

## Time & Space Complexity Started

Date → 8 November  
16/1

Day → Monday

Agenda for today: 6-7:30pm meeting with  
1. 1 Product to Time and Space Sample

## 1. Introduction to Time and Space Complexity

## 1. Time Complexity →

→ Time Complexity of an algorithm is the representation of the amount of time required by the algorithm to execute to completion

किसी भी algorithm को पूरी तरीके से execute होने में जो  
eff time चाहिए होगा, और उस time की अवधि हम  
represent करें तो उसे Time Complexity कहते हैं।  
उस algorithm की

अपर हमें किसी भी algorithm की efficiency check करना पड़ता है (कि वो algorithm कितना efficient है)

↳ efficient अल्गोरिदम  
विलोभी eligible हैं  
या विलोभी नियुण (क्रमातः)  
हैं उस algorithm की

लो किसी भी algorithm की efficiency determine करने के  
- 2 important factors होते हैं:-

2 important factors: Eth & Env

- The time is calculated or measured by counting the number of key operations such as comparison in sorting algorithm (Ex Find GTR)
  - For example → # number of comparisons

The space is calculated or measured by counting the maximum memory space required by algorithm

Date \_\_\_\_\_  
Page \_\_\_\_\_

## 2. Space Complexity

Space Complexity of an algorithm represents the amount of memory space the algorithm needed in its life cycle.

# TIME COMPLEXITY

किसी भी problem को solve करने के हमारे पास  $n$  solution हो सकते हैं (eg → हम अपने सब friends के problem discuss करें तो सब अलग -अलग solution बताएँगे).

ऐसे ही किसी भी program को solve करने के  $n$  no. of solution हो सकते हैं।

Example ↗

Ques. अगर हमें किसी भी number का square find करना है।  
→ To find the square of a number.

**SOLUTION 1**

One solution can be running a loop for  $n$  times, starting with the number  $n$ , and adding  $n$  to it until  $n=5$ ,

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

$$\{ \quad n = n + n;$$

}

return  $n$ .

Second solution can be, we can just use a mathematical operator  $(*)$ , multiply its to find the square until  $n=5$ .

return  $n * n$ ;

$$\Downarrow$$

$$n=25$$

↑

This is the Square

पहले हमने  $n$  times loop run किया था।

$n$  times operation की (addition) की, तो  $n$  operation में प्रयोग किया तो  $n$  times 1 operation हुआ तो बहुत कम time लगा।

$$\Downarrow$$

$$n=25.$$

पहले हमने क्या करके थे?  
multiply operation किया तो वही 1 operation हुआ तो बहुत कम time लगा।

## SOLUTION 1 TO FIND SQUARE

unit  $n = 5$  Time यहाँ  $n$  पर depend  
for (unit  $i = 1; i < n; i++$ ) करे RE की कितनी न की value होगी? उतनी GTR time लगेगा  
 $n = n + 5$  यहाँ look  $n$  no. of times होगा  
if : करे तकि न की value होगी  
return  $n$  To time complexity will be  
n atleast

Time Complexity इत्तम estimate किया जाता है  
by calculating the no. of elementary steps performed  
by an algorithm to finish execution.

## SOLUTION 2 TO FIND SQUARE

unit  $n = 5$

return  $n * n;$

→ पहली time complexity  
constant होती है  
फिर यहाँ time  $n$  की  
value TR depend होती है (  $n$  की value  
कुछ भी हो, time  
तो सक operation की  
लिए ही लगता )

इत्तम Time Complexity को usually  
using the metric Big O Notation

→ Big Oh मिलता worst case करता है  
Time लगता

## Types of Notations for Time Complexity

1. Big Oh → denotes "fewer than or same as"  
 $\langle \text{expression} \rangle$  iterations
2. Big Omega → denotes "more than or same as"  
 $\langle \text{expression} \rangle$  iterations
3. Big Theta → denotes "the same as"  
 $\langle \text{expression} \rangle$  iterations
4. Little Oh → denotes "fewer than"  
 $\langle \text{expression} \rangle$  iterations
5. Little Omega → denotes "more than"  
 $\langle \text{expression} \rangle$  iterations

There are different types of Time

Complexities used:

1. Constant time  $\rightarrow O(1)$
2. Linear time  $\rightarrow O(n)$ .
3. Logarithmic time  $\rightarrow O(\log n)$
4. Quadratic time  $\rightarrow O(n^2)$ .
5. Cubic time  $\rightarrow O(n^3)$

and many more complex notations  
like Exponential time,  
Quasilinear time,  
Factorial time,  
etc.

1.

$O(1)$  → Constant time

Date \_\_\_\_\_  
Page \_\_\_\_\_

→ An algorithm is said to have constant time of Order  $O(1)$  when it is not dependent on the input size  $n$ . irrespective of the input size  $n$ , the runtime will always be same.

→ Input के size पर depend नहीं करेगा time taken by algo.

पहले Input के लिए अपेक्षित time लगा = पहले Input के लिए अपेक्षित time लगा

Input का Size छोला या उड़ा करेने से  
Time consumed पर  
फ़ैट फ़ैट नहीं पड़ेगा ।

Eg 1)

$$\text{Time consumed in doing } (200 * 500) = \frac{\text{Time consumed in doing } (2 * 5)}{100}$$

**How ?** → 1) ये सा इसलिए है क्योंकि 2 और 5 भी विषय हैं। ये गाँठ लगा देते हैं।

• integer

2) Integer Size in Java = 4 bytes = 32 bits

$2^{15}$  stored in 32 bits  $\rightarrow$  00000000000000000000000000000010

5 It's stored in 32 bits  $\rightarrow$  00000.00000000000000000000000000000000

जितना time 2 की 5 से multiply करने में लगेगा, उन्हाँही time 200 को 500 से multiply करने में कमी है। 200 की तीव्रता की case में सारे 32 bits process करने ही

200 is stored in 32 bit  $\rightarrow$  00000000000000000000000000000000

500 is stored in 32 bits → 0000 0000 0000 0000 0000 0000 0000 1111 0100



Type

# Initial Equation      Final Equation      Examples

 $O(1)$ 

Constant

$$T(n) = K$$

Time taken for  
Input of Size  $n$

$$T(n) = K$$

 $c_0 + c_1 n + c_2$  $c_3 -$  $c_4 *$  $c_5 /$  $c_6 \%$  $c_7 \wedge$  $c_8 /$  $c_9 \&$  $c_{10} \oplus$  $c_{11} \ominus$  $c_{12} \otimes$  $c_{13} \oplus \ominus$  $c_{14} \otimes \oplus$  $c_{15} \otimes \ominus$  $c_{16} \oplus \otimes$  $c_{17} \otimes \oplus \ominus$  $c_{18} \oplus \otimes \oplus$  $c_{19} \otimes \oplus \oplus$  $c_{20} \oplus \otimes \oplus \ominus$  $c_{21} \otimes \oplus \oplus \oplus$  $c_{22} \oplus \otimes \oplus \oplus \ominus$  $c_{23} \otimes \oplus \oplus \oplus \oplus$  $c_{24} \oplus \otimes \oplus \oplus \oplus \ominus$  $c_{25} \otimes \oplus \oplus \oplus \oplus \oplus$  $c_{26} \oplus \otimes \oplus \oplus \oplus \oplus \ominus$  $c_{27} \otimes \oplus \oplus \oplus \oplus \oplus \oplus$  $c_{28} \oplus \otimes \oplus \oplus \oplus \oplus \oplus \ominus$  $c_{29} \otimes \oplus \oplus \oplus \oplus \oplus \oplus \oplus$  $c_{30} \oplus \otimes \oplus \oplus \oplus \oplus \oplus \oplus \ominus$  $c_{31} \otimes \oplus \oplus \oplus \oplus \oplus \oplus \oplus \oplus$ 

Plus

Minus

Multiply

Divide

Modulus

Xor

Divide

And

Array के अंदर पर्याप्त

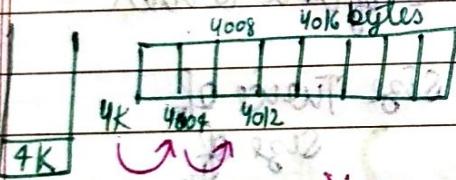
स्थिति से get

करना, या Set करना

taken constant

or same time

Because:



Address में ER

element of array

कोलेक्शन 4 byte होनी चाही है

bcz 1 integer takes 4bytes

Take equal  
time

$$\text{arr}[5] = 20$$

$$\text{arr}[500] = 20$$

Position

पर get/set

करना होता

time equal

पर get/set Position

के लिए

ऐसे arr[5] पर कोई value ST लगें में नहीं

time लगेगा तब वही time arr[500] पर

कोई value ST लगें में लगेगा।

ऐसी arr[5] की value ST address 4E हो जाएगी

arr[500] की value ST address 4E हो जाएगी

CPU के

पर address

पुराना

equal

time लगेगा।

∴ Time Complexity  
is same

2.

 **$O(n)$** 

→ Linear Time

An algorithm is said to have linear time complexity when the running time increases linearly with the size of input.

As Input of Size  $n$  will take Time  
consumed of  $n$

\* When the function involves checking the values in input data, such function has the Time Complexity of Order  $O(n)$ .

Input data की हर values पर अपर की function चलता है (कुछ कम तक अपर Input data की value की होती है), तो यह function की time complexity होती है of the order  $O(n)$ .

अपर की length या size Twice of size of  $n_2$ .

$$n_1 = 2n_2$$

$$T(n_1) = 2 T(n_2)$$

∴ Time of execution of  $n_1$

will be twice the time taken in execution of  $n_2$ .

# Linear Search

Eg. find element in an array.

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

public static int linearSearch(int arr, int x){}

for (int i=0; i<~~arr.length~~ length ; i++)

input की length TR depend

करते हैं Time

if (arr[i] == x) {

return i;

मिलनी वारे i comparison (input length)

उल्लंघन करते हैं Time

अमानवीय 1 OR comparison में  
K time लगता है

return -1;

मिल दी नहीं

तो Best Case क्या होगा यहाँ?  
जब पहला ही element x के  
equal बनता है तो हमेहस  
1 comparison करनी पड़ी।



Time : K K K K K

Worst Case में

सबसे ज्यादा compare

होना पड़ेगा

Worst Case में हम मानते हैं कि x element  
array में ही नहीं

तो हमने n element से compare किया, तो K time

दिलाकर element ही नहीं

(एक वार compare करने में K time लगता)

तो n element को compare करने में

$$\begin{aligned}
 &= K + K + \dots + K \quad (\text{n times } K \text{ को add करने}) \\
 &= (K)(n) \\
 &= nk
 \end{aligned}$$

**T(n)  $\propto$  n**

अब array का Size double होता है

तो Search करने में time भी double

happens

Type

$O(n)$



Linear

Initial Equation

$$T(n) = Kn$$

Final Equation

$$T(n) \propto n$$

Example

10	20	30	40
1	2	3	

oc = 50 element to search

Worst case एक भी element  
में से कोई नहीं मिले तो average में

Index [array[index] ≠ 50]

comparison

तो अब हमारे पास  $(n-1)$  element बचे & check करने के लिए

करने में K

constant  
time में

मिला

उत्तर हमें  $n-1$  element  
में search करना हो, तो first element को compare

करने में अगर K constant time

लगता है। अब हमें  $n-2$  elements

में X की search करने में  $T(n-2)$  time लगता है।  
लेकिन  $T(n-1) = K + T(n-2)$  लगता है।

$$T(n-1) = K + T(n-2)$$

after

X  
equation

$$T(n-2) = K + T(n-3)$$

$$T(1) \approx T(n-3)$$

$$T(1) = K$$

$$T(n) + T(n-1) + T(n-2) + T(n-3) = K + T(n-1) + K + T(n-2) + K + T(n-3) + K$$

$$T(n) = 4K \rightarrow x \text{ equation में}$$

$$T(n) = xK$$

$(n) (n-1) (n-2) (n-3) \dots 1$

How many equations  
are there?

5 4 3 2 1

$$\therefore x = n$$

5 eqn हर बार n में से 1 subtract  
हो जाता है और 1 तक जाना है

$$\therefore T(n) = nk$$

$$\therefore T(n) \propto n$$

अगर n1 वाले twice n2

$$\therefore T(n) \text{ is also twice } T(n_2)$$

Time subtraction

negation

### 3. $O(\log n)$ → Logarithmic Time

- \* An algorithm is said to be logarithmic time complexity when it reduces the size of the input data in each step.
- \* This means, that the number of operations is not same as the input size.
- \* The number of operations gets reduced as the input size increases.
- \* The algorithms with Logarithmic Time Complexity are found in Binary Trees or Binary Search Functions.



\* This involves the search of a given value in an array by splitting the array into two and starting the search in one split.

\* This ensures that the operation is not done on every element of the data.

# Example → Binary Search

$$[x = 70] \quad low = 0 \\ high = 9.$$

Value →	10	20	30	40	50	60	70	80	90	100
Index	0	1	2	3	4	5	6	7	8	9.

↑  
low  
  
high ↑

अगर हमें mid निकालना है →  $low + high$

इस n elements में search कर रहे थे

हमें प्रत्येक comparison करने

में इक constant K time लगता

अब वहे दुसरे elements

में  $T(\frac{n}{2})$  time लगेगा

$$\underline{mid} = \frac{2}{2}$$

$$mid = \frac{0+9}{2} = \frac{4.5}{2} = 4.$$

$$50 < 70$$

$$arr[mid] < 70$$

$$mid + 1$$

$$T(n) = K + T\left(\frac{n}{2}\right)$$

तो अब हम low को कर देंगे mid + 1

n elements में

$$low = 5, high = 9.$$

10	20	30	40	50	60	70	80	90	100
0	1	2	3	4	5	6	7	8	9

इस n elements में search कर रहे थे

प्रत्येक comparison करने में इक constant

K time लगता, अब वहे

दुसरे elements के में

search करने के

लिए  $T(\frac{n}{4})$  time

low

high ↑

$$mid = \frac{5+9}{2} = \frac{14}{2} = 7$$

$$arr[mid] = 80.$$

लगेगा क्योंकि अब दुसरे में search

$$T\left(\frac{n}{4}\right) = K + T\left(\frac{n}{4}\right)$$

करेंगे

$$70 < 80.$$

$$70 < arr[mid]$$

तो अब हम high को कर देंगे mid - 1.

$$low = 5$$

$$high = 6$$

10	20	30	40	50	60	70	80	90	100
0	1	2	3	4	5	6	7	8	9

दूसरे  $\frac{n}{4}$  elements को search कर रहे थे  
इसमें मैं 1 comparison करने

में K constant time लगता है

अब दूसरे पास  $\frac{n}{8}$  element

बचे हुए search करना है

$n$  elements में value[mid] = 60.

$\frac{n}{8}$  search करने में  $T\left(\frac{n}{8}\right)$        $60 < 70$ .

time लगता है

$$value[mid] \xleftarrow{x} \left(\frac{n}{8}\right) T + K = \left(\frac{n}{4}\right) T$$

$$T(n) = K + T\left(\frac{n}{4}\right)$$

so we need to increase low  
तो अब low को करके mid + 1.

$$low = 6$$

$$high = 6$$

10	20	30	40	50	60	70	80	90	100
0	1	2	3	4	5	6	7	8	9

$$\frac{10}{8} = \frac{n}{8} \approx 1 = 1.$$

तो इस 1 element

में ही search कर रहे थे

यह compare कर रहे

थे तो 1 element

को compare

करने में time

लगता है

constant K.

$$mid = \frac{low + high}{2} = \frac{6+6}{2} = \frac{12}{2} = 6$$

$$value[mid] = 70.$$

$$70 = 70$$

$$\uparrow \quad \uparrow \\ value[mid] \quad x.$$

$$T(1) = K$$

# परं हमारी Initial Equation है।

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

How  
many  
equations?

$$T(n) = K + T\left(\frac{n}{2}\right)$$



$$T\left(\frac{n}{2}\right) = K + T\left(\frac{n}{4}\right)$$

any no.

$\times$  no. of equations

4 equations



$$T\left(\frac{n}{4}\right) = K + T\left(\frac{n}{8}\right)$$

↓  
any no.



$$T(1) = K$$

Now let's add all these equations

~~$$T(n) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T(1) = K + T\left(\frac{n}{2}\right) + K +$$~~

~~$$T\left(\frac{n}{8}\right)$$~~

~~$$+ K$$~~

$$T(n) = K + K + K + K$$

$$T(n) = 4K$$

no of equation  
formed =  $x = (1)$

$$T(n) = xK$$

How many equations will be formed?

GP

$$\boxed{n \quad \frac{n}{2} \quad \frac{n}{4} \quad \dots \quad 1}$$

$x$ th term in GP is 1 here.

2) GP (Geometric Progression) ↗

Here,  
Given → first element

$$a = n$$

$$r = \frac{1}{2}$$

→ gap/move

find  $a_x \rightarrow$  मात्रा वर्ग GP की  $x$ th term (दृष्टि)

How to calculate  $a_x$  ( $x$ th term) in GP.

$$a_x = a r^{x-1}$$

$$1 = n \left(\frac{1}{2}\right)^{x-1}$$

$$a = n$$

$$r = \frac{1}{2}$$

$$a_x = 1$$

$$1 = n \frac{\left(\frac{1}{2}\right)^{x-1}}{(2)^{x-1}}$$

→ 1

$$1 = \frac{n}{(2)^{x-1}}$$

$$(2)^{x-1} = n$$

उपर्युक्त लिखित दोनों सिद्ध

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

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

$$(x-1) \log_2(2^1) = \log_2(n)$$

$$(x-1) \underbrace{\log_2 2}_\text{value 1} = \log_2 n$$

$\log_2$  की value 1 है

$$(x-1) 1 = \log_2 n.$$

$$x-1 = \log_2 n.$$

$$x = \log_2 n + 1$$

तो यहाँ equation होगी  $x = \log_2 n + 1$ .

Binary Search की Time Complexity हो

$$T(n) = \underline{x} * K$$

$$\log_2 n + 1$$

$$T(n) = (\log_2 n + 1) \cdot K$$

$$T(n) = K \cdot (\log_2 n + 1)$$

$$T(n) = K \log n + K$$

$$T(n) \propto \log n$$

Type	Initial Equation	Final Equation	Example
$O(\log n)$	$T(n) = K + T\left(\frac{n}{2}\right)$	$T(n) \propto \log n$	eg-1 Binary Search
			eg-2 Binary Tree

अगर Initial Equation है  $\rightarrow T(n) = K + T\left(\frac{n}{2}\right)$   
 तो final Eq. होती है  $T(n) \propto \log n$

Eg 1)  $n_1 = 4n_2$

अगर n का value  $n_2$  से 4 गुणी है  
 तो Time वही 2 गुणी होगा  $T(n_2)$  के

$$T(n) \propto \log n$$

$$T(n_1) \propto \log(4n_2) = 2T(n_2)$$

$$T(n_1) = 2T(n_2)$$

Eg 2)  $n_1 = 1024 n_2 = 2^{10} n_2$

$$T(n) \propto \log n$$

$$T(n_1) \propto \log(2^{10} n_2)$$

अगर n 1024 times हो जाए  $n_2$  के  
 तो Time तो केवल 10 times ही  
 हो जाए  $T(n_2)$  के

$$T(n_1) = 10T(n_2)$$

Ex 3)

$$n_1 = 10^9 (n_2)$$

$$T(n) \propto \log(n)$$

$$T(n_1) = T(10^9 n_2)$$

$$10^9 = 2^{30}$$

इसको 2 की Power  
की terms  
में express  
करो

$$T(n_1) = T(2^{30} n_2)$$

$$\therefore T(n_1) \propto \log_2 (2^{30} n_2)$$

$$T(n_1) = 30 T(n_2)$$

जैसे अगर  $n_1$  की value  $n_2$  के  $10^9$  times हो।  
Time consumed by  $n_1$   $T(n_1)$  के 30 times हो।

जैसे किसी  $2^n$  algorithm की time complexity  $O(\log n)$  हो।

जैसे किसी अच्छी algorithm मानी

जार्जरी क्योंकि INPUT बहुत ज्यादा

बढ़ने पर भी time बहुत उपरा

जाए बढ़ता।

e.g. → Input  $10^9$  item कर दिया  
Time 30 items ही बढ़ा

$$T(n_1) \propto \log(n_1)$$

$$T(n_2) \propto \log(n_2)$$

$$T(n_1) = K \log(n_1)$$

$$T(n_2) = K \log(n_2)$$

जब हम  
proportional  
का sign  
होता है  
तो K constant  
लगाते हैं.

अब Divide ~~दें~~  $T(n_1)$  by  $T(n_2)$

$$\frac{T(n_1)}{T(n_2)} = \frac{K \log(n_1)}{K \log(n_2)}$$

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(n_1)}{\log(n_2)}$$

$$n_1 = 1024 n_2$$

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(1024 n_2)}{\log(n_2)}$$

$$1024 = 2^{10} \rightarrow 1024 \text{ को } 2 \text{ की Power of}$$

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(2^{10} n_2)}{\log(n_2)}$$

terms में represent  
कर

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(1024 n_2)}{\log(n_2)}$$

$$1024 = 2^{10}$$

$$\frac{T(n_1)}{T(n_2)} = \frac{\log(2^{10} n_2)}{\log(n_2)}$$

$$\frac{T(n_1)}{T(n_2)} = 10 \frac{\log n_2}{\log n_2}$$

$$T(n_1) = 10 T(n_2)$$

This transition will be explained in next class

CLASSMATE  
Date \_\_\_\_\_  
Page \_\_\_\_\_

$O(\log_2(n))?$

for loop →  $O(n)$  ?

हमें कैसे पता चलेगा for loop की Time Complexity  $O(n)$  है या  $O(\log n)$  ?

जब Time Complexity  $O(\log n)$  है

तो अगर Input  $2^3 = 8$  तो भी होगा  
तो भी Time 3 times हो जाएगा।

---

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

// do some work

}

$$T(n) = kn$$

$\{ O(n)$

---


$$T(n) = k + T(n-1) = T(n) \propto n$$

```

for (int i = n; i >= 1; i--) )  

{  

    // do work  

}

```

$T(n) = kn$

$O(n)$

## MOST IMP TRICKY FOR LOOP

```

for (int i = n; i >= 1; i = i/2) )  

{  

    do some work  

}

```

$T(n) = K + T\left(\frac{n}{2}\right)$

$O(\log n)$

$\rightarrow T(n) \propto \log(n)$

Initial Eq<sup>n</sup>

Final Eq<sup>n</sup>

छाले # एमी 3 Program जैसा होगा, जो अपनी Time Complexity घटानी होगी.

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

The Program is written to find the Power of a number

$$\rightarrow x^n \rightarrow ?$$

## WAY 1

→ public static void power (int x, int n)

{ if ( $n == 0$ )

    return 1;

(0 == n)  $\Rightarrow$   
Base Case

x की Power n minus 1.

if  $n > 0$   
else if  $n < 0$

t(n-1)

int  $x^{p(n-1)} = \text{power}(x, n-1);$

Recursive  
call to

K  
constant  
time

int  $x^p = x^{p(n-1)} * x;$

find  
 $x^{n-1}$

return  $x^p;$

x की Power n minus 1.

$x^{n-1} * x$

$$t(n) = K + t(n-1).$$

$$t(n) \propto n$$

if  $n > 0$

Self

work

part

# Way 2 to find $x^n$

public static void power (int x, int n)

```
if (n == 0)  
{ return 1; }  
Base Case
```

$t\left(\frac{n}{2}\right) \leftarrow$  int  $x^{pn} b_2 = \text{power}(x, n/2);$  ] Faith

int  $x^{pn} = x^{pn} b_2 * x^{pn} b_2;$

```
if (n % 2 == 1)
```

BTJR

$$x=2$$

$$n=4$$

$$x^n = x^{\frac{n}{2}} + x^{\frac{n}{2}}$$

Self work

$$x^n = 4 * 4$$

$$x^n = 16$$

$$x^{pn} = x^{pn} * x;$$

K  
constant time

return  $x^{pn};$

BTJR  $x=2, n=5,$

$$x^n = x^{5/2} * x^{5/2} * x$$

$$= x^2 * x^2 * x$$

$$= 4 * 4 * 2$$

$$t(n) = k + t\left(\frac{n}{2}\right)$$

$$t(n) \propto \log n$$

# Way 3 to find $x^n$

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Recursive static void power (int x, int n)

{ if ( $n == 0$ )

{ return 1;

}

if ( $n \cdot 2 == 0$ )

{ return power ( $x, n/2$ ) \* power ( $x, n/2$ ); }

$t(n/2)$

$t(n/2)$

else

{ return power ( $x, n/2$ ) \* <sup>0</sup>power ( $x, n/2$ ) \*  $x^k$ ; }

$t(n/2)$

$t(n/2)$

}

$k$

$O(\log n)$

$t(n)$

$t(n/2)$

$O(n)$

$t(n)$

$$t(n) = t\left(\frac{n}{2}\right) + t\left(\frac{n}{2}\right) + k$$

$$t(n) = k + 2t\left(\frac{n}{2}\right)$$

Add all eq?

$$2^0 t(n) + 2^1 t\left(\frac{n}{2}\right) + 2^2 t\left(\frac{n}{4}\right) + 2^3 t\left(\frac{n}{8}\right) + \dots + t(1)$$

$$= 2K + K + 2K + 2^2 K + 2^3 K + \dots + 2t\left(\frac{n}{2}\right) + t\left(\frac{n}{4}\right)$$

$$+ 2t\left(\frac{n}{8}\right)$$

Multiply by  $2^0$

$$2^0 t(n) = 2K + 2^1 t\left(\frac{n}{2}\right)$$

Multiply by  $2^1$

$$2^1 t\left(\frac{n}{2}\right) = -2K + 2^2 t\left(\frac{n}{4}\right)$$

Multiply by  $2^2$

$$2^2 t\left(\frac{n}{4}\right) = 2^2 K + 2^3 t\left(\frac{n}{8}\right)$$

Multiply by  $2^{n-1}$

$$2^{n-1} t(1) = 2K$$

$$+ 2^0 t(n) = 4K$$

Assume there were  $4$  eqn.

$$t(n) = K + (2K + (2^2 K + (2^3 K + \dots + 2^{n-2} K))$$

$$= K + (2^{n-1} K)$$

$$\text{GP sum} = \frac{a(r^n - 1)}{r - 1}$$

Initial Eq<sup>n</sup>

$$t(n) = K + 2t\left(\frac{n}{2}\right)$$

$$t\left(\frac{n}{2}\right) = K + 2t\left(\frac{n}{4}\right)$$

$$t\left(\frac{n}{4}\right) = K + 2t\left(\frac{n}{8}\right)$$

$$t(1) = K$$

cancel out  
by  $2t$

$x$  eq<sup>n</sup> so  
use

need to

multiply first  
eq<sup>n</sup> by 2<sup>0</sup>, next  
by 2<sup>1</sup>

& soon  $2^{x-1}$

$$\rightarrow 2^0 t(n) = 2^0 K + 2^0 * \left(2t\left(\frac{n}{2}\right)\right)$$

$$\rightarrow 2^1 t\left(\frac{n}{2}\right) = 2^1 K + 2^1 \left(2t\left(\frac{n}{4}\right)\right)$$

$$\rightarrow 2^2 t\left(\frac{n}{4}\right) = 2^2 K + 2^2 \left(2t\left(\frac{n}{8}\right)\right)$$

$$\rightarrow 2^{x-1} (t(1)) = 2^{x-1} K$$

Assume  
that  
there  
are  
 $x$   
eq<sup>n</sup>  
& these  
are  
 $x$  eq<sup>n</sup>



$$\rightarrow 2^0 t(n) = 2^0 (K) + 2t\left(\frac{n}{2}\right)$$

$$\rightarrow 2^1 t\left(\frac{n}{2}\right) = 2K + 2^2 t\left(\frac{n}{4}\right)$$

$$\rightarrow 2^2 t\left(\frac{n}{4}\right) = 2^2 K + 2^3 t\left(\frac{n}{8}\right)$$

$$\rightarrow 2^{x-1} (t(1)) = 2^{x-1} K$$

assume  $2^3 t\left(\frac{n}{8}\right)$

$$\therefore t(n) = 2^0 k + 2^1 k + 2^2 k + 2^{x-1} k$$

GP का Sum =  $a \frac{(r^n - 1)}{r - 1}$

first term  
 $2^x$  से इसका रूप है  
or here  
is ratio

$$t(n) = \frac{k(2^x - 1)}{2 - 1}$$

प्राप्ति करते हैं  
इसका रूप है  
प्राप्ति है

$$t(n) = \frac{k(2^x - 1)}{1}$$

K,  $2^1 2^2$   
multiply it  
रहते हैं

$$t(n) = K(2^x - 1)$$

$$t(n) = 2^x K - K$$

$$x = \log_2 n + 1$$

In any GP, x की value की दोनों ओरी बराबर है।

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

$$t(n) = K(2^{\log_2 n}) \cdot (2^1) - K$$

$$= 2^{\log_2 n} K - K$$

$$\Rightarrow 2^{\log_2 n} K - K$$

$$t(n) \Rightarrow K(2n - 1)$$

$t(n) \propto n$

Way 3 में  $O(n)$   
Time Complexity