

Data Structures and Algorithms

(CS210A)

Semester I – 2014-15

Lecture 30

Binary Trees

Magical applications

Two interesting problems on sequences

Problem 1

Multi-increment

Problem 1

Given an initial sequence $S = \langle x_0, \dots, x_{n-1} \rangle$ of n numbers, maintain a compact data structure to perform the following operations efficiently :

- **Report(i):**
Report the current value of x_i .
 - **Multi-Increment(i, j, Δ):**
Add Δ to x_k
-

Example:

Let the initial sequence be $S = \langle 14, 12, 23, 12, 111, 51, 321, -40 \rangle$

After **Multi-Increment(2,6,10)**, S becomes

$\langle 14, 12, 33, 22, 121, 61, 331, -40 \rangle$

Trivial solution discussed in the last class :

- $O(n)$ time per **Multi-Increment(i, j, Δ)**
- $O(1)$ time per **Report(i)**

Towards efficient solution of Problem 1

Explore ways to maintain sequence S **implicitly** such that

- **Multi-Increment**(i, j, Δ) is efficient.
- **Report**(i) is efficient too.

Main hurdle: To perform **Multi-Increment**(i, j, Δ) efficiently

Assumption: without loss of generality assume n is power of 2.

A SYSTEMATIC JOURNEY TO THE SOLUTION

A motivating problem

$$S = \{1, 2, 3, \dots, 2^n\}$$

Question:

Can we have a small set $X \subset S$ of numbers s.t.

Every number from S can be expressed as a sum of a few numbers from X ?

Answer: $X = \{1, 2, 4, 8, \dots, 2^n\}$

$$|X| = n$$

1 0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0

1 0 0 0 0

1 0 0 0

1

1 0 0 1 0 1 1 0 0 1

If it is too trivial, try to answer the problem of next slide. 😊

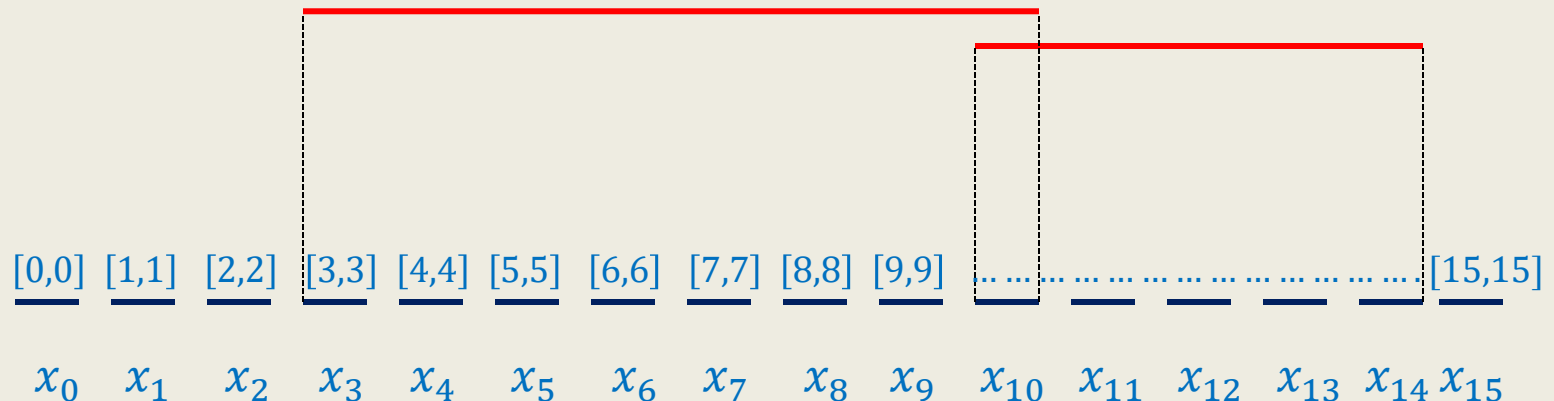
Extension to intervals

$$S = \{[i, j], 0 \leq i \leq j < n\}$$

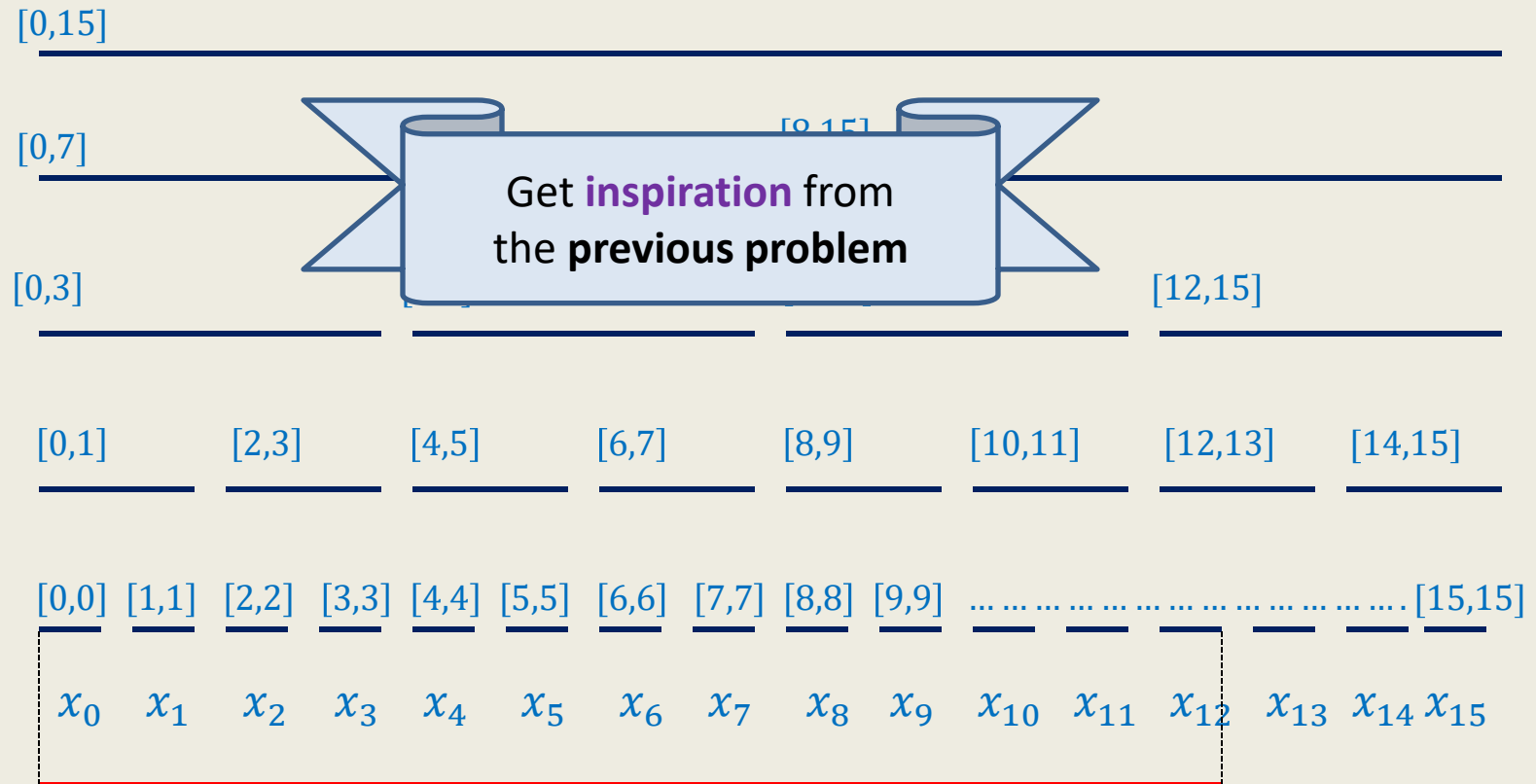
Question:

Can we have a small set $X \subset S$ of **intervals** s.t.

every interval in S can be expressed as a union of a few intervals from X ?

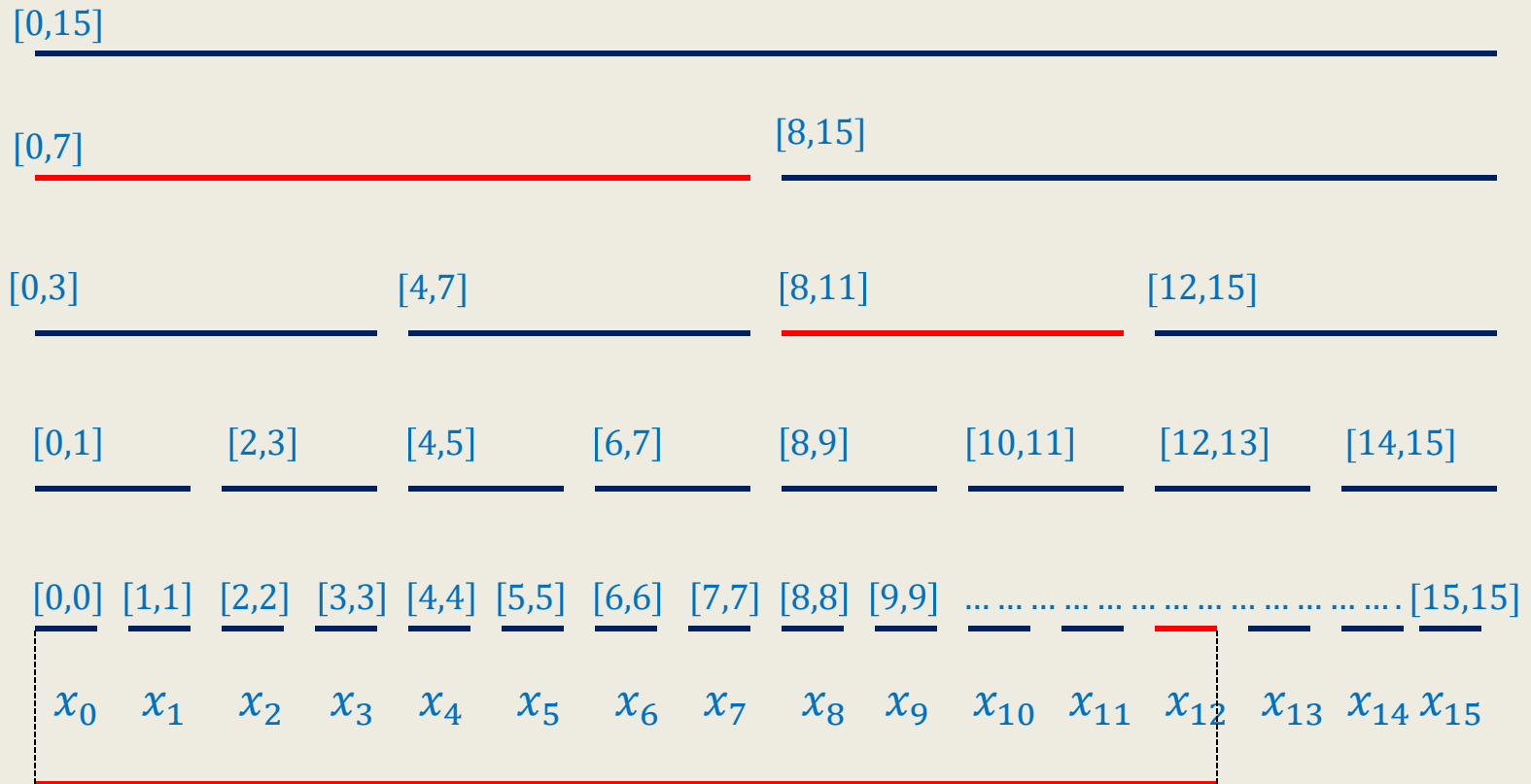


Extension to intervals



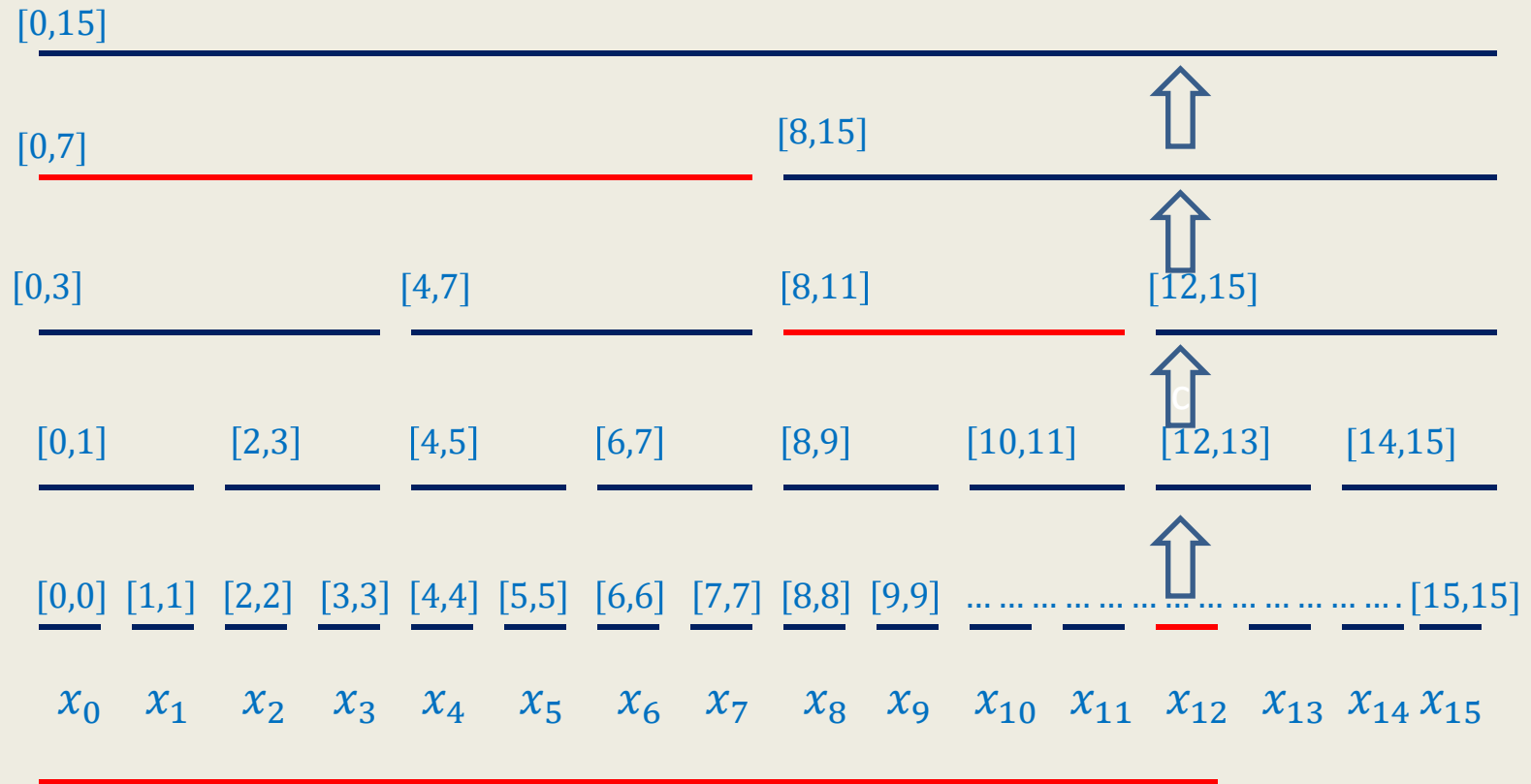
How to express $[0, 12]$?

Extension to intervals



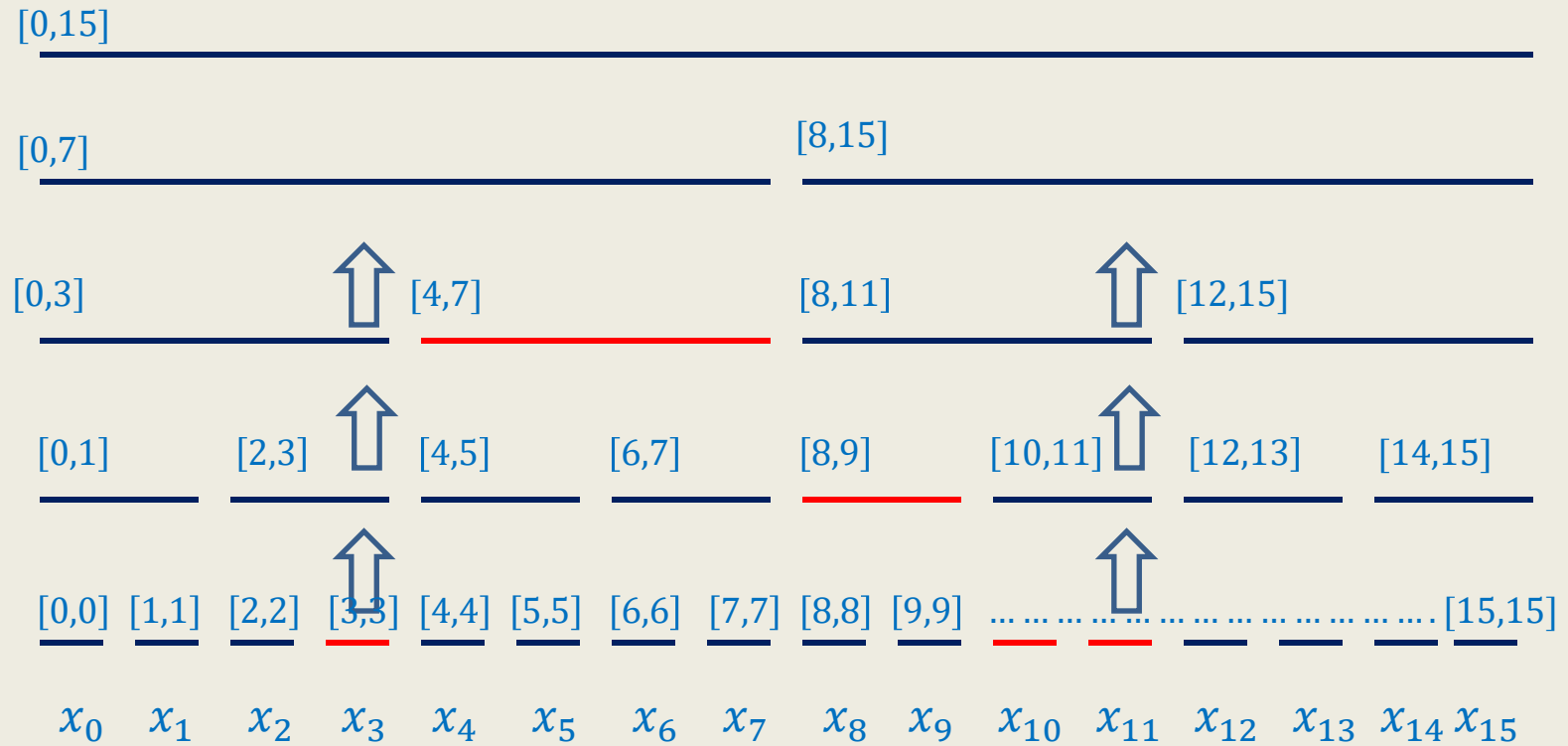
How to express $[0, 12]$?

Extension to intervals



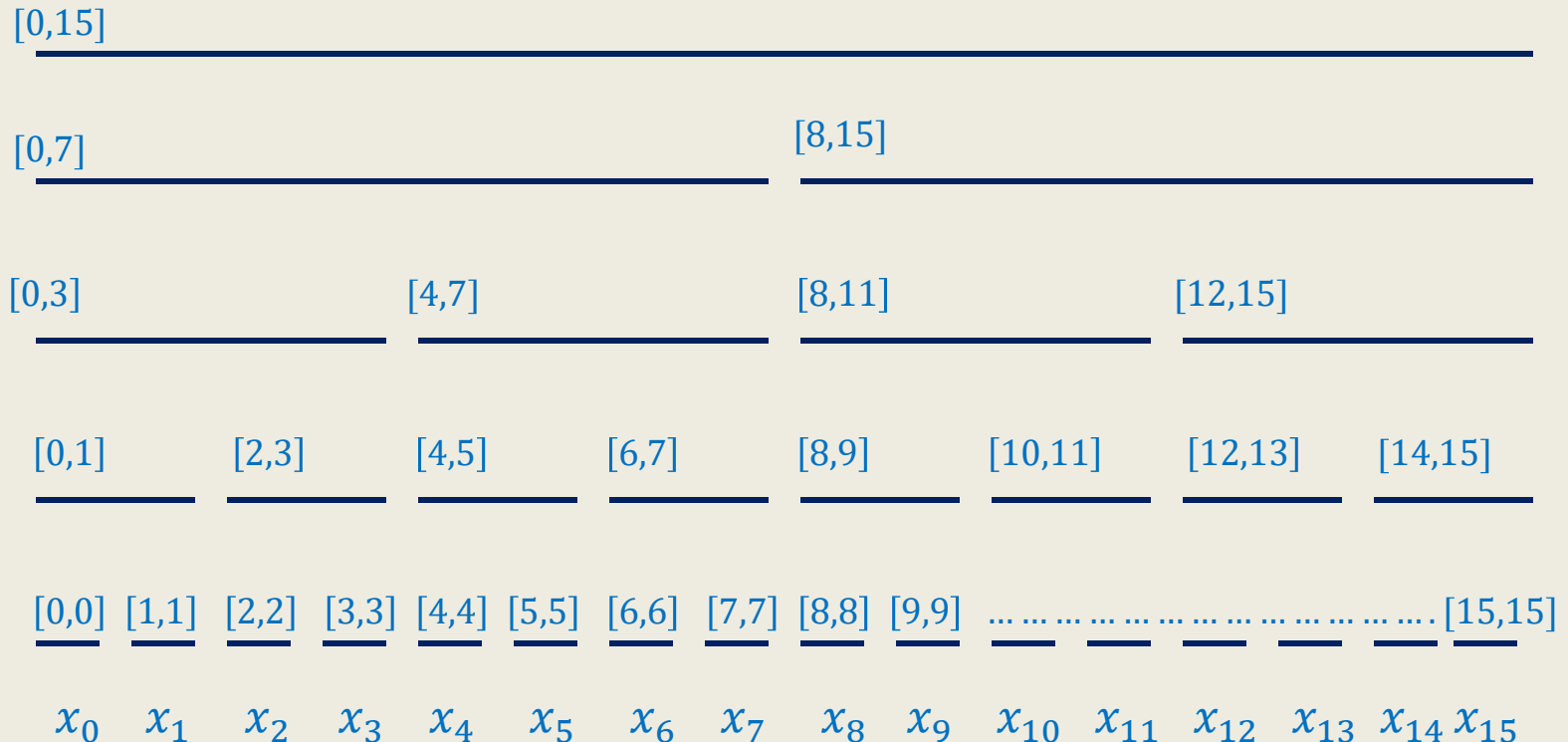
How to express $[0, 12]$?

Extension to intervals



How to express $[3, 11]$?

How to use this **Observation** to perform Multi-Increment(i, j, Δ) efficiently?

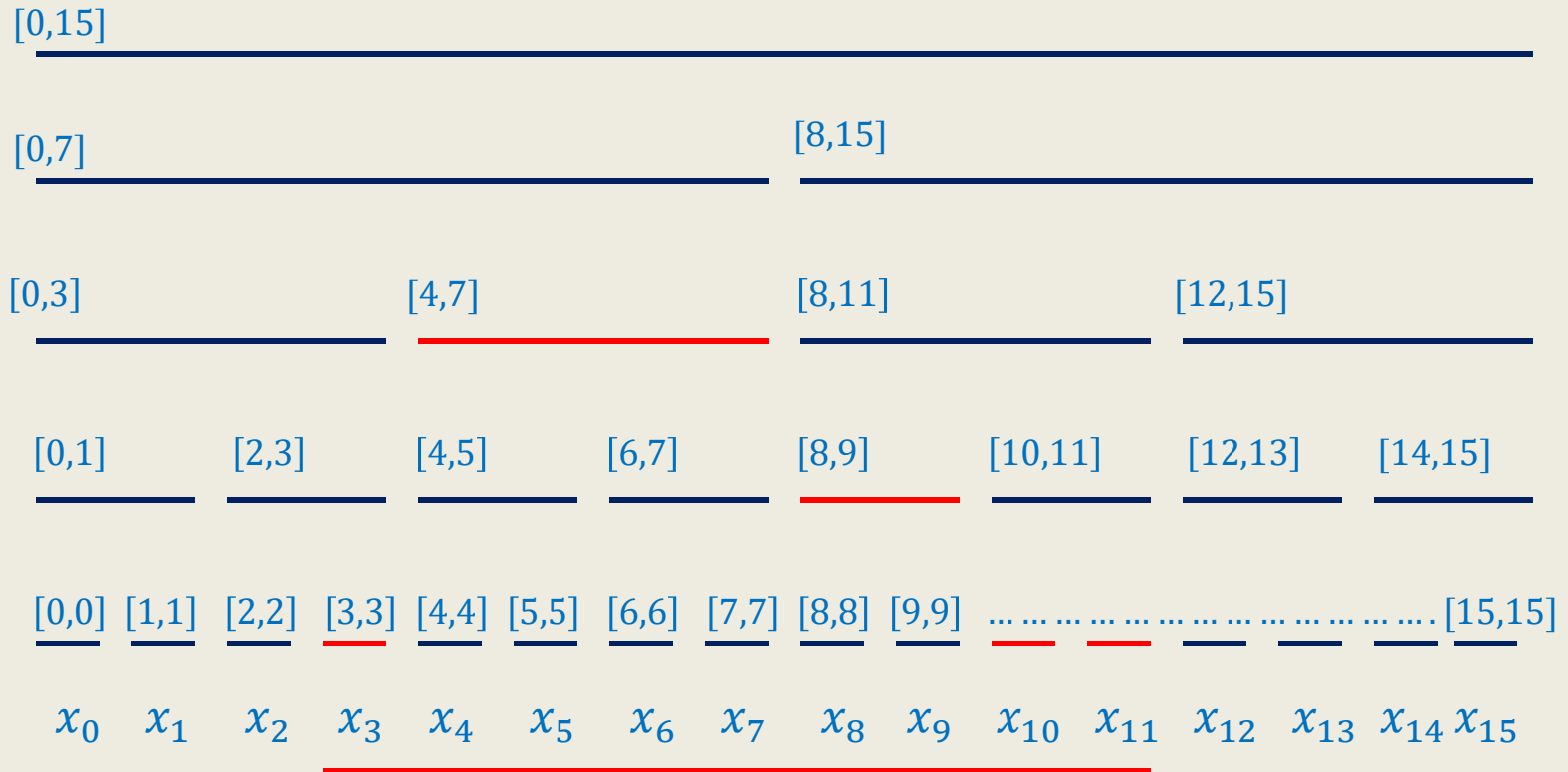


Observation:

There are $2n$ intervals such that

any interval $[i, j]$ can be expressed as **union** of $O(\log n)$ basic intervals 😊

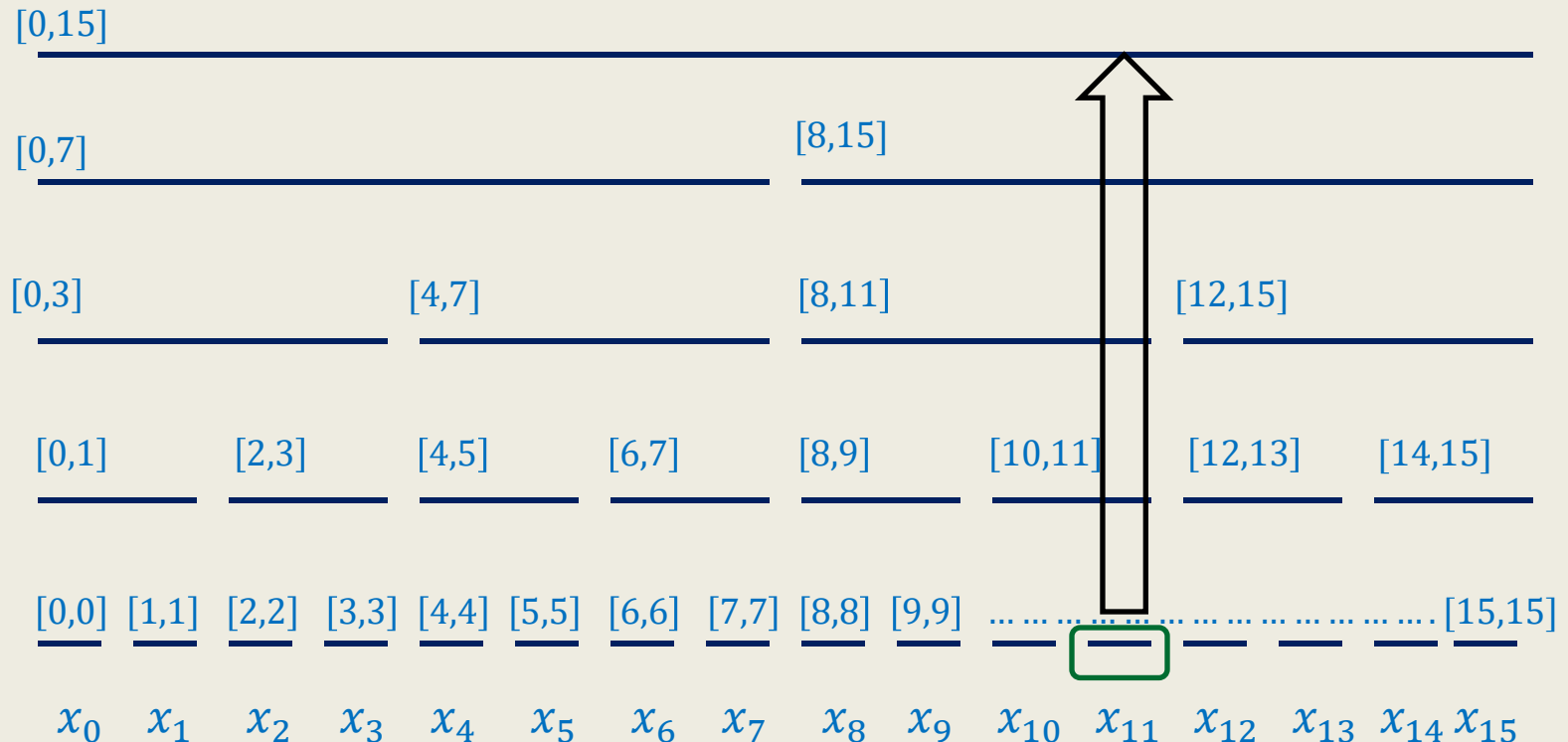
How to use this **Observation** to perform **Multi-Increment**(i, j, Δ) efficiently?



Maintain $2n$ intervals with a field **increment**

Multi-Increment(i, j, Δ) \rightarrow add Δ to **increment** field of its $O(\log n)$ intervals.

How to use this **Observation** to perform
Multi-Increment(i, j, Δ) efficiently?

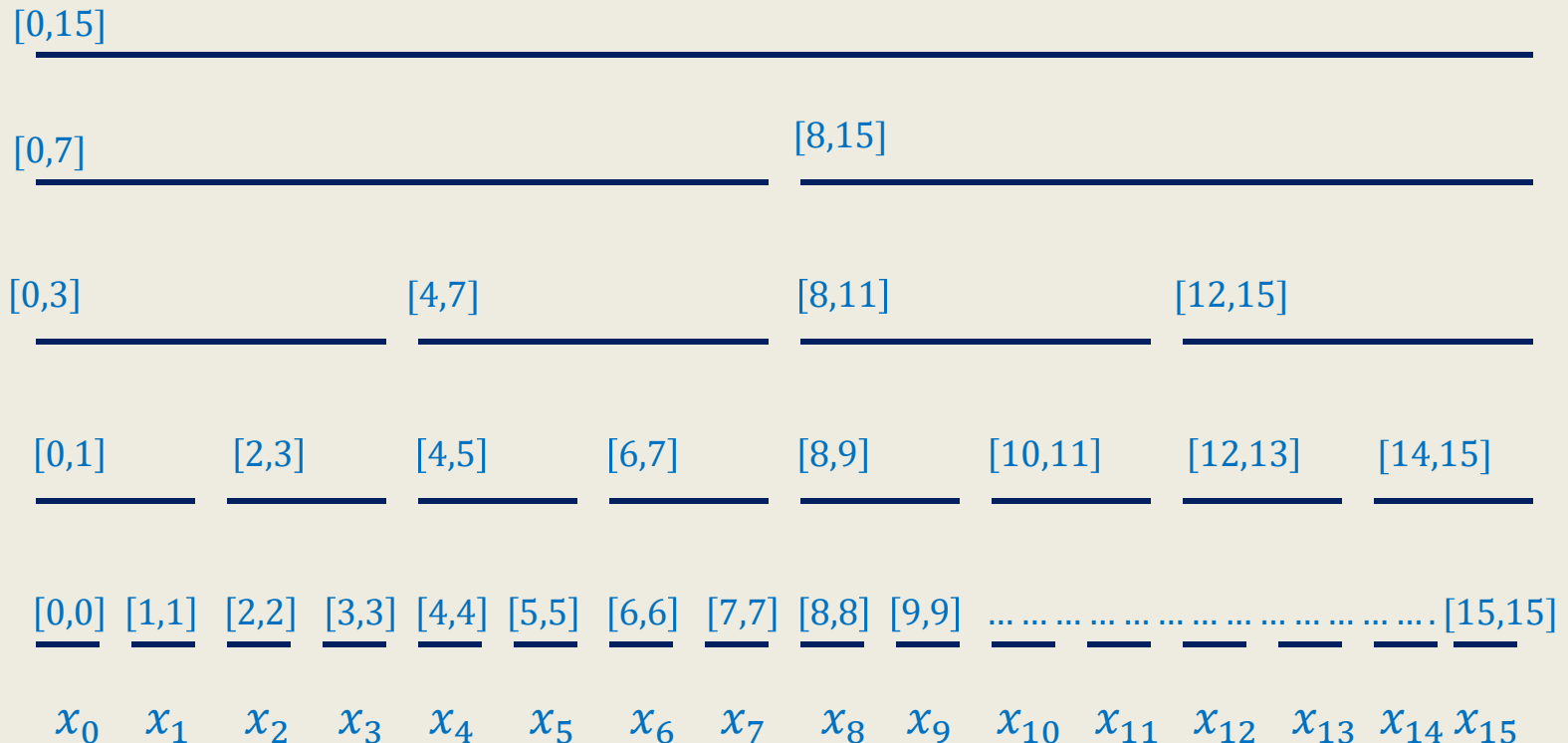


Maintain $2n$ intervals with a field **increment**

Multi-Increment(i, j, Δ) \rightarrow add Δ to **increment** field of its $O(\log n)$ intervals.

How to perform **Report**(i) ?

How to use this **Observation** to perform
Multi-Increment(i, j, Δ) efficiently?



Maintain $2n$ intervals with a field **increment**

Multi-Increment(i, j, Δ) \rightarrow add Δ to **increment** field of its $O(\log n)$ intervals.

How to perform **Report**(i) ?

What data structure to use ?

You might like to have another look on the last slide to answer this question.

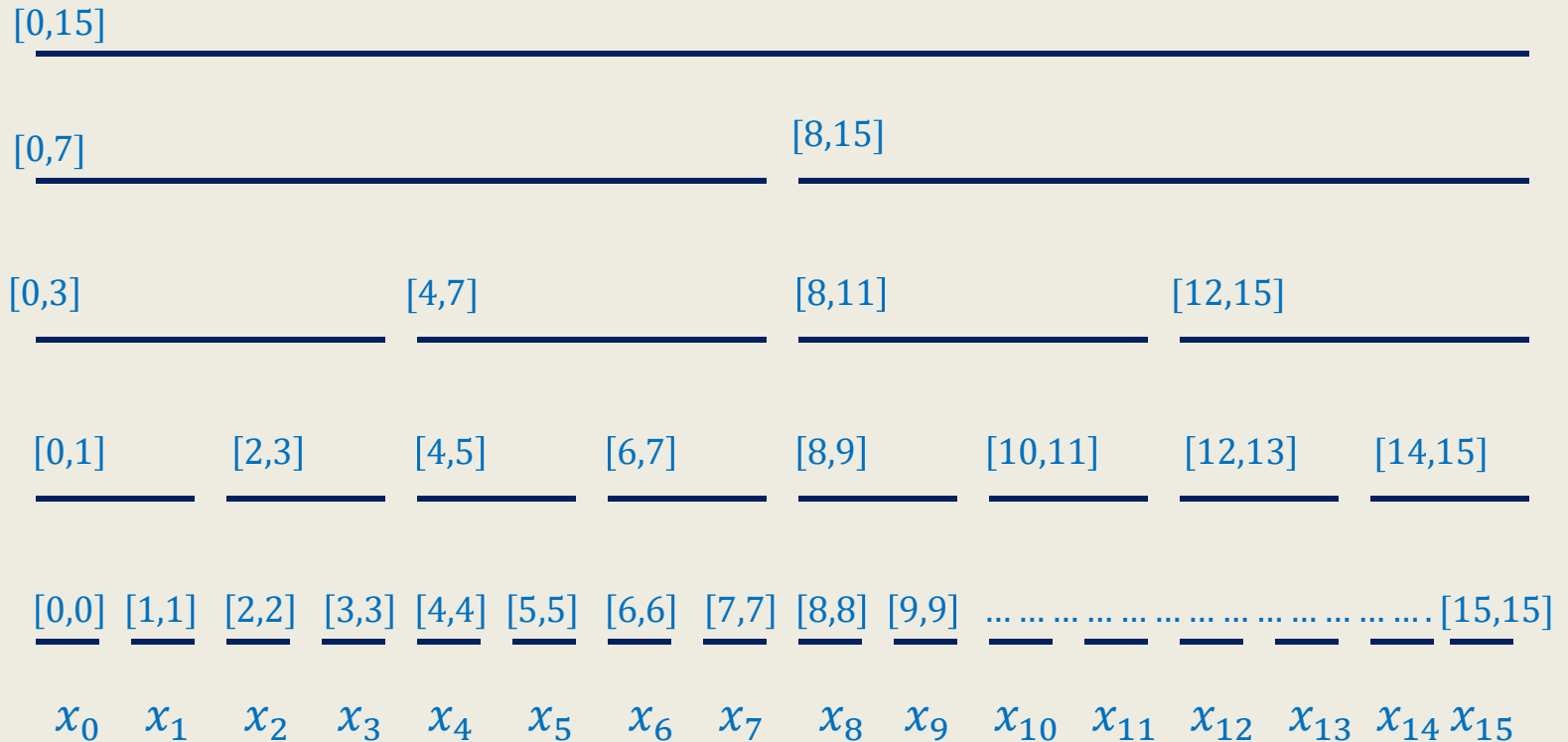
I have **reproduced** it for you again in the next slide

Have another look,

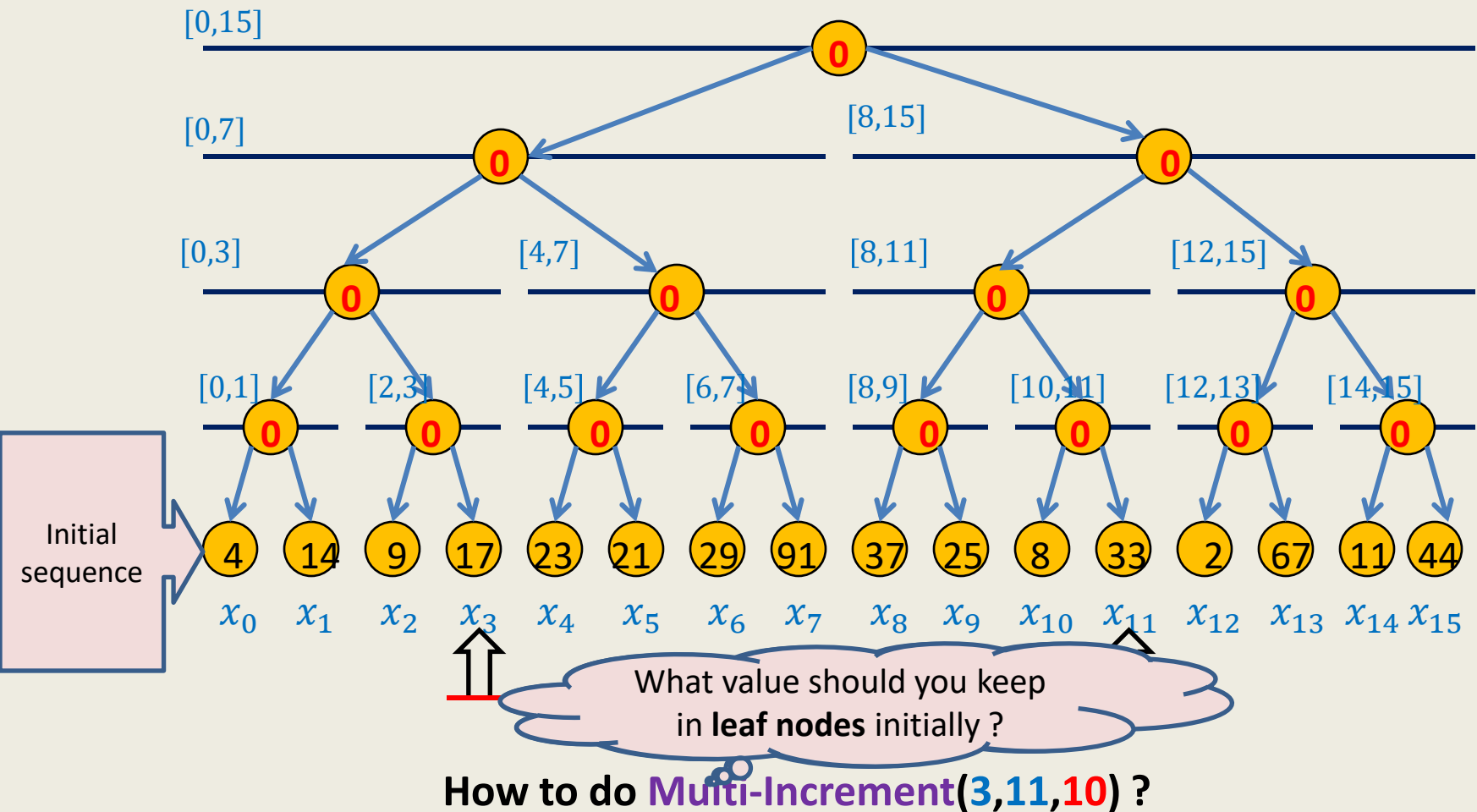
think for a while ...

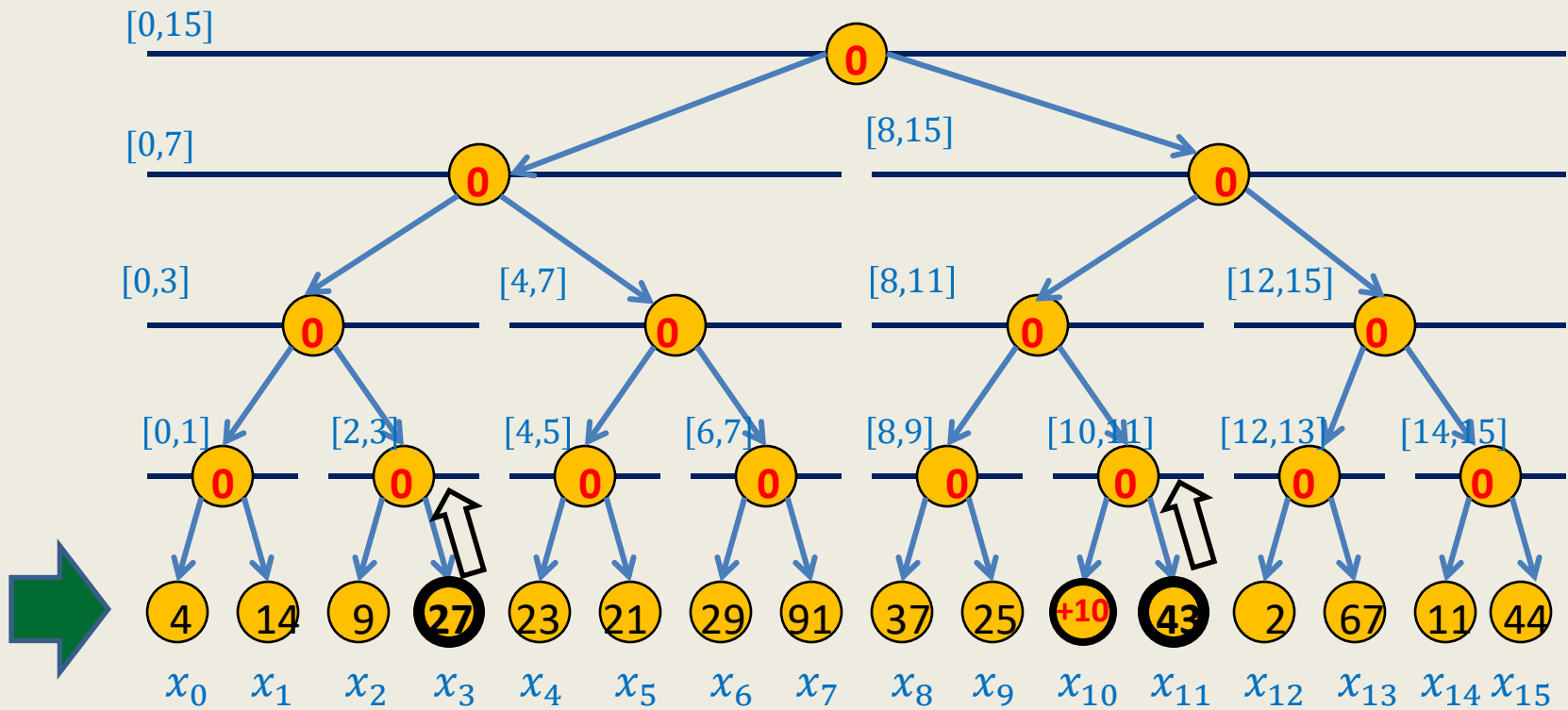
and then only proceed.

Which data structure emerges ?

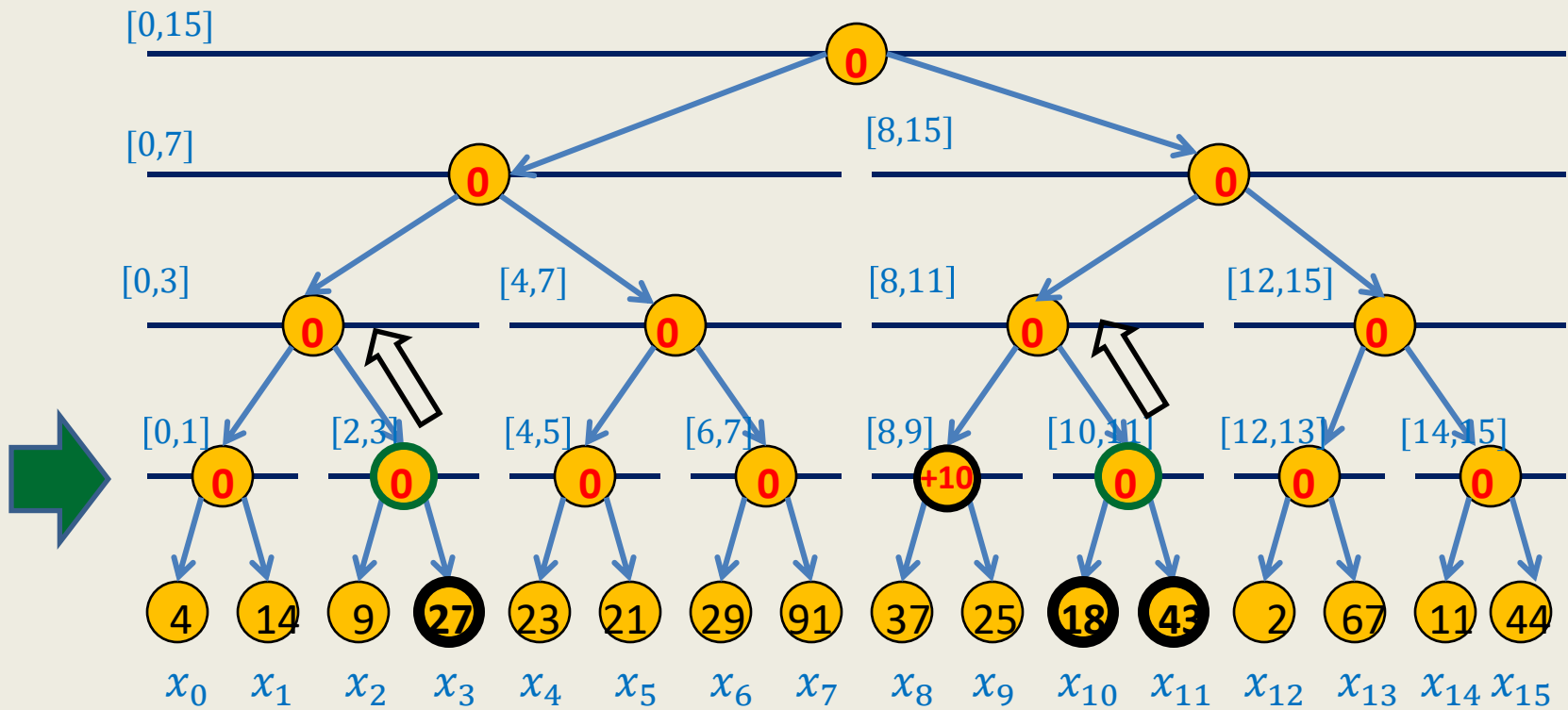


Isn't it a **Binary tree** that you thought ?

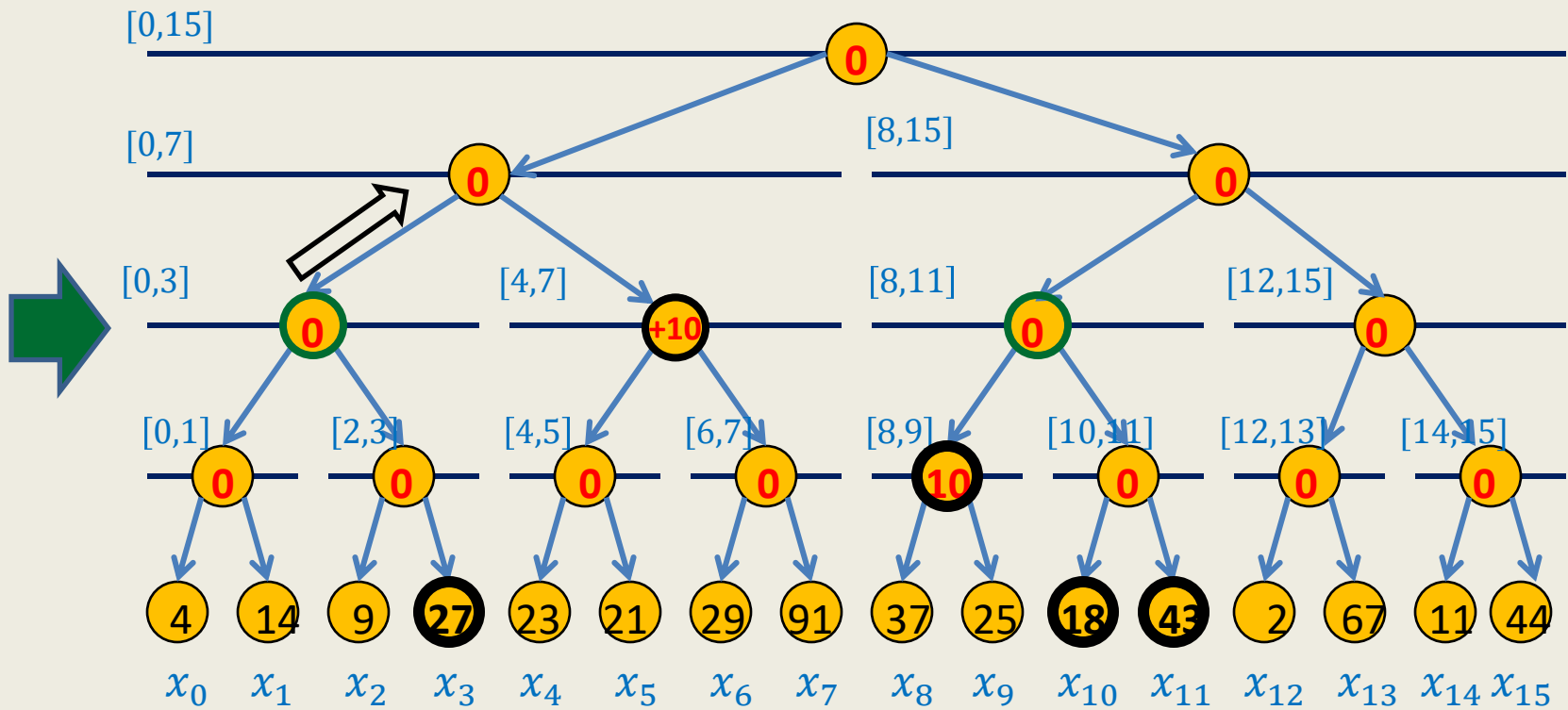




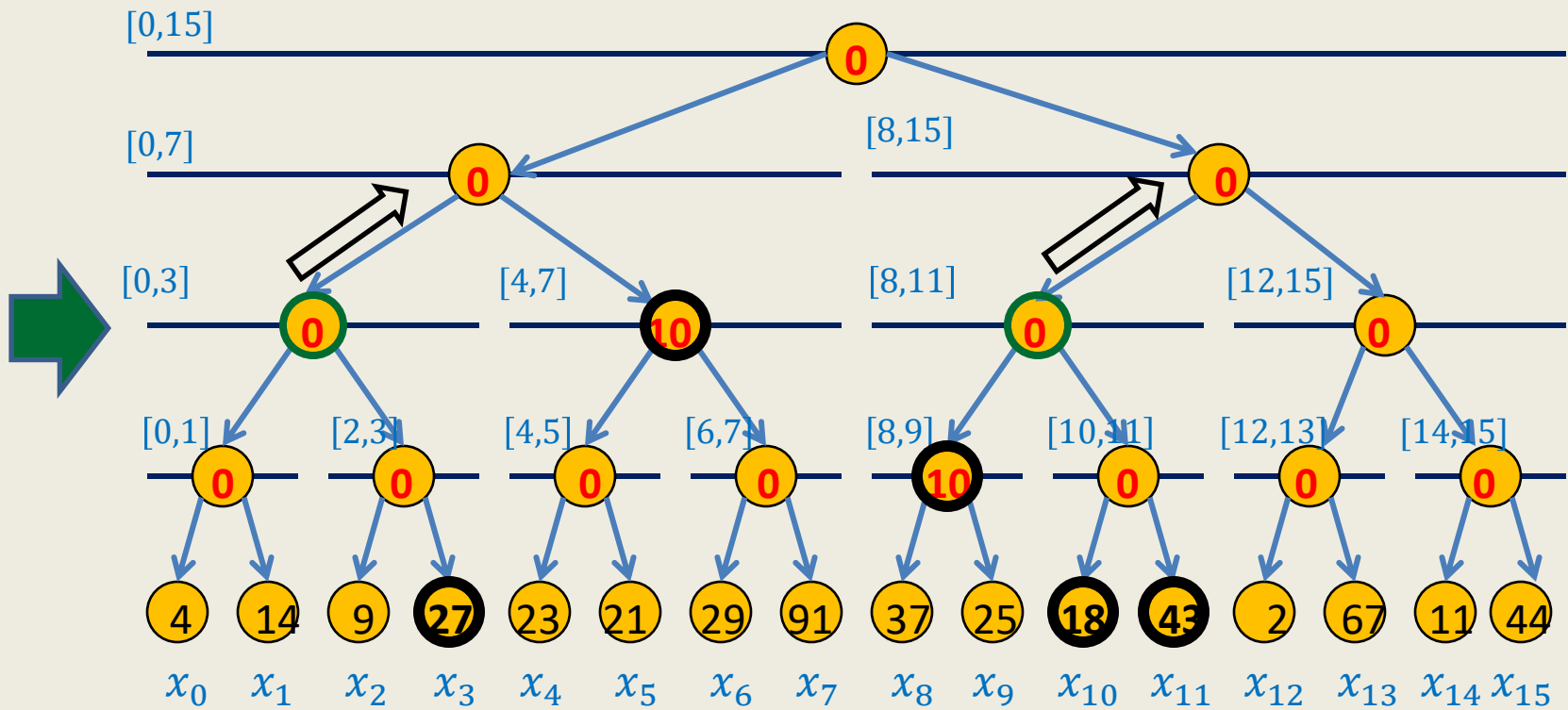
How to do Multi-Increment(3,11,10) ?



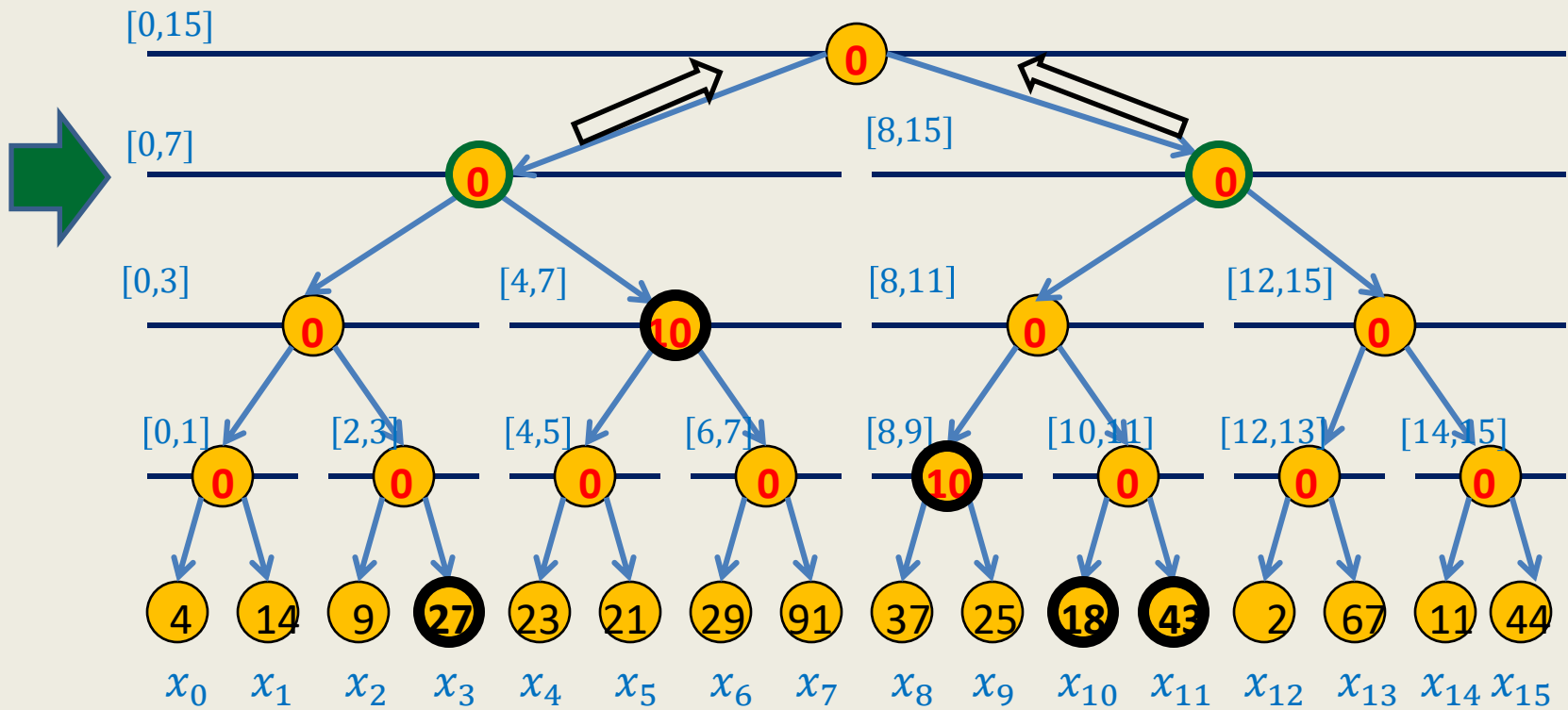
How to do Multi-Increment(3,11,10) ?



How to do Multi-Increment(3,11,10) ?

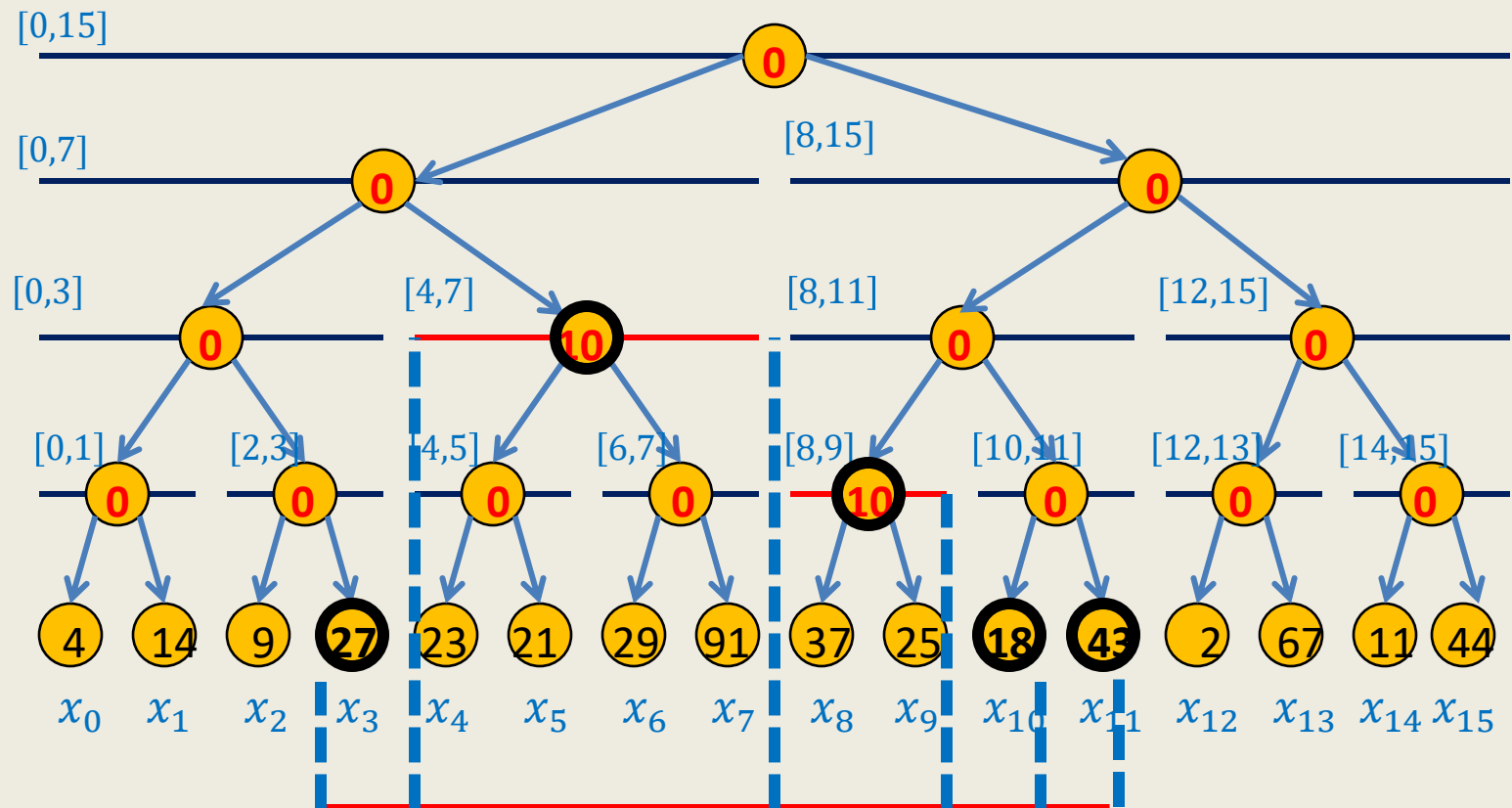


How to do Multi-Increment(3,11,10) ?



How to do Multi-Increment(3,11,10) ?

Are we done ?



How to do Multi-Increment(3,11,10) ?

Yes

Multi-Increment(i, j, Δ) efficiently

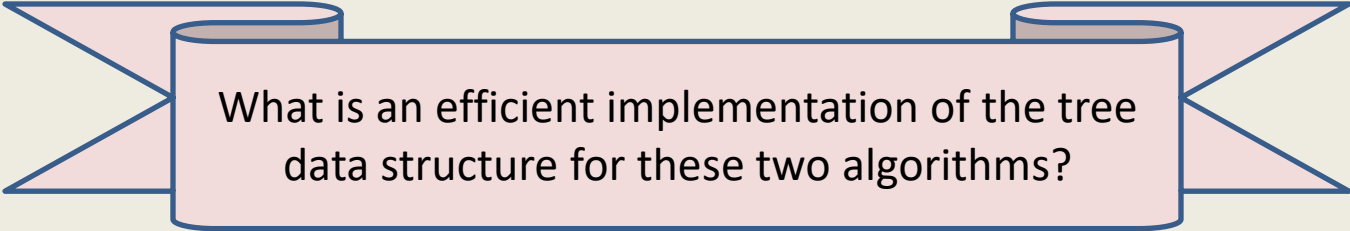
Sketch:

1. Let u and v be the leaf nodes corresponding to x_i and x_j .
2. Increment the value stored at u and v .
3. Keep repeating the following **step** as long as $\text{parent}(u) \neq \text{parent}(v)$
Move up by one step simultaneously from u and v
 - If u is **left child** of its parent, increment value stored in sibling of u .
 - If v is **right child** of its parent, increment value stored in sibling of v .

Executing **Report(*i*)** efficiently

Sketch:

1. Let **u** be the leaf nodes corresponding to x_i .
2. **val** \leftarrow 0;
3. Keep moving up from **u** and keep adding the value of all the nodes on the path to the root to **val**.
4. Return **val**.

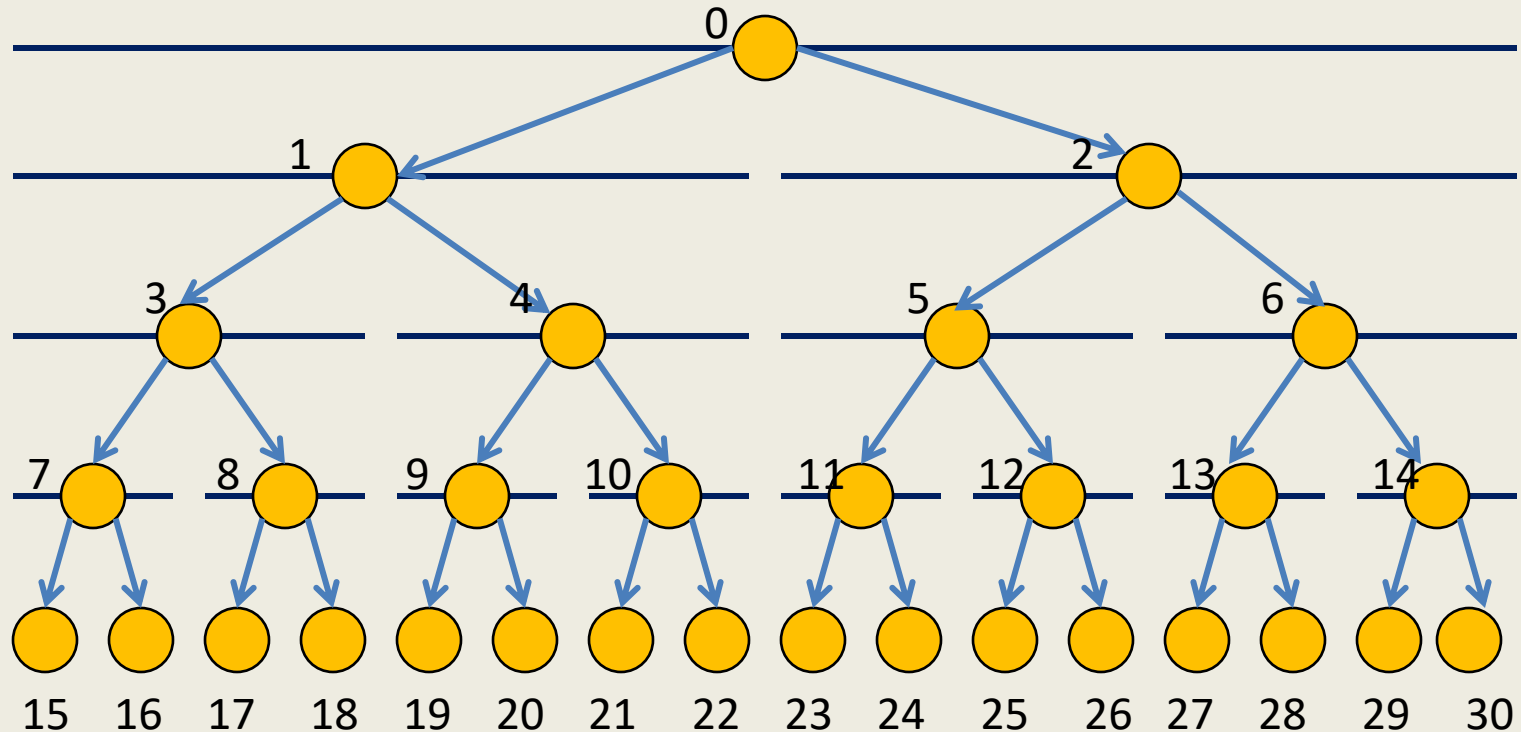


What is an efficient implementation of the tree data structure for these two algorithms?



Realize that it was a **complete binary tree**.

Exploiting complete binary tree structure



Data structure: An array **A** of size $2n-1$.

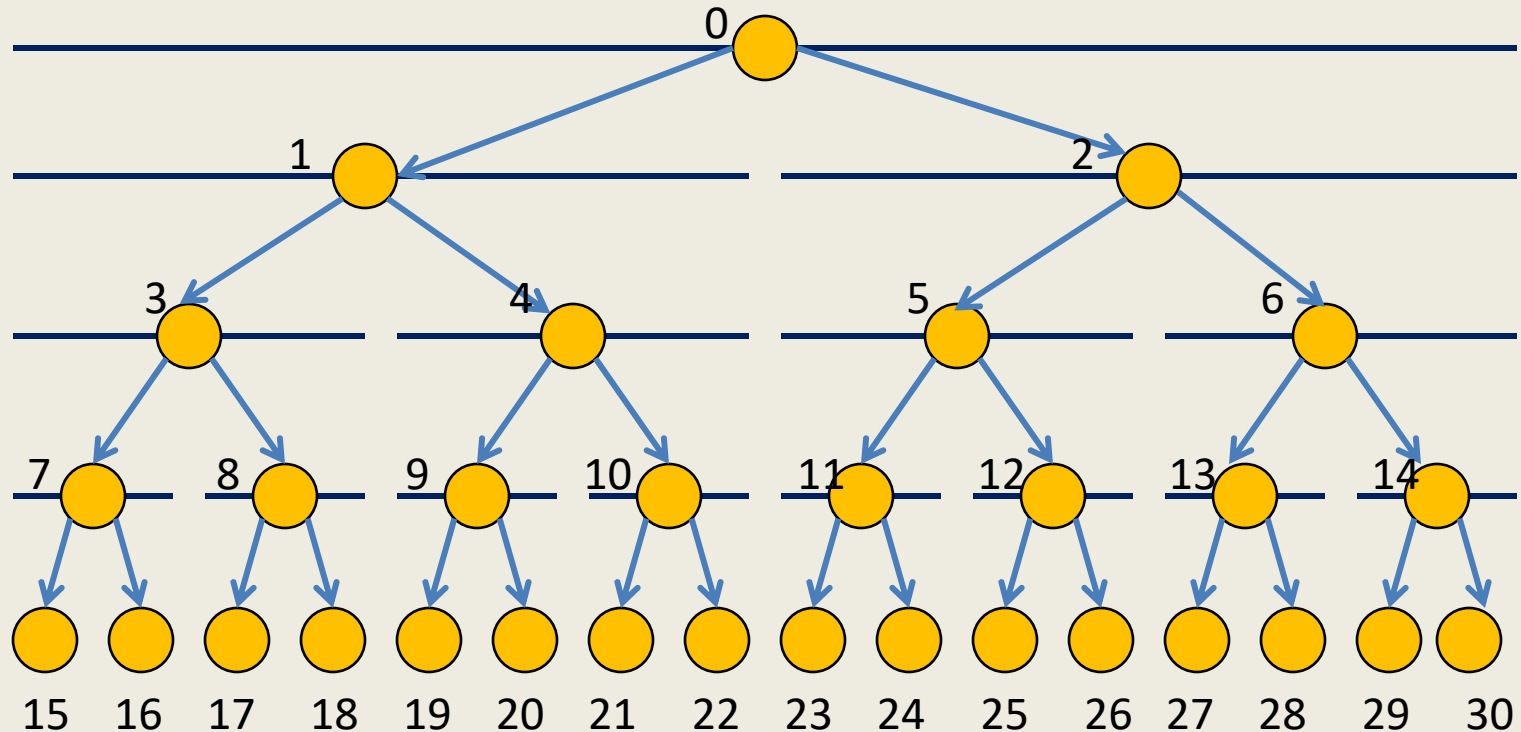
Copy the sequence $\mathbf{S} = \langle x_0, \dots, x_{n-1} \rangle$ into $\mathbf{A}[n-1] \dots \mathbf{A}[2n-2]$

Leaf node corresponding to $x_i = \mathbf{A}[(n-1) + i]$

How to check if a node is **left child** or **right child** of its parent ?

(if index of the node is odd, then the node is left child, else the node is right child)

Exploiting complete binary tree structure



Data structure: An array **A** of size $2n-1$.

Copy the sequence $\mathbf{S} = \langle x_0, \dots, x_{n-1} \rangle$ into $\mathbf{A}[n-1] \dots \mathbf{A}[2n-2]$

Leaf node corresponding to $x_i = \mathbf{A}[(n-1) + i]$

How to check if a node is **left child** or **right child** of its parent ?

MultIncrement(i, j, Δ)

MultIncrement(i, j, Δ)

$i \leftarrow (n - 1) + i$;

$j \leftarrow (n - 1) + j$;

$A(i) \leftarrow A(i) + \Delta$;

If ($j > i$)

{ $A(j) \leftarrow A(j) + \Delta$;

While($\lfloor (i - 1)/2 \rfloor \neq \lfloor (j - 1)/2 \rfloor$)

{

 If($i \% 2 = 1$) $A(i + 1) \leftarrow A(i + 1) + \Delta$;

 If($j \% 2 = 0$) $A(j - 1) \leftarrow A(j - 1) + \Delta$;

$i \leftarrow \lfloor (i - 1)/2 \rfloor$;

$j \leftarrow \lfloor (j - 1)/2 \rfloor$;

}

}

Report(*i*)

Report(*i*)

$i \leftarrow (n - 1) + i ;$

$val \leftarrow 0;$

While($i > 0$)

{

$val \leftarrow val + A[i];$

$i \leftarrow \lfloor (i - 1) / 2 \rfloor ;$

}

return val ;

The **solution** of **Multi-Increment Problem**

Theorem:

There exists a data structure of size $\mathcal{O}(n)$
for maintaining a sequence $\mathbf{S} = \langle x_0, \dots, x_{n-1} \rangle$ such that
each **Multi-Increment()** and **Report()** operation takes $\mathcal{O}(\log n)$ time.

Problem 2

Dynamic Range-minima

Problem 2

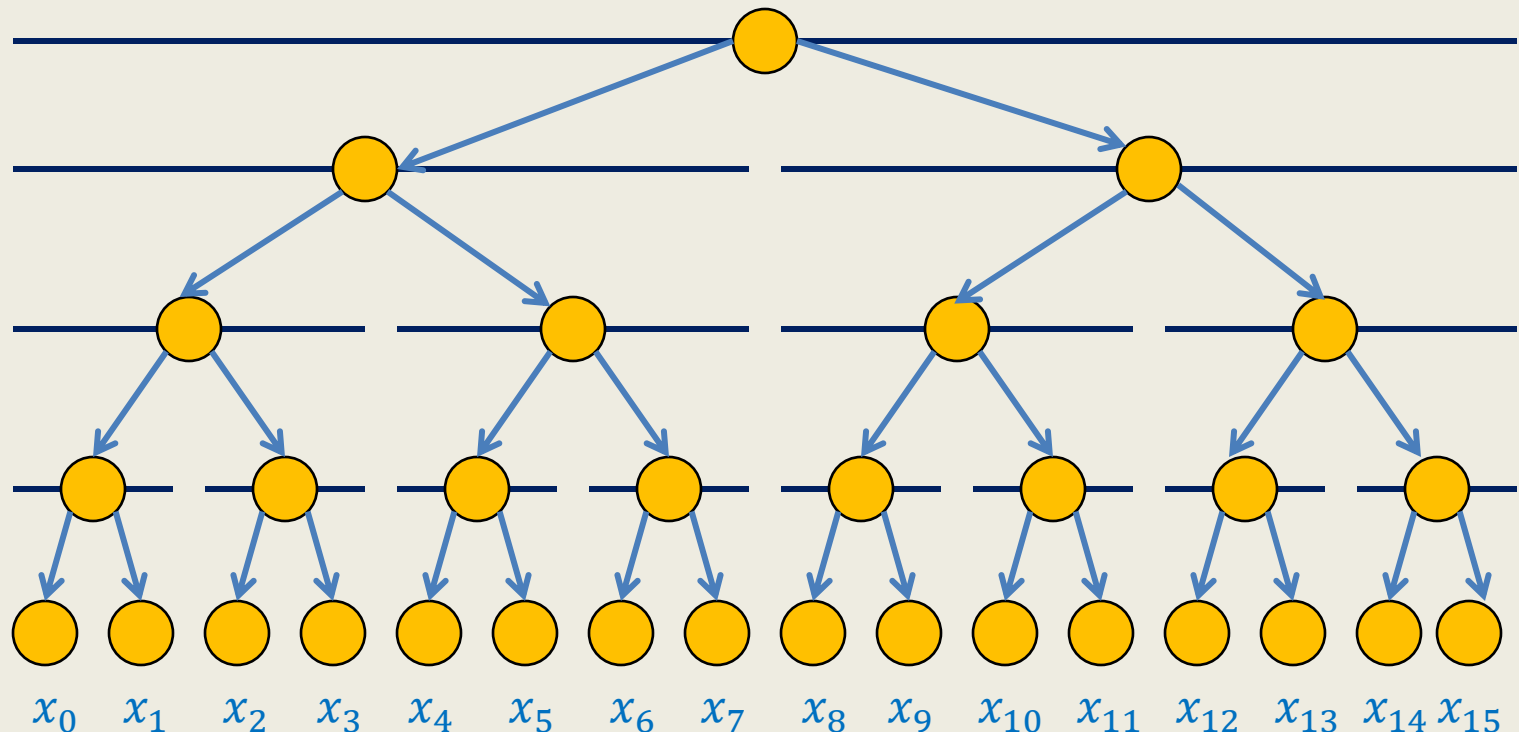
Given an initial sequence $S = \langle x_0, \dots, x_{n-1} \rangle$ of numbers, maintain a compact data structure to perform the following operations efficiently for any $0 \leq i < j < n$.

- **ReportMin**(i, j):
Report the minimum element from $\{x_k \mid \text{for each } i \leq k \leq j\}$
- **Update**(i, a):
 a becomes the new value of x_i .

AIM:

- $O(n)$ size data structure.
- **ReportMin**(i, j) in $O(\log n)$ time.
- **Update**(i, a) in $O(\log n)$ time.

Efficient dynamic range minima



What **to store** at internal nodes ?

How to perform **ReportMin**(i, j) ?

How to perform **Update**(i, a) ?

Make sincere attempts to solve the problem. We shall discuss it in next class 😊