n)} \quad \underline{\Omega(n)}$$

$$\frac{1n^2}{C_1 g(n)} \leq \frac{n^2 + n + 1}{f(n)} \leq \frac{3n^2}{g(n)}$$

$\Theta(n^2)$  ✓



$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

# Data Structure Training(Algo&Complexity)

$$f(n) = \underline{\underline{n^2 + n + 1}} \quad \underline{\underline{O(g(n))}}$$

$$n^2 \geq n^2$$

$$n^2 + n \geq n^2$$

$$\underline{\underline{n^2 + n + 1}} \geq \textcircled{1} \underline{\underline{n^2}}, n, 1 \quad \underline{\underline{\Omega(n)}} \quad f(n) > g(n)$$

$$\rightarrow \underline{\underline{\Omega(1)}} \quad \underline{\underline{\Omega(n^2)}}$$

Atleast

$$\textcircled{n^2} \quad \textcircled{n^2} \quad \textcircled{1} \quad \underline{\underline{\Omega(n^2) \leftrightarrow \underline{\underline{O(n^2)}}}}$$

GLB

# Data Structure Training(Algo&Complexity)

Diagram illustrating a recursive process:

- Start at 0.
- Move to  $\sqrt{n}$  (circled in red).
- Move to  $n/2$ .
- Move to  $n/4$ .
- Move to  $\sqrt{n}$  (circled in red).

Annotations:

- The word "almost" is written above the first  $\sqrt{n}$ .
- The word "almost" is written above a red box containing the word "almost".
- A red arrow loops back from the second  $\sqrt{n}$  to 0.

isprime (Int n)

Int L, n;

for (L = 2, L <= sqrt(n); L++)

{

if (n % L == 0)

{

return 0;

}

return 1;

Handwritten notes and corrections:

- $O(\sqrt{n})$  (with a checkmark)
- $\times O(1) \equiv$  (with a checkmark)
- $\frac{n}{196}$
- $\sqrt{196}$
- $(14)$  (circled)
- $2, 3, 4, (5)$  (with a checkmark)
- $\sqrt{n}$  (with a checkmark)
- $\text{sqrted}$  (with a checkmark)
- $\Omega(1)$  (with a checkmark)
- $99 \cdot 1 \rightarrow O(1) \leftrightarrow O(\sqrt{n})$  (with a checkmark)
- $\frac{O(1)}{\Omega(1)}$  (with a checkmark)
- $\sqrt{5} \times \rightarrow$  (with a checkmark)