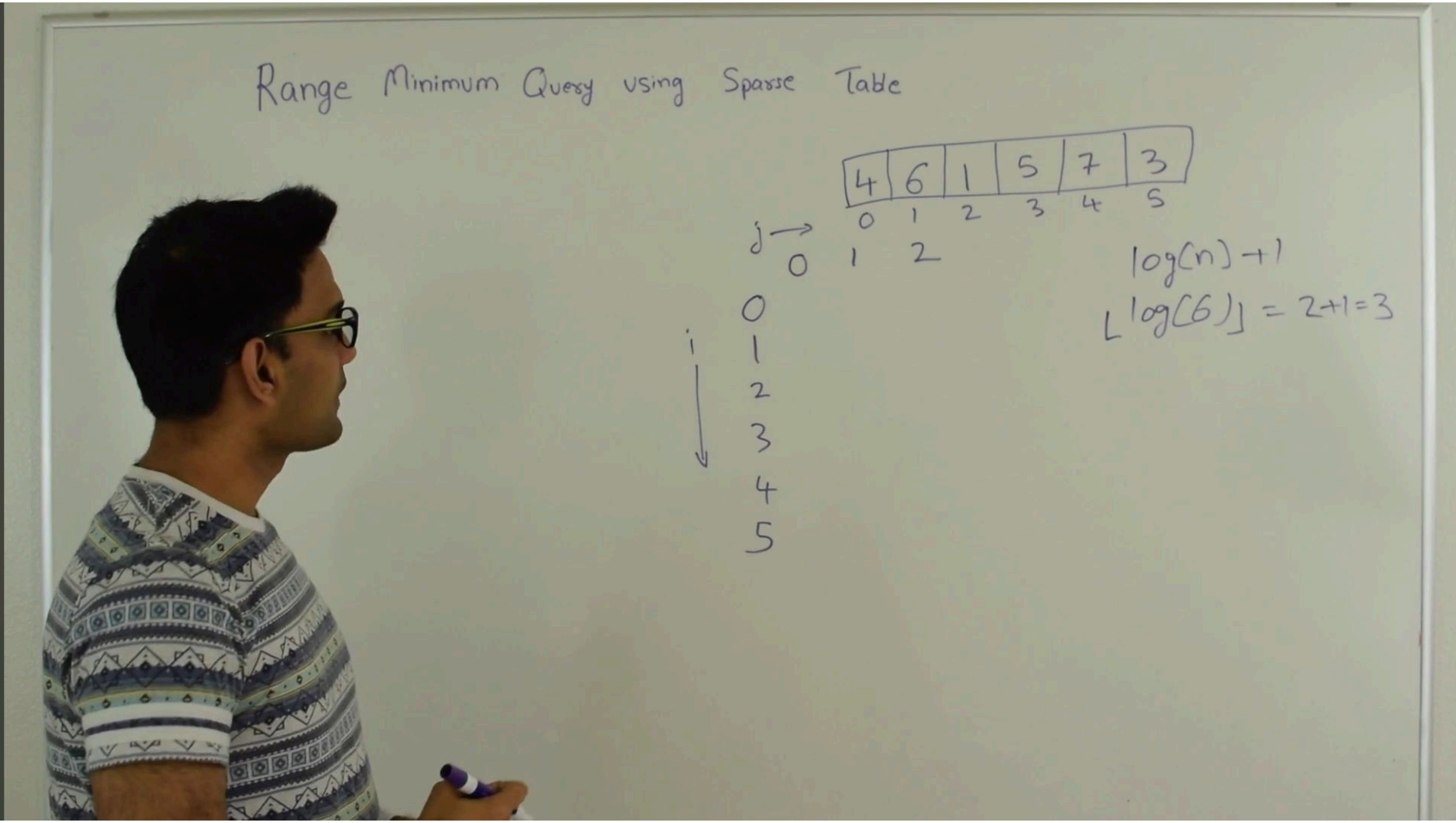
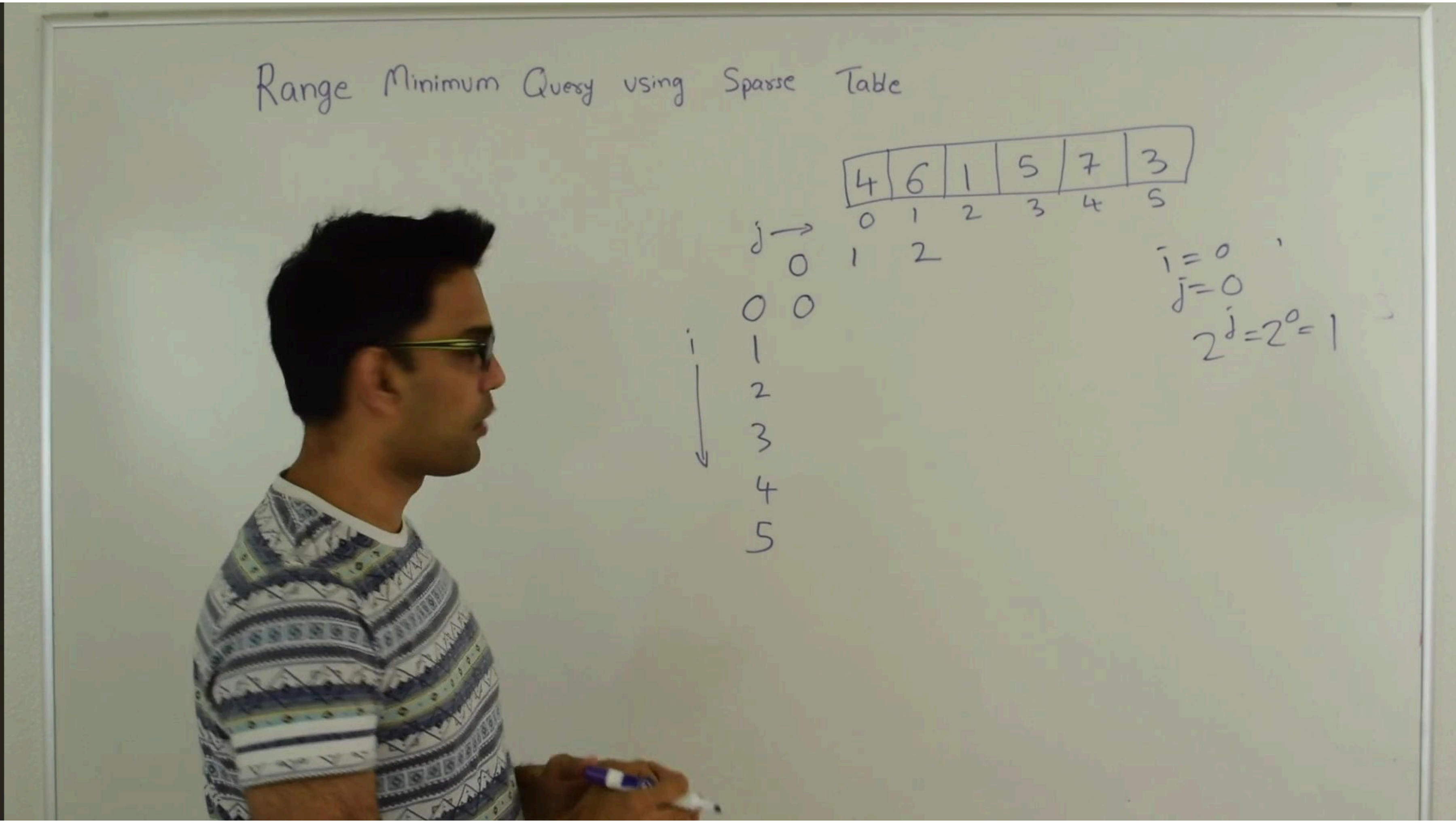
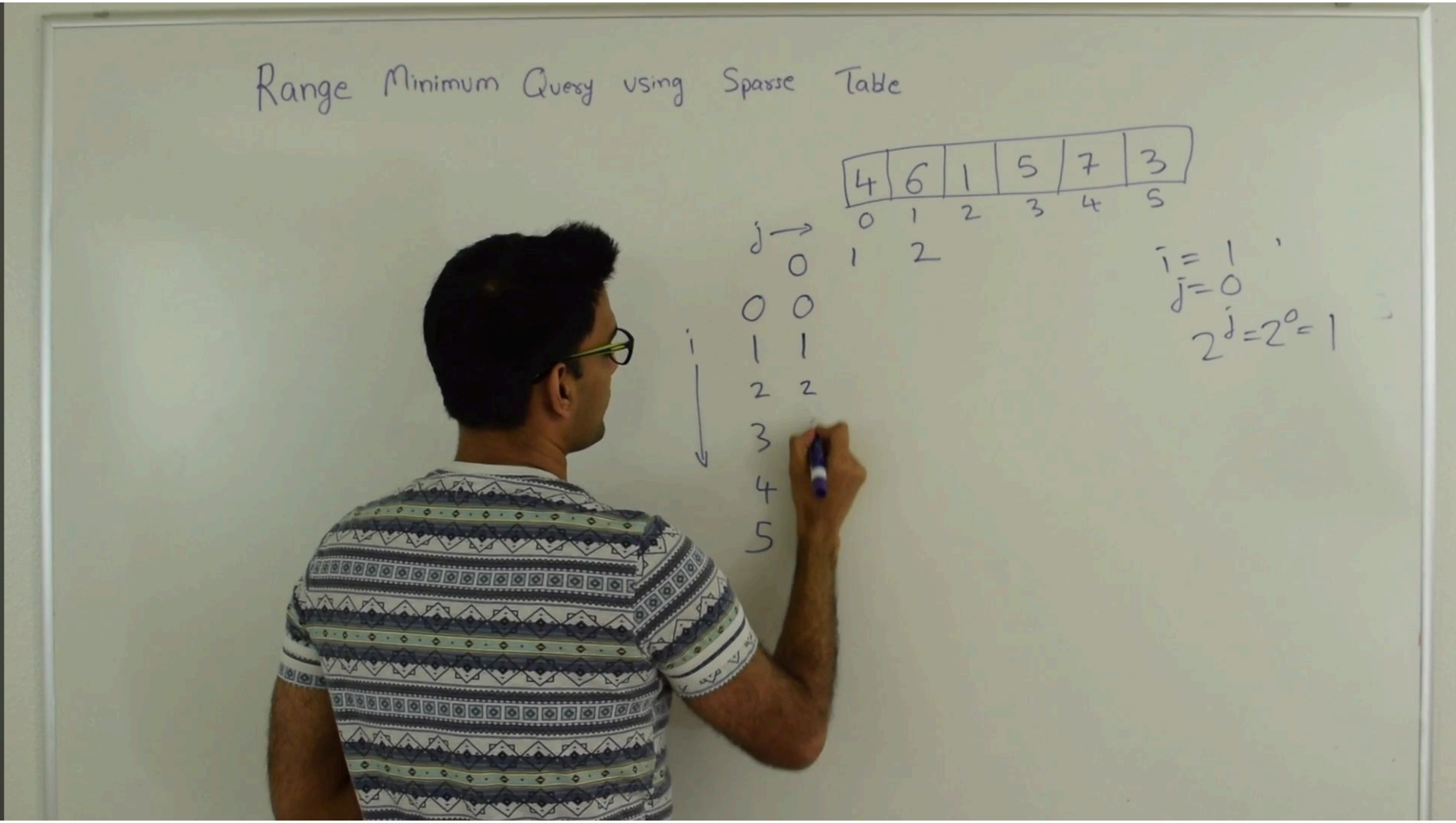
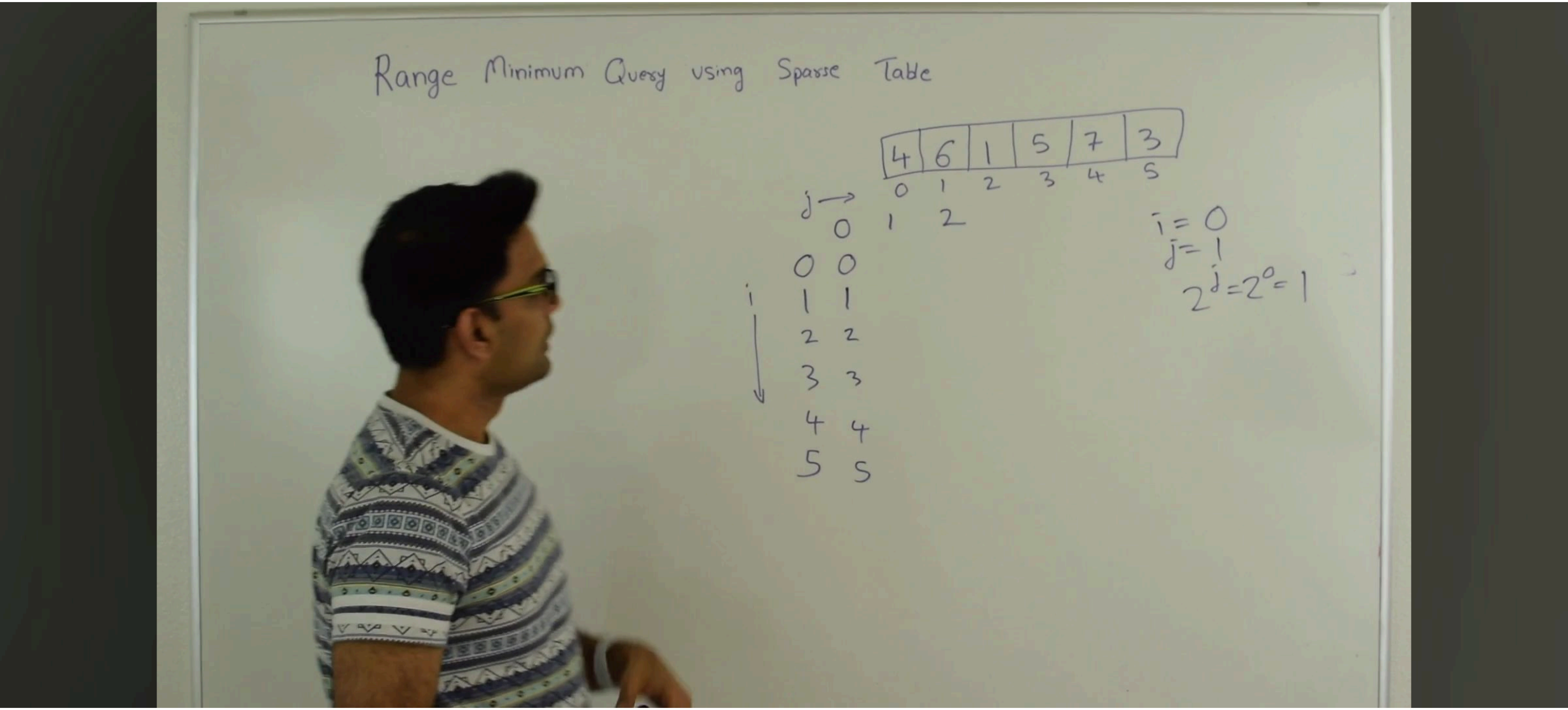
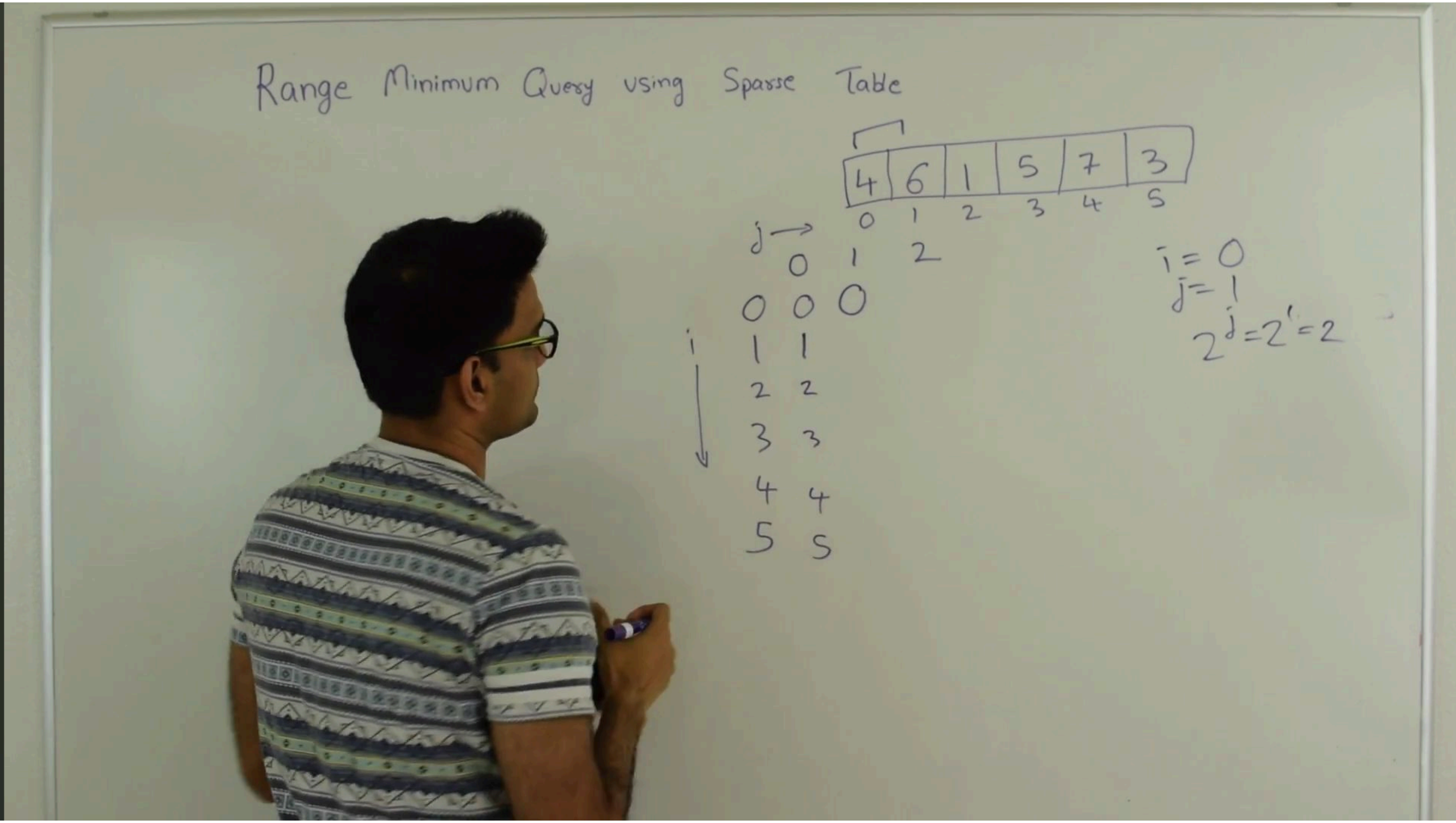
Sparse Table algorithm

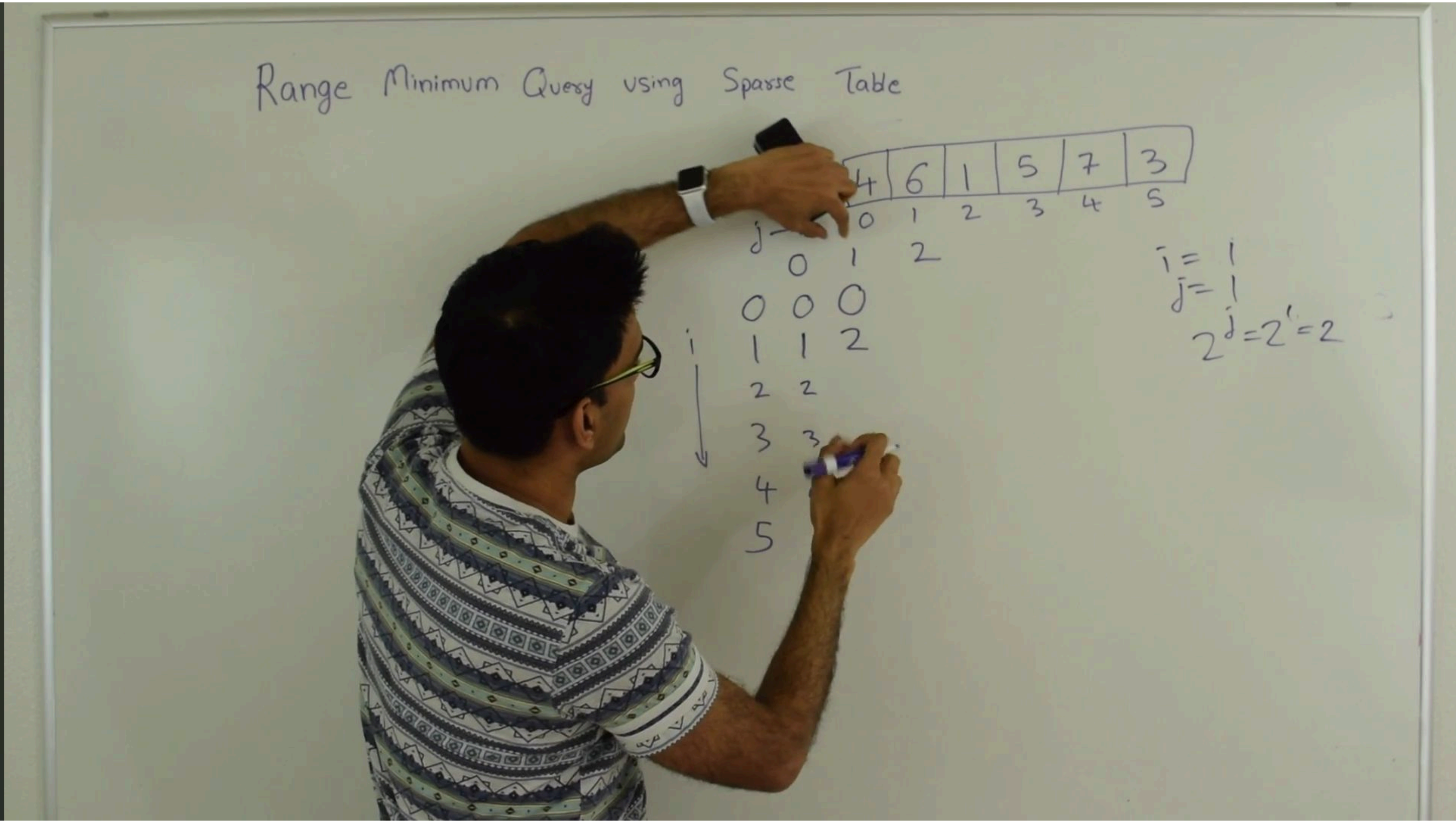

Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0
1
2
3
4
5
log(n) +1
⌊log(6)⌋ = 2+1=3


Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0 0
1
2
3
4
5
i = 0
j = 0
2^j = 2^0 = 1

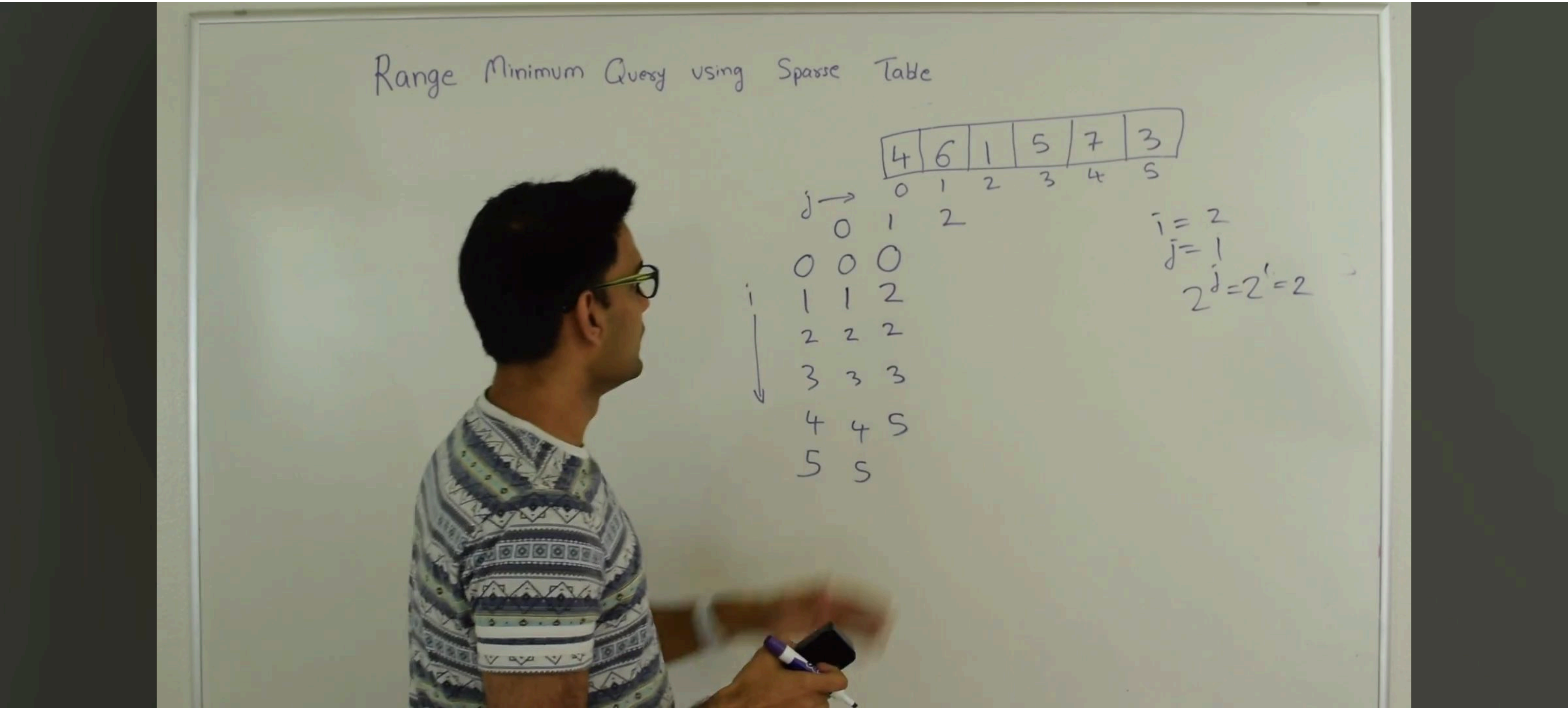

Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0 0
1 1
2 2
3
4
5
i = 1
j = 0
2^j = 2^0 = 1

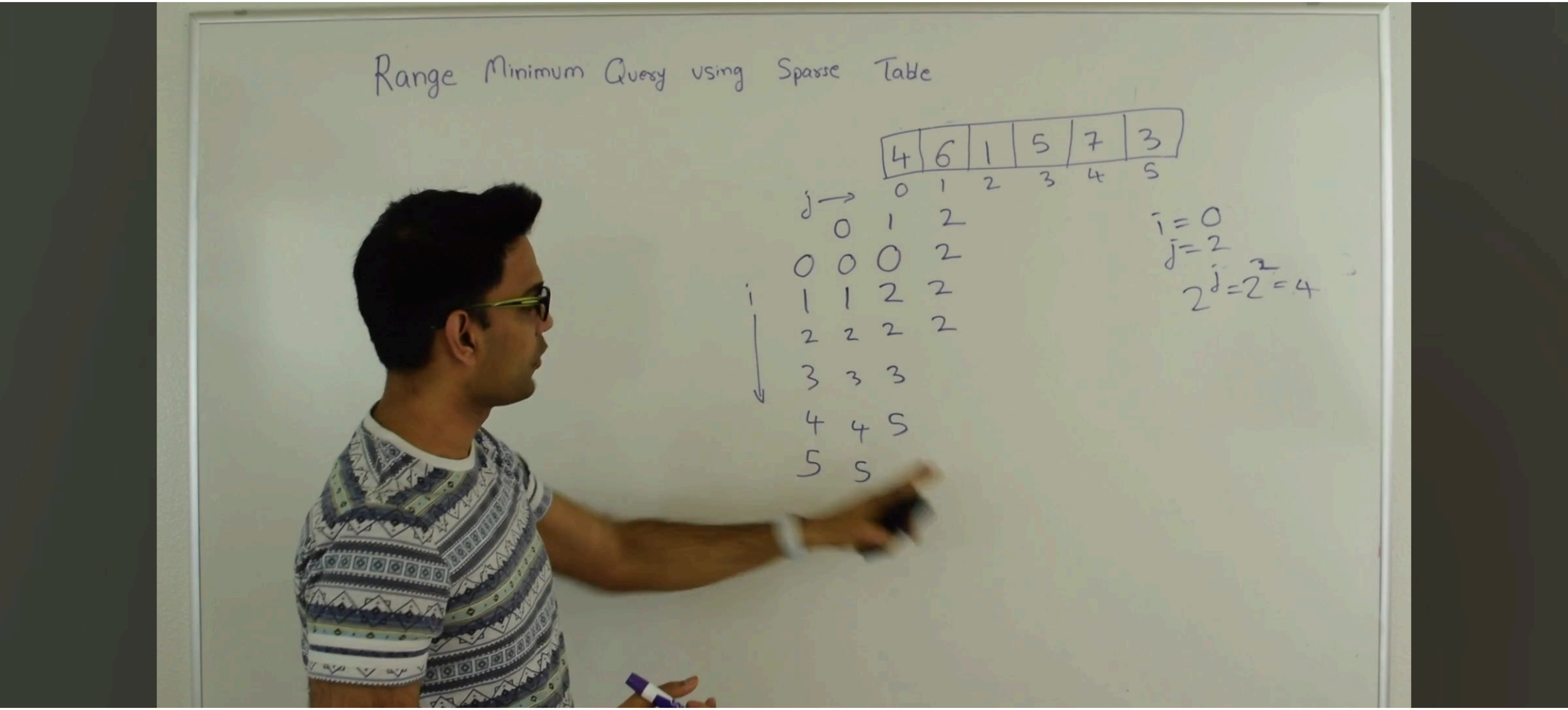

Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0 0
1 1
2 2
3 3
4 4
5 5
i = 0
j = 1
2^j = 2^0 = 1

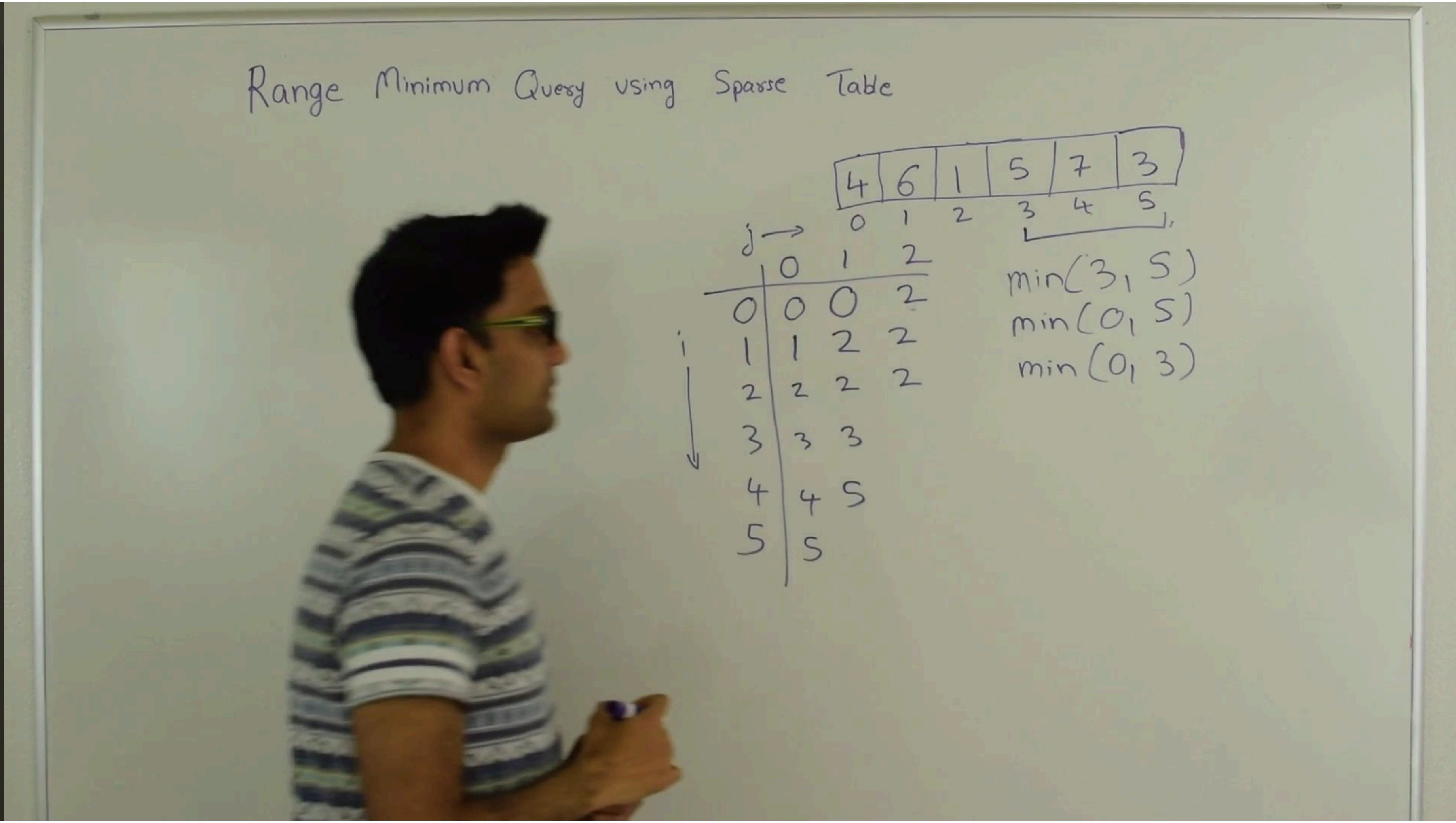

Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0 0 0
1 1
2 2
3 3
4 4
5 5
i = 0
j = 1
2^j = 2^1 = 2

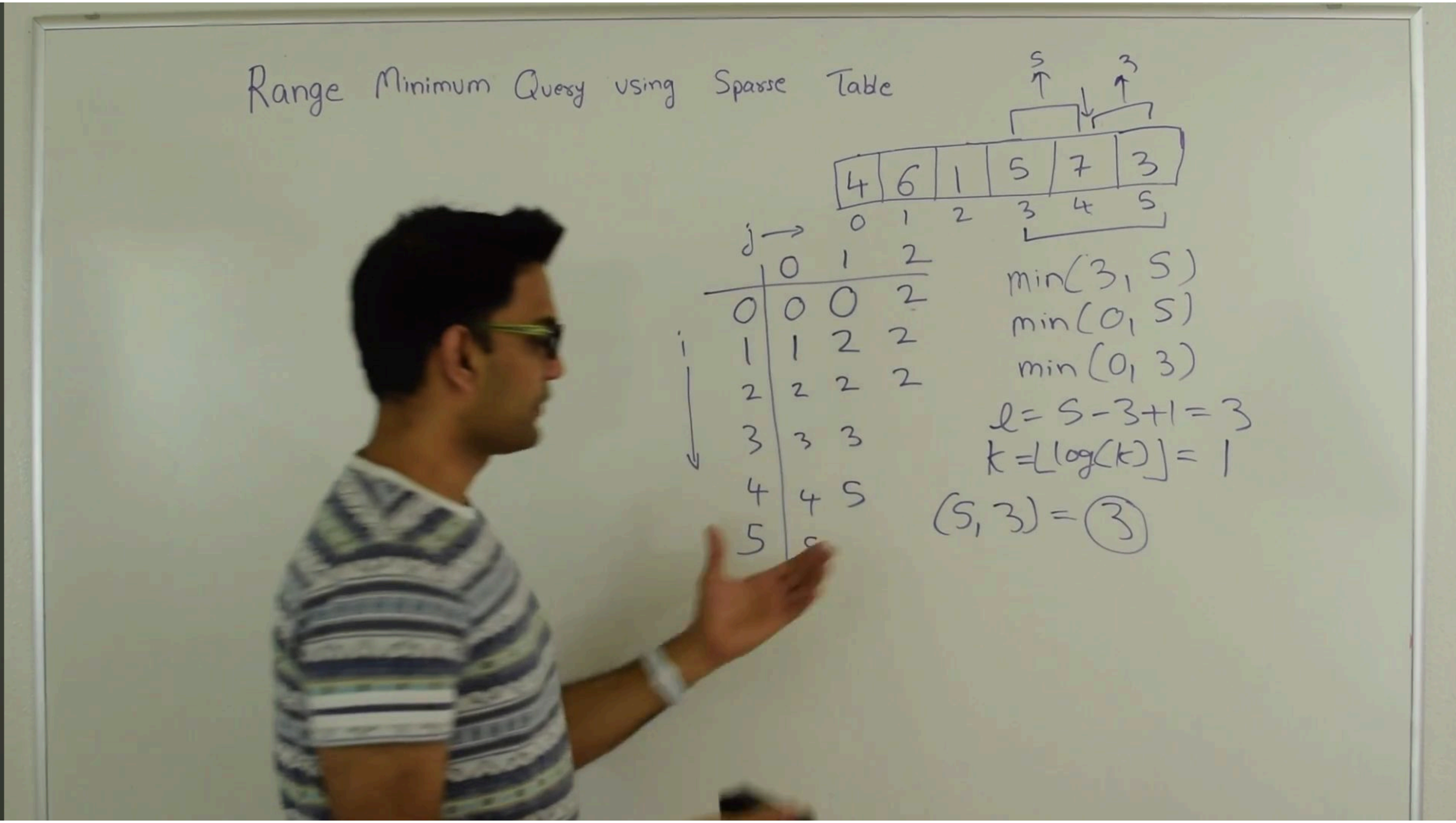


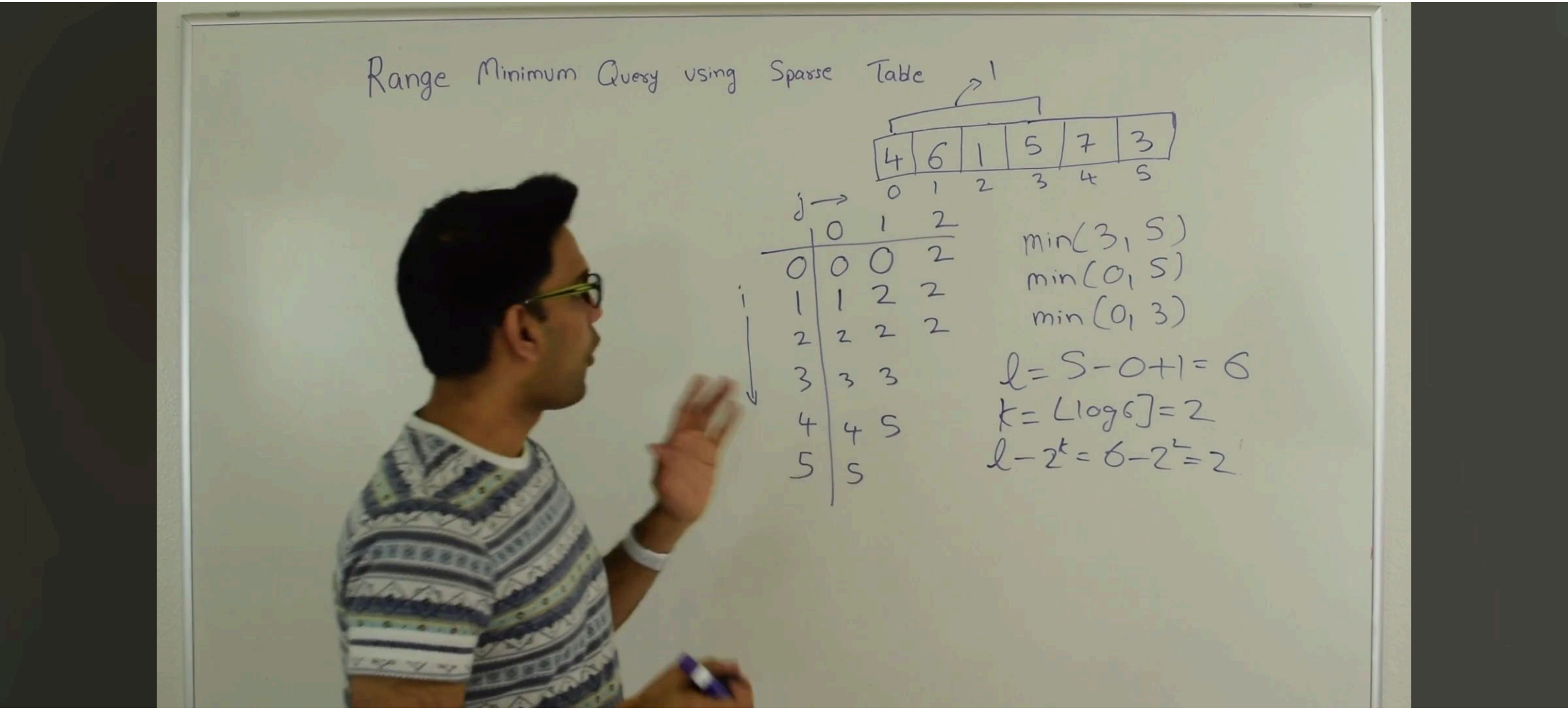
Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
i = 1
j = 1
2^j = 2^1 = 2

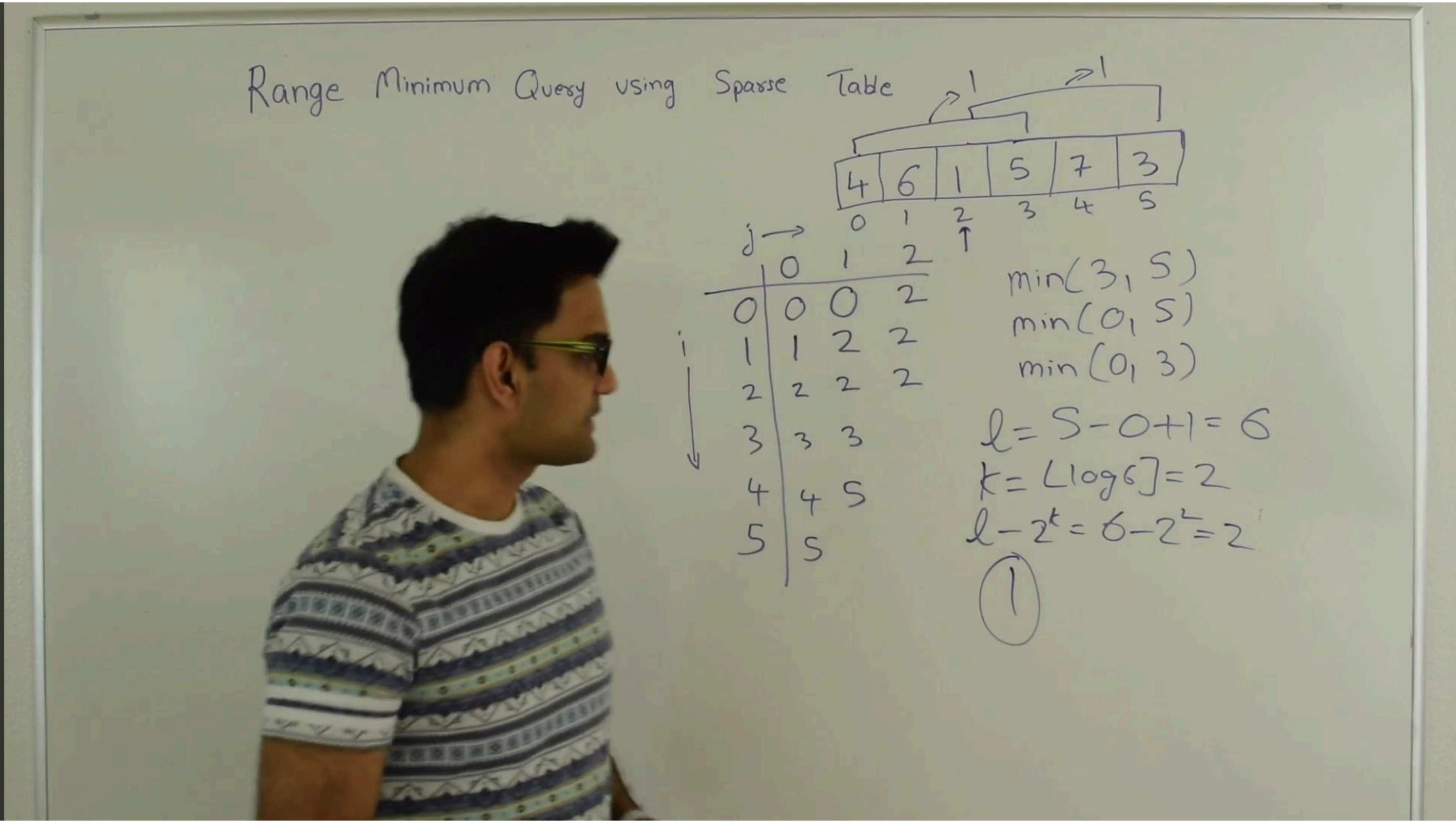

Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
i
0 0 0
1 1 2
2 2 2
3 3 3
4 4 5
5 5
i = 2
j = 1
2^j = 2^1 = 2


Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
0 0 0 2
1 1 2 2
2 2 2 2
3 3 3
4 4 5
5 5
i
i = 0
j = 2
2^j = 2^2 = 4


Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
0 | 0 0 2
1 | 1 2 2
2 | 2 2 2
3 | 3 3
4 | 4 5
5 | 5
i
min(3, 5)
min(0, 5)
min(0, 3)


Range Minimum Query using Sparse Table
5
3
4 6 1 5 7 3
0 1 2 3 4 5
j
0 1 2
0 0 0 2
1 1 2 2
2 2 2 2
3 3 3
4 4 5
5 5
i
min(3, 5)
min(0, 5)
min(0, 3)
l = 5 - 3 + 1 = 3
k = ⌊log(k)⌋ = 1
(5, 3) = 3


Range Minimum Query using Sparse Table
1
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
0 | 0 0 2
1 | 1 2 2
2 | 2 2 2
3 | 3 3
4 | 4 5
5 | 5
i
min(3, 5)
min(0, 5)
min(0, 3)
l = 5-0+1 = 6
k = ⌊log6⌋ = 2
l-2^k = 6-2^2 = 2


Range Minimum Query using Sparse Table
4 6 1 5 7 3
0 1 2 3 4 5
j →
0 1 2
0 | 0 0 2
1 | 1 2 2
2 | 2 2 2
3 | 3 3
4 | 4 5
5 | 5
i
min(3, 5)
min(0, 5)
min(0, 3)
l = 5 - 0 + 1 = 6
k = ⌊log 6⌋ = 2
l - 2^k = 6 - 2^2 = 2
1

Fenwick tree

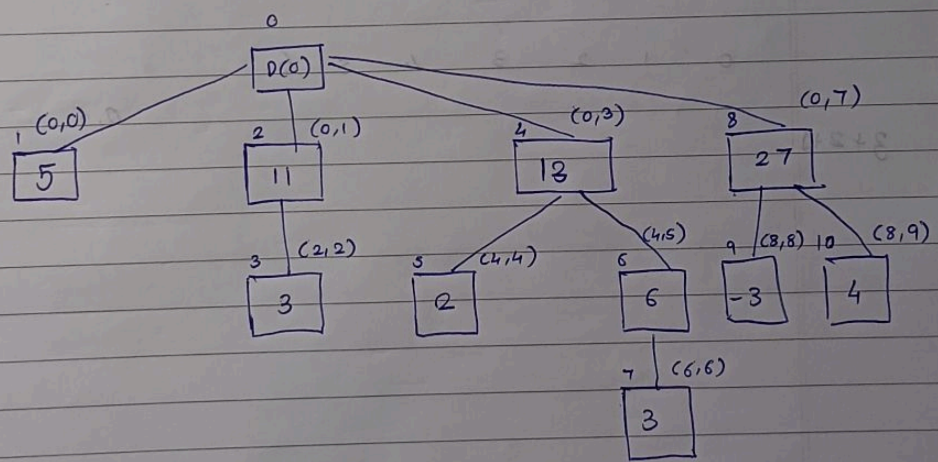
5	6	3	-1	2	4	3	5	-3	7	← array
0	1	2	3	4	5	6	7	8	9	← indices

Fenwick Tree

* To find Parent Node (flip Right most 1)

			Parent
1	0001	0000	0
2	0010	0000	0
3	0011	0010	2
4	0100	0000	0
5	0101	0100	4
6	0110	0100	4
7	0111	0110	6
8	~~0~~1000	0000	0
9	1001	1000	8
10	1010	1000	8

8 4 2 1



Theory

no. of elements to be considered from start index

Tree Index

1 — $0 + 2^0 \rightarrow (0,0)$

start index in array

2 — $0 + 2^1$ — $(0,1)$
3 — $2^1 + 2^0$ — $(2,2)$
4 — $0 + 2^2$ — $(0,3)$
5 — $2^2 + 2^0$ — $(4,4)$
6 — $2^2 + 2^1$ — $(4,5)$
7 — $[2^2 + 2^1] + 2^0$ — $(6,6)$
8 — $0 + 2^3$ — $(0,7)$
9 — $2^3 + 2^0$ — $(8,8)$
10 — $2^3 + 2^1$ — $(8,9)$

* Prefix Sum

$(0,3) = 3+1 = 4 \rightarrow$ starting tree node no. (Index (4) $\xrightarrow{\text{Parent}}$ Node(0)) from tree

$(0,3) = 13$

$(0,8) = \text{index}(9) \xrightarrow{\text{Parent}} \text{Node}(8) \xrightarrow{\text{Parent}} \text{Node}(0)$

$= -3 + 27$

$= 24$

$(0,5) = \cancel{2+15}$ $\text{index}(6) \rightarrow \text{Node}(4) \rightarrow \text{Node}(0)$

$= \cancel{15}$ $6 + 13$

$= 19$

$(0,6) = \text{index}(7) \rightarrow \text{node}(6) \rightarrow \text{Node}(4) \rightarrow \text{Node}(0)$

$= 3 + 6 + 13$

$= 22$

Knapsack Problem

Knapsack Problem

objects	ob_1	ob_2	ob_3
Profit	25	24	15
Weight	18	15	10

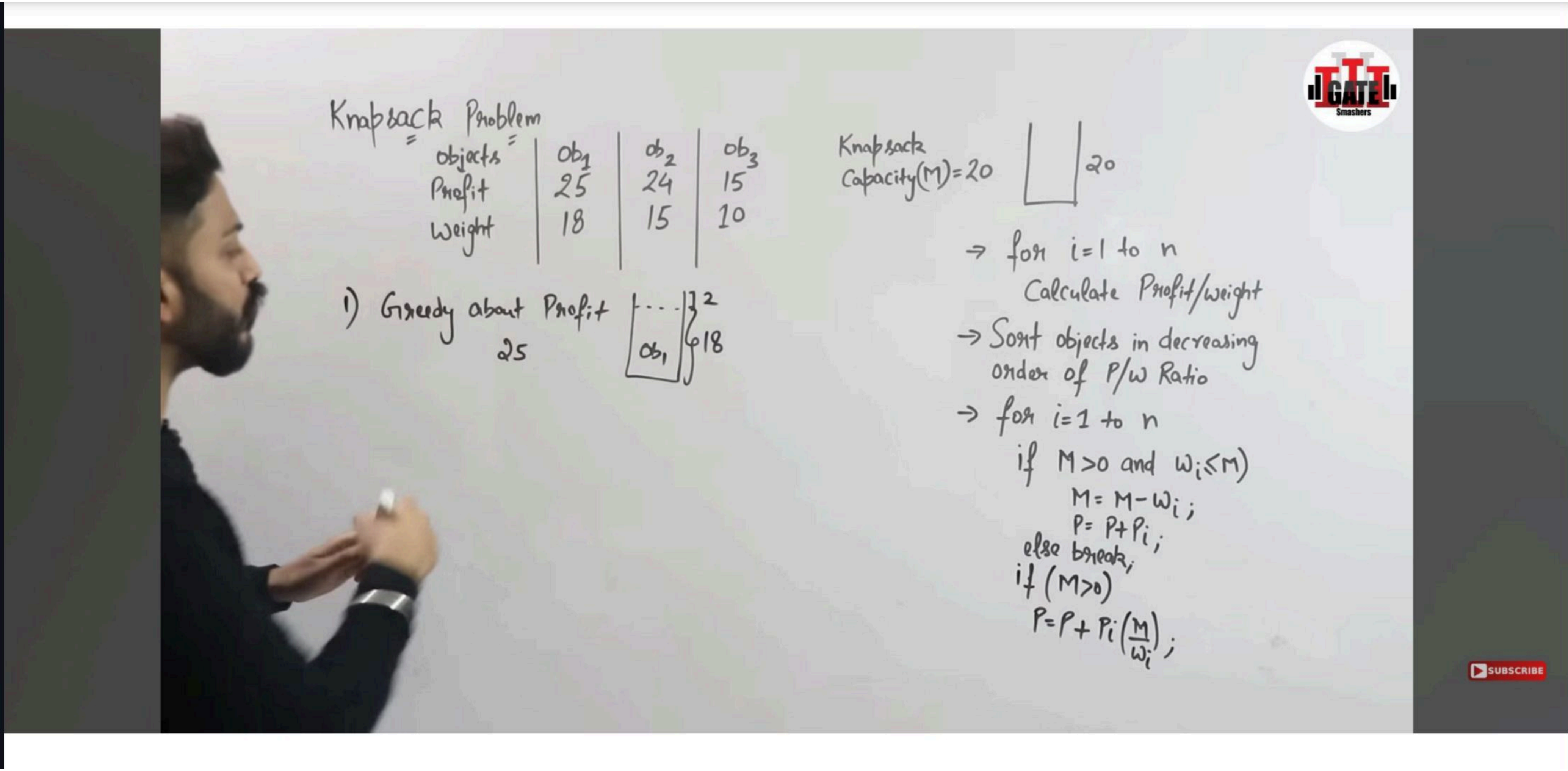
Knapsack Capacity(M) = 20

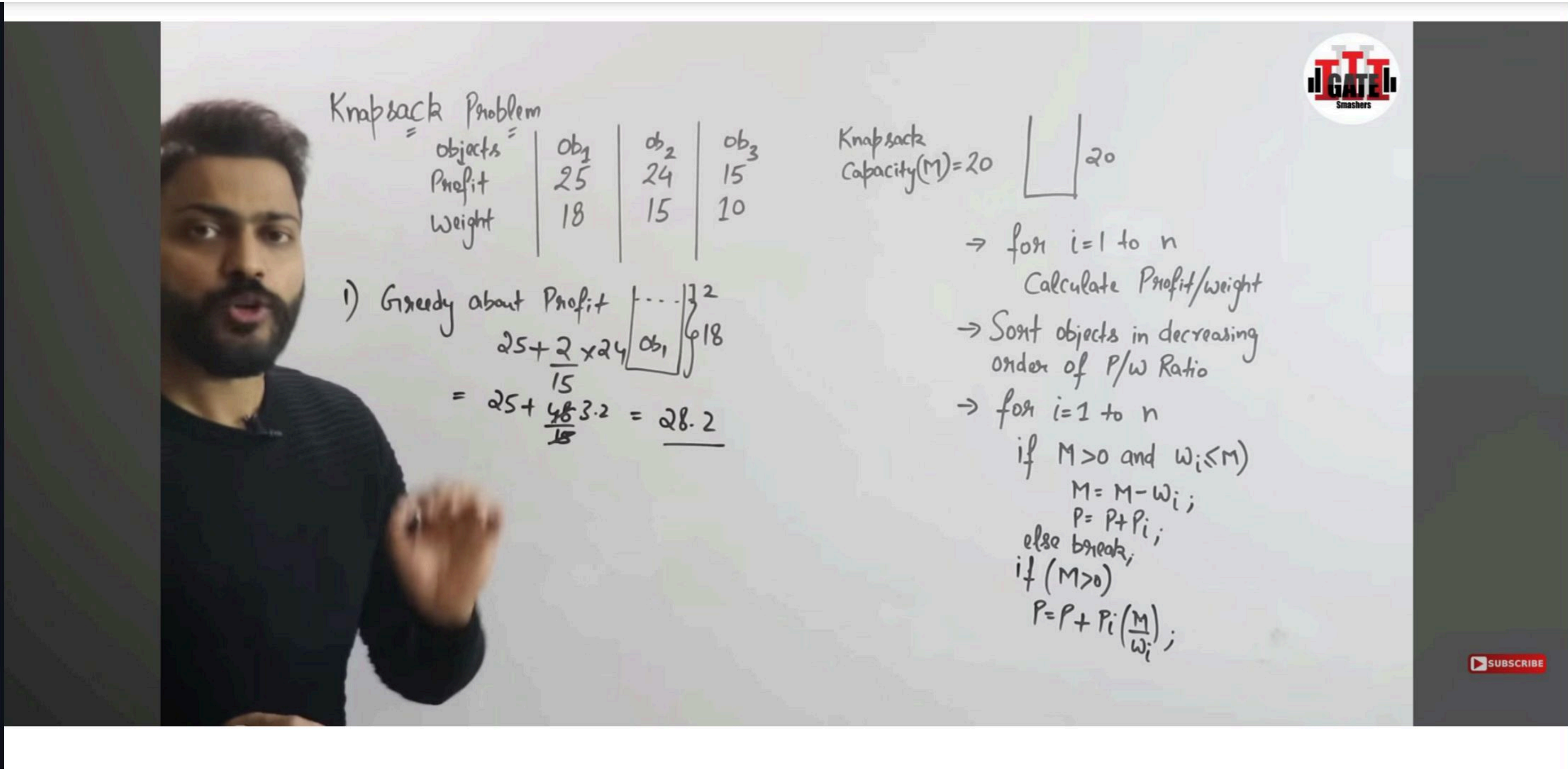
20

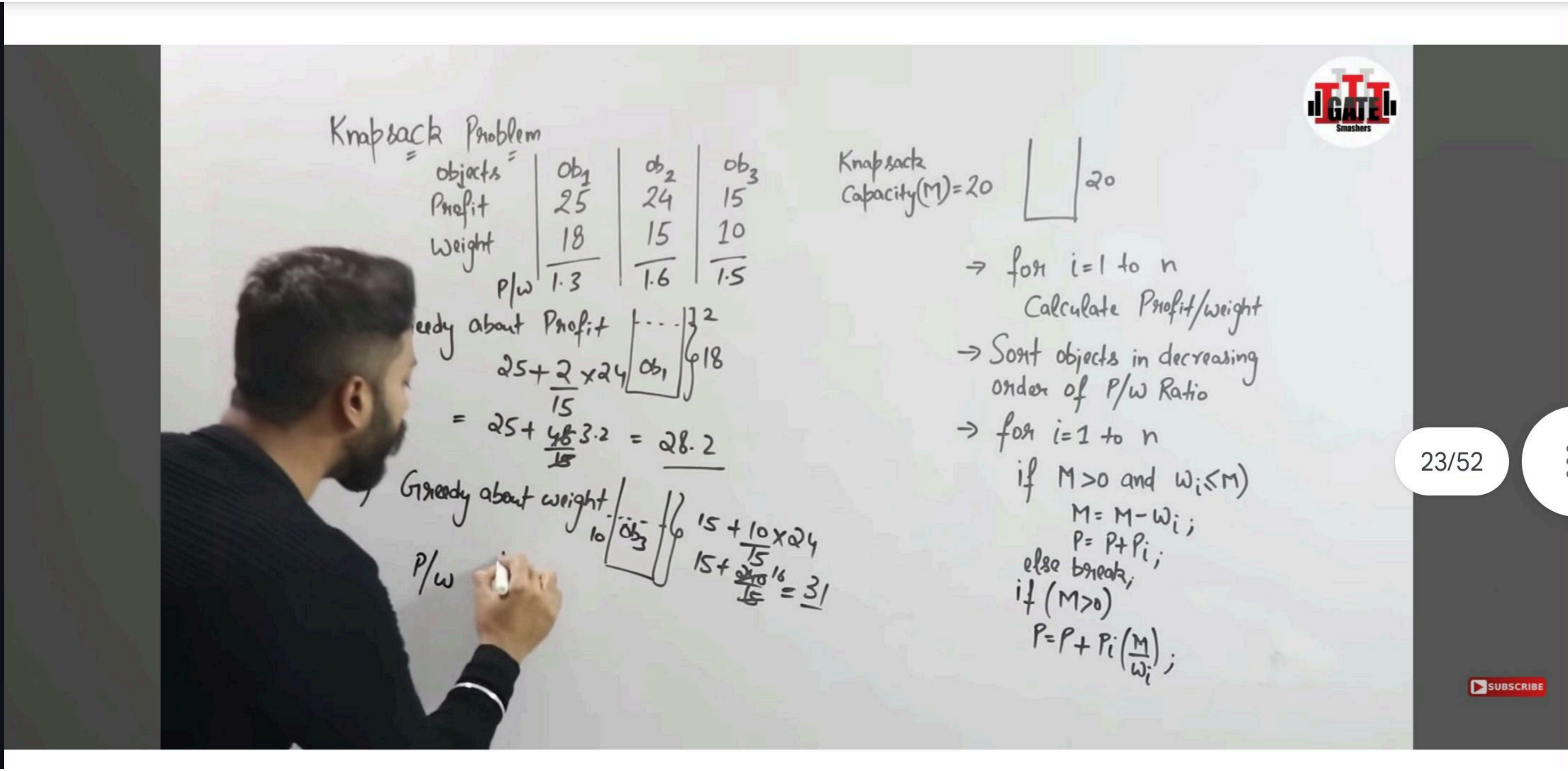
→ for i=1 to n
  Calculate Profit/weight

→ Sort objects in decreasing order of P/ω Ratio

→ for i=1 to n
  if M>0 and $w_i \le M$)
    $M = M - w_i$;
    $P = P + P_i$;
  else break;
  if (M>0)
    $P = P + P_i\left(\frac{M}{w_i}\right)$;


Knapsack Problem
objects | ob1 | ob2 | ob3
Profit | 25 | 24 | 15
Weight | 18 | 15 | 10
Knapsack Capacity(M) = 20
20
1) Greedy about Profit
25
ob1
2
18
→ for i=1 to n
Calculate Profit/weight
→ Sort objects in decreasing order of P/w Ratio
→ for i=1 to n
if (M>0 and wi≤M)
M = M−wi;
P = P+Pi;
else break;
if (M>0)
P=P+Pi(M/wi);
SUBSCRIBE
GATE Smashers


Knapsack Problem
objects | ob_1 | ob_2 | ob_3
Profit | 25 | 24 | 15
Weight | 18 | 15 | 10
Knapsack Capacity(M) = 20
20
1) Greedy about Profit
ob_1
2
18
$25 + \frac{2}{15} \times 24$
$= 25 + \frac{\cancel{48}}{\cancel{15}} 3.2 = \underline{28.2}$
→ for i = 1 to n
Calculate Profit/weight
→ Sort objects in decreasing order of P/w Ratio
→ for i = 1 to n
if M > 0 and $w_i \le M$)
$M = M - w_i$;
$P = P + P_i$;
else break;
if (M > 0)
$P = P + P_i \left(\frac{M}{w_i}\right)$;
GATE Smashers
SUBSCRIBE


Knapsack Problem
objects | ob1 | ob2 | ob3
Profit | 25 | 24 | 15
Weight | 18 | 15 | 10
P/w | 1.3 | 1.6 | 1.5
Knapsack Capacity(M) = 20
20
eedy about Profit
25 + 2/15 × 24
= 25 + 48/15 3.2 = 28.2
ob1
2
18
Greedy about weight
ob3
10
15 + 10/15 × 24
15 + 240/15 16 = 31
P/w
→ for i=1 to n
Calculate Profit/weight
→ Sort objects in decreasing order of P/w Ratio
→ for i=1 to n
if M>0 and wi≤M)
M = M − wi ;
P = P + Pi ;
else break;
if (M>0)
P = P + Pi (M/wi) ;
GATE Smashers
SUBSCRIBE
23/52

Knapsack Problem

objects	ob_1	ob_2	ob_3
Profit	25	24	15
Weight	18	15	10
P/w	1.3	1.6	1.5

Knapsack Capacity(M) = 20

1) Greedy about Profit

$$25 + \frac{2}{15} \times 24$$

$$= 25 + \frac{48}{15}\,3.2 = 28.2$$

2) Greedy about weight

$$15 + \frac{10}{15} \times 24$$

$$15 + \frac{240}{15}\,16 = 31$$

3) P/w Both

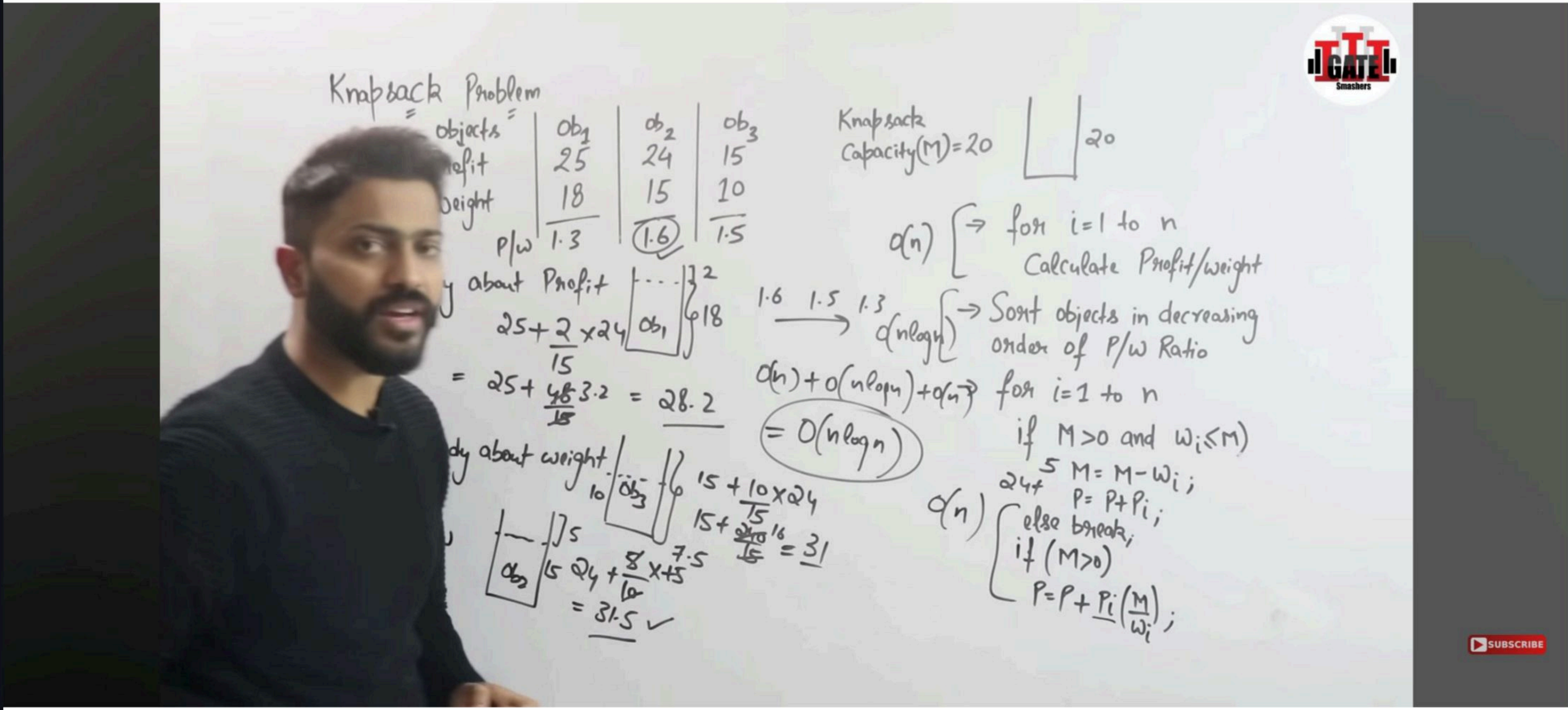
$$24 + \frac{5}{10} \times 15 \quad 7.5$$

$$= 31.5$$

→ for i=1 to n
  Calculate Profit/weight

→ Sort objects in decreasing order of P/w Ratio

→ for i=1 to n
  if (M>0 and $w_i \le M$)
    $M = M - w_i$;
    $P = P + P_i$;
  else break;
  if (M>0)
    $P = P + P_i\left(\frac{M}{w_i}\right)$;


Knapsack Problem
objects | ob1 | ob2 | ob3
Profit | 25 | 24 | 15
Weight | 18 | 15 | 10
P/w | 1.3 | 1.6 | 1.5
Knapsack
Capacity(M)=20
20
about Profit
25 + 2/15 × 24
= 25 + 3.2 = 28.2
about weight
15 + 10/15 × 24
= 31
24 + 5/10 × 15
= 31.5
1.6 1.5 1.3
O(n) → for i=1 to n
Calculate Profit/weight
O(nlogn) → Sort objects in decreasing order of P/w Ratio
O(n)+O(nlogn)+O(n) → for i=1 to n
= O(nlogn)
if M>0 and wi≤M)
M= M−wi;
P= P+Pi;
else break;
if (M>0)
P=P+Pi(M/wi);
GATE Smashers
SUBSCRIBE

Coin Problem

Coin Problem (Using Greedy Approach):

The **Coin Problem** involves finding the **minimum number of coins** needed to make a certain amount of money, given a set of coin denominations.

In the **greedy approach**, we always **choose the largest possible denomination** first, then the next smaller one, and so on — until the amount becomes zero.

Example:

If the coin denominations are `{1, 2, 5, 10}` and we want to make `18`, the greedy way would be:

- Take one `10` → remaining = `8`
- Take one `5` → remaining = `3`
- Take one `2` → remaining = `1`
- Take one `1` → remaining = `0`

Total coins used = 4.

When Greedy Works:

Greedy works **only when the coin denominations are canonical** (like Indian currency: 1, 2, 5, 10, 20, 50, etc.). It may **fail** for some custom denominations.

Key Point:

- Greedy = **take the biggest coin ≤ remaining amount**
- Repeat until total amount becomes 0
- Not always optimal for all coin systems