Data Structures and Algorithms

(CS210A)

Semester I – **2014-15**

Lecture 30

- Magical applications of Binary trees

Two interesting problems on sequences

Problem 1

Multi-increment

Problem 1

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

- **Report**($\boldsymbol{i}$):

  Report the current value of x_i.

- **Multi-Increment**($\boldsymbol{i}$, $\boldsymbol{j}$, **Δ**):

  Add **Δ** to each x_k for each $\boldsymbol{i} \leq \boldsymbol{k} \leq \boldsymbol{j}$

Example:

Let the initial sequence be **S** = $\prec$ 14, 12, 23, 12, 111, 51, 321, -40 $\succ$

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

$\prec$ 14, 12, **33**, **22**, **121**, **61**, **331**, -40 $\succ$

Trivial solution discussed in the last class :

- **O**($\boldsymbol{n}$) time per **Multi-Increment**($\boldsymbol{i}$, $\boldsymbol{j}$, **Δ**):
- **O**(1) time per **Report**($\boldsymbol{i}$):

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 = $\{1,2,3, \ldots, 2^n\}$

Question: Can we have a small set **X**⊂**S** of numbers s.t.

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

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

|**X**|= n

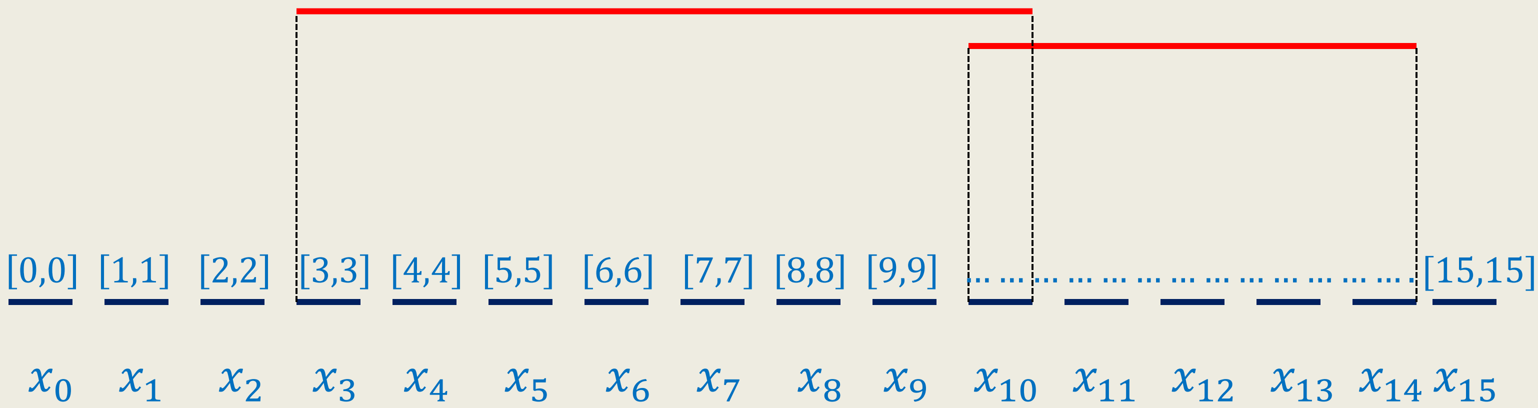
```
1 0 0 0 0 0 0  0 0 0
      1 0 0 0 0 0 0
            1 0 0 0 0
               1 0 0 0
                       1
1 0 0 1 0 1 1  0 0 1
```

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

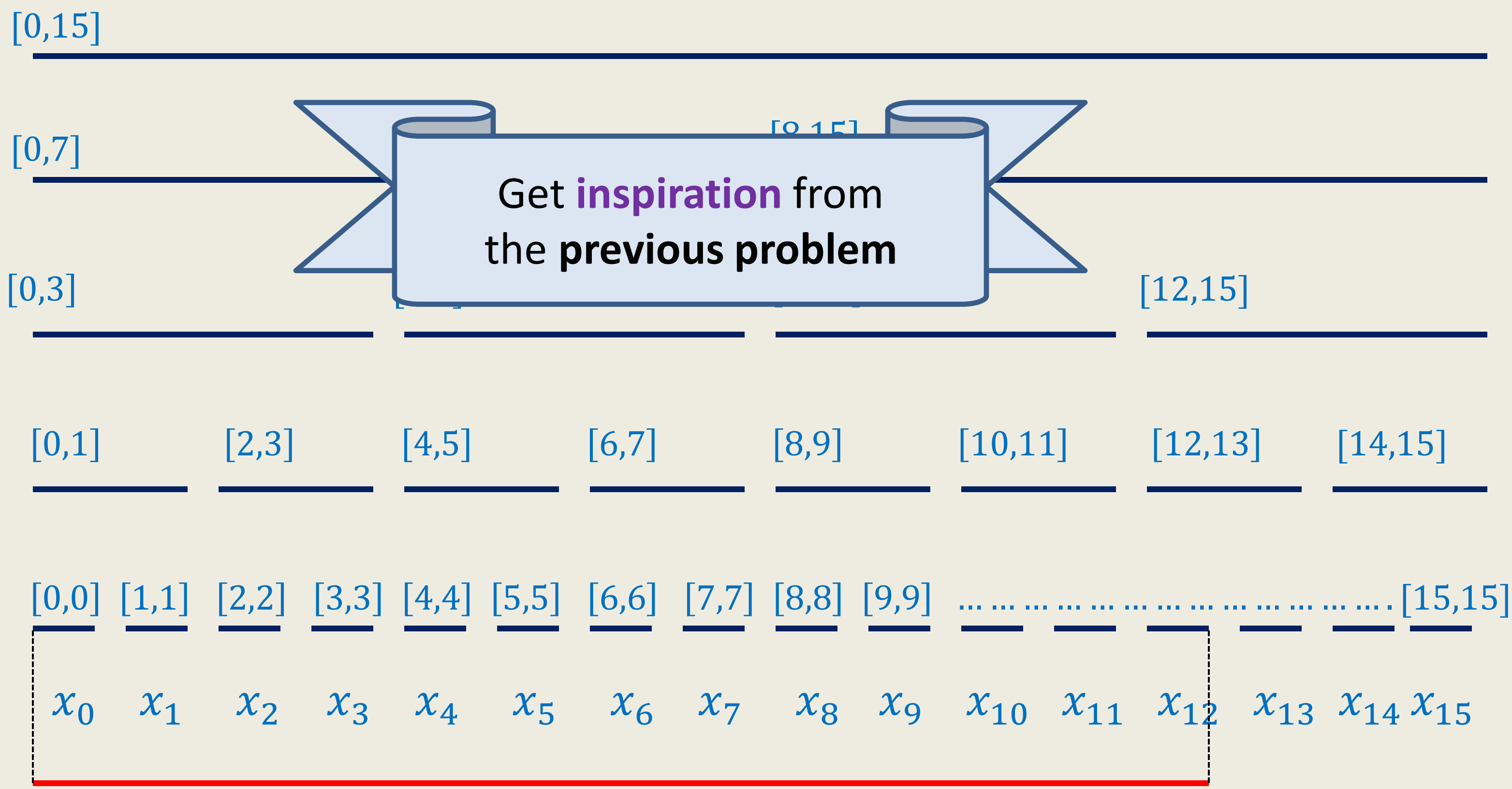
Extension to intervals

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

Question: Can we have a small set **X**⊂**S** of **intervals** s.t.
every interval in **S** can be expressed as a union of a few **intervals** from **X** ?

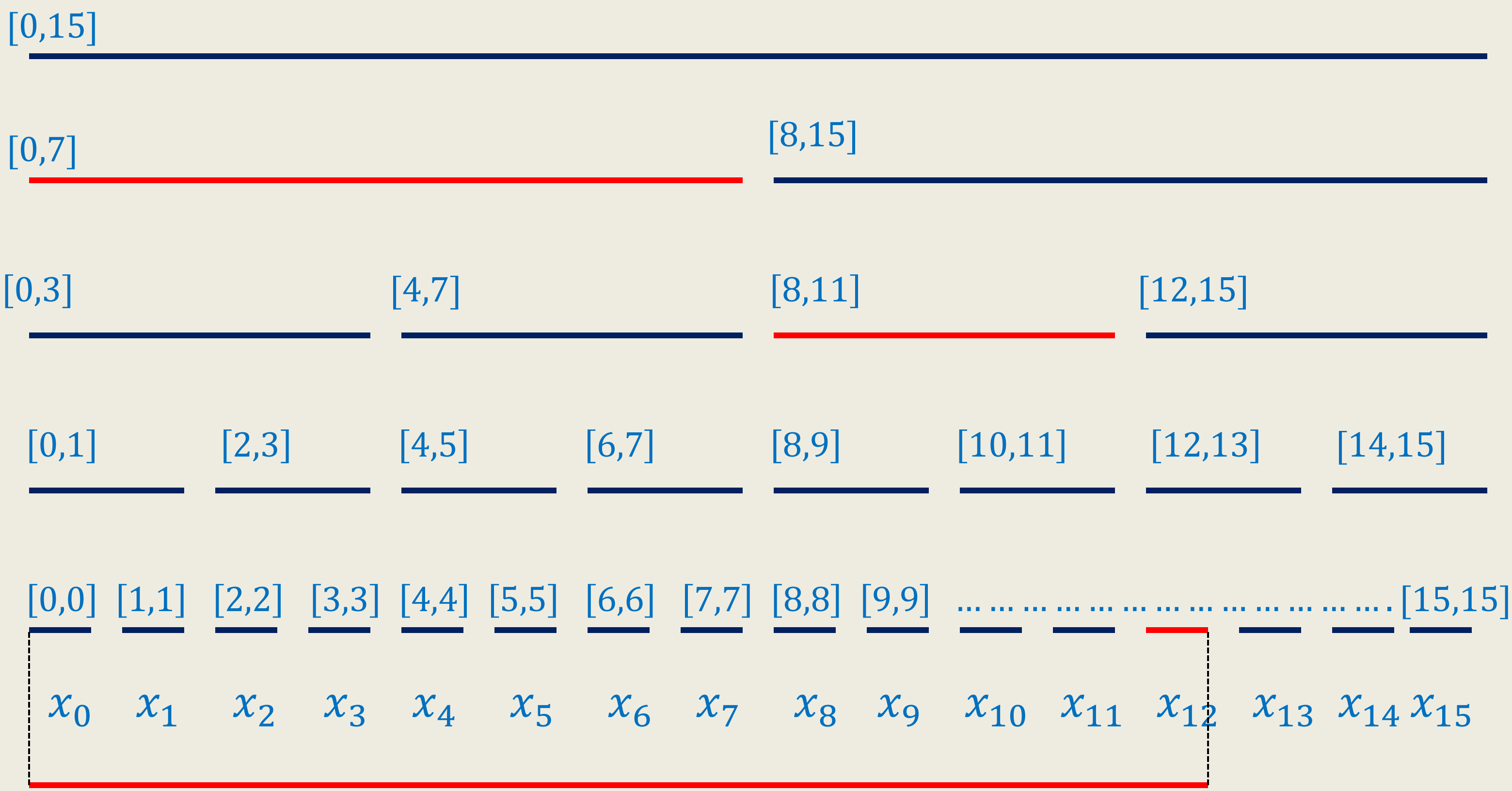


Extension to intervals



How to express [**0**, **12**] ?

Extension to intervals

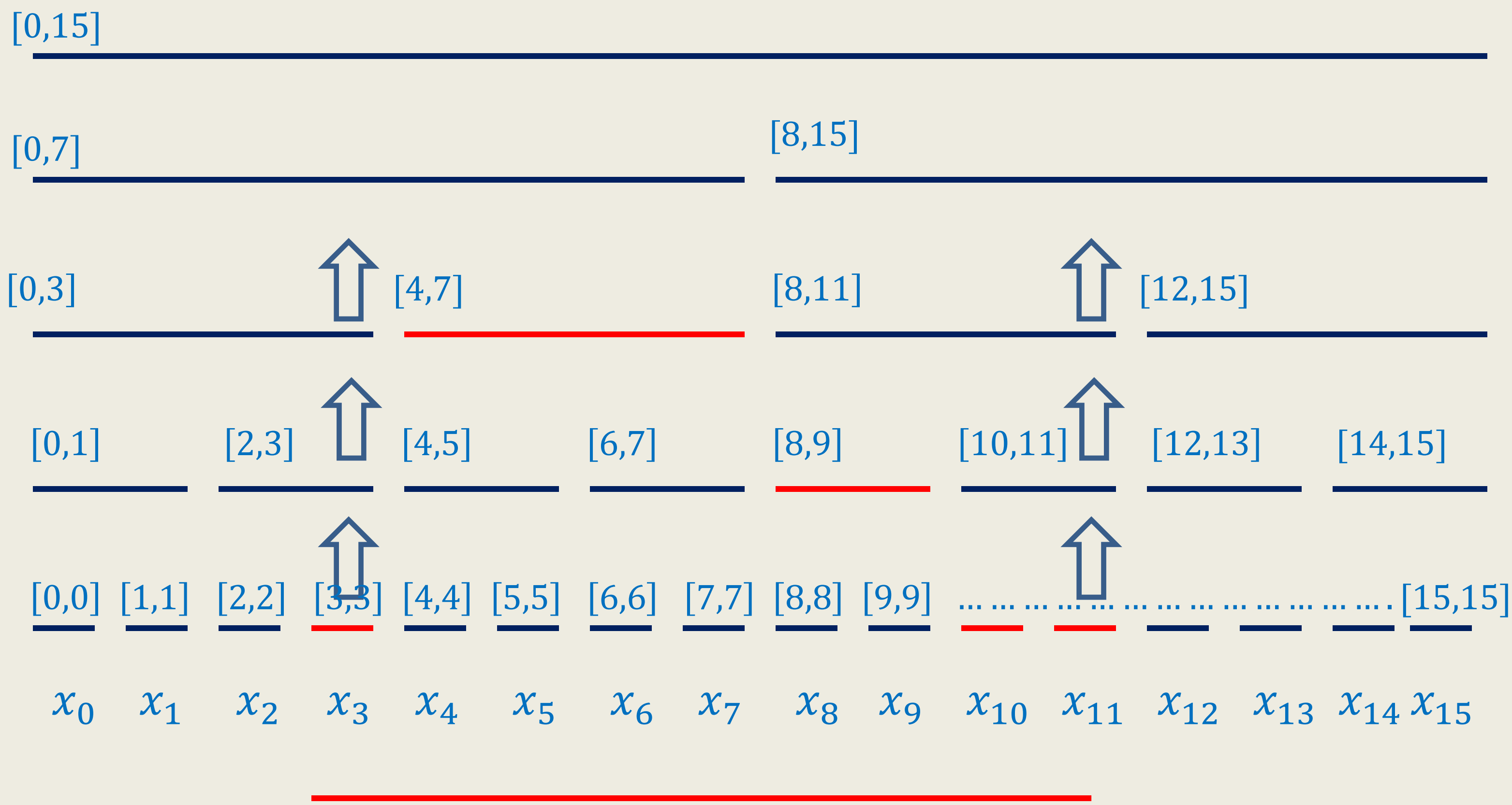


How to express [0, 12] ?

Extension to intervals

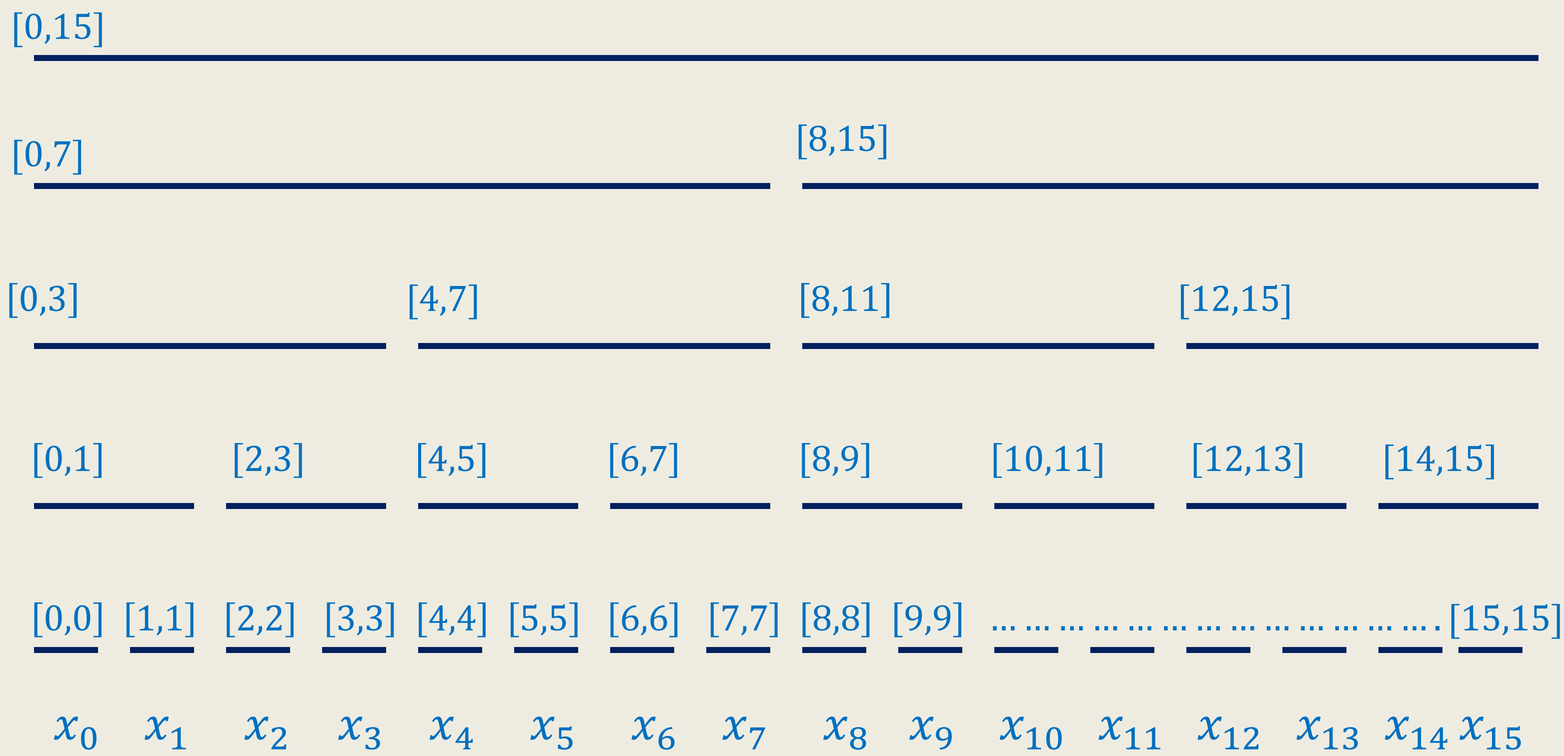
How to express [0, 12] ?

Extension to intervals



How to express [**3**, **11**] ?

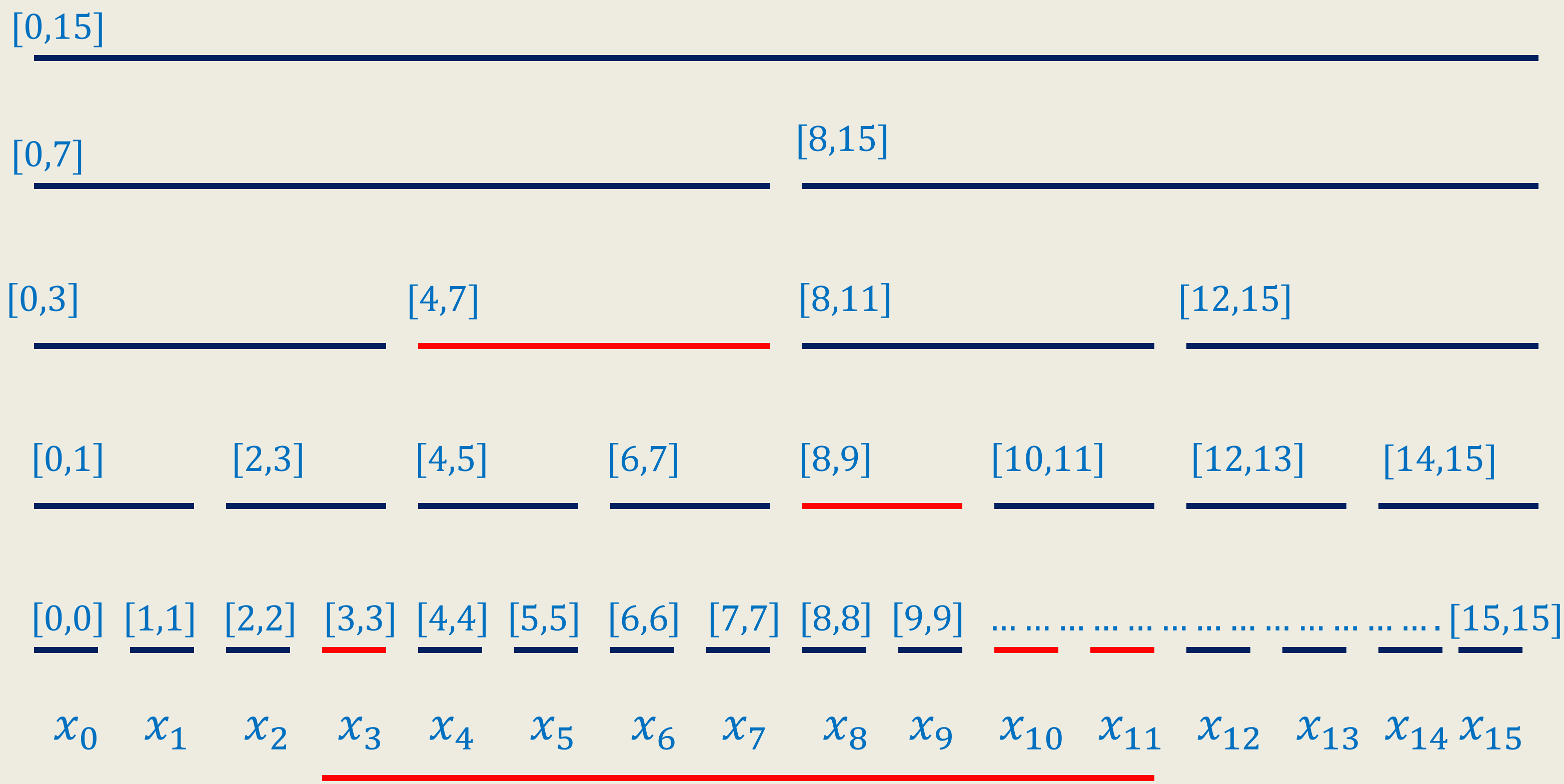
How to use this Observation to perform Multi-Increment(i, j, Δ) efficiently?



Observation: There are $2n$ intervals such that

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

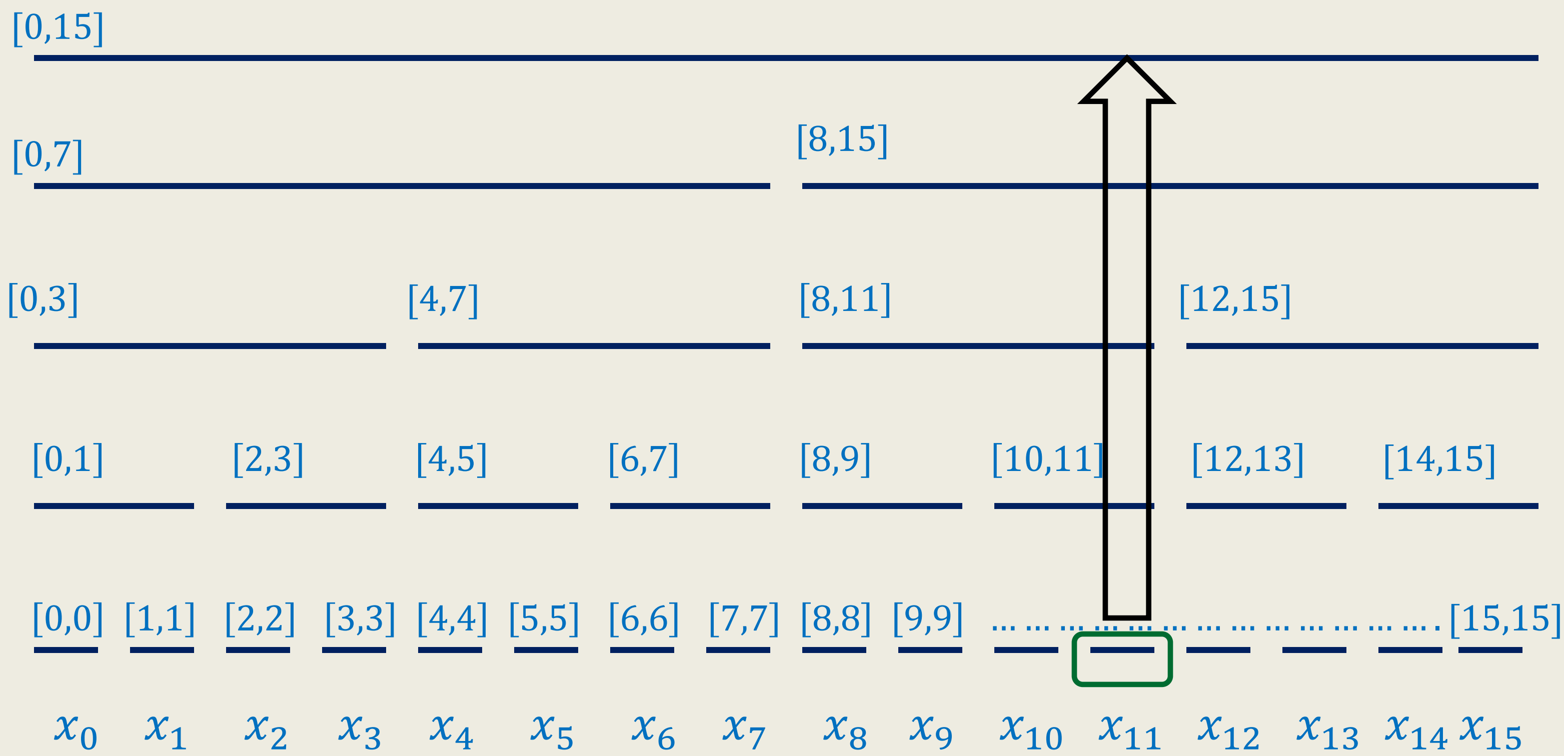
How to use this Observation to perform
Multi-Increment(i, j, Δ) efficiently?



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

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

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

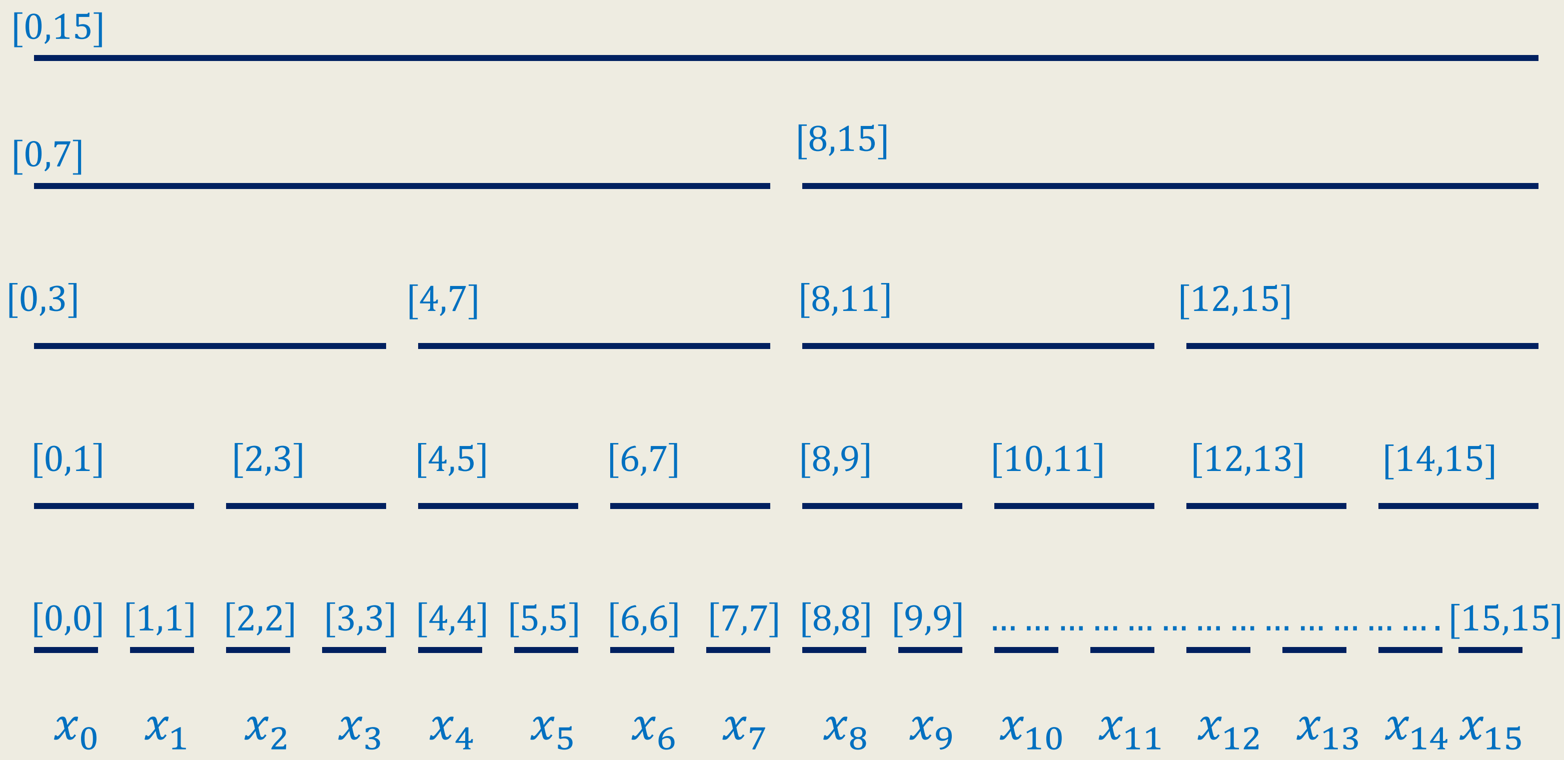


Maintain $2n$ intervals with a field **increment**

Multi-Increment(i, j, **Δ**) ➔ 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) ➔ 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.

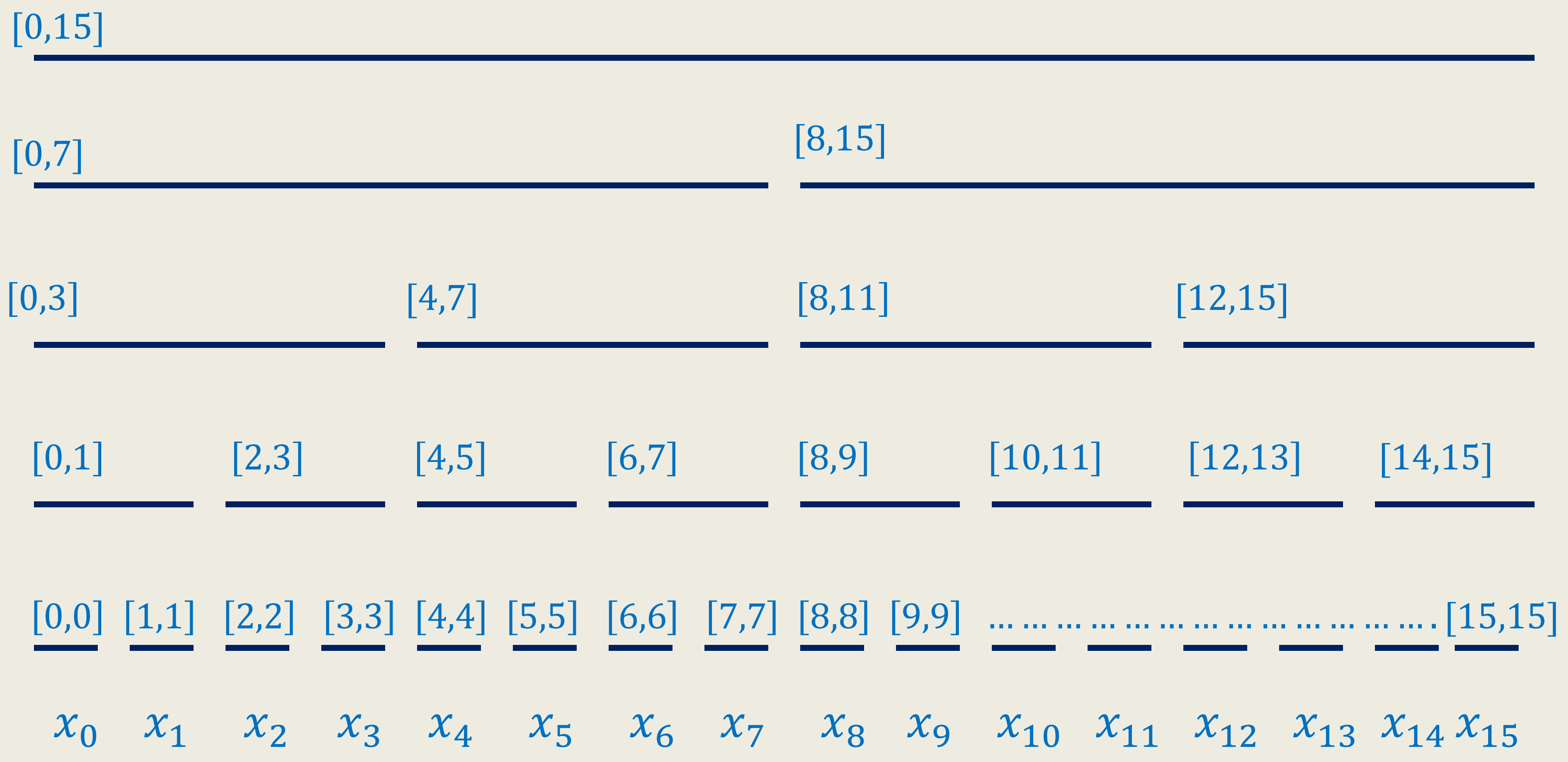
I have **reproduced** it for you again in the next slide

Have another look,

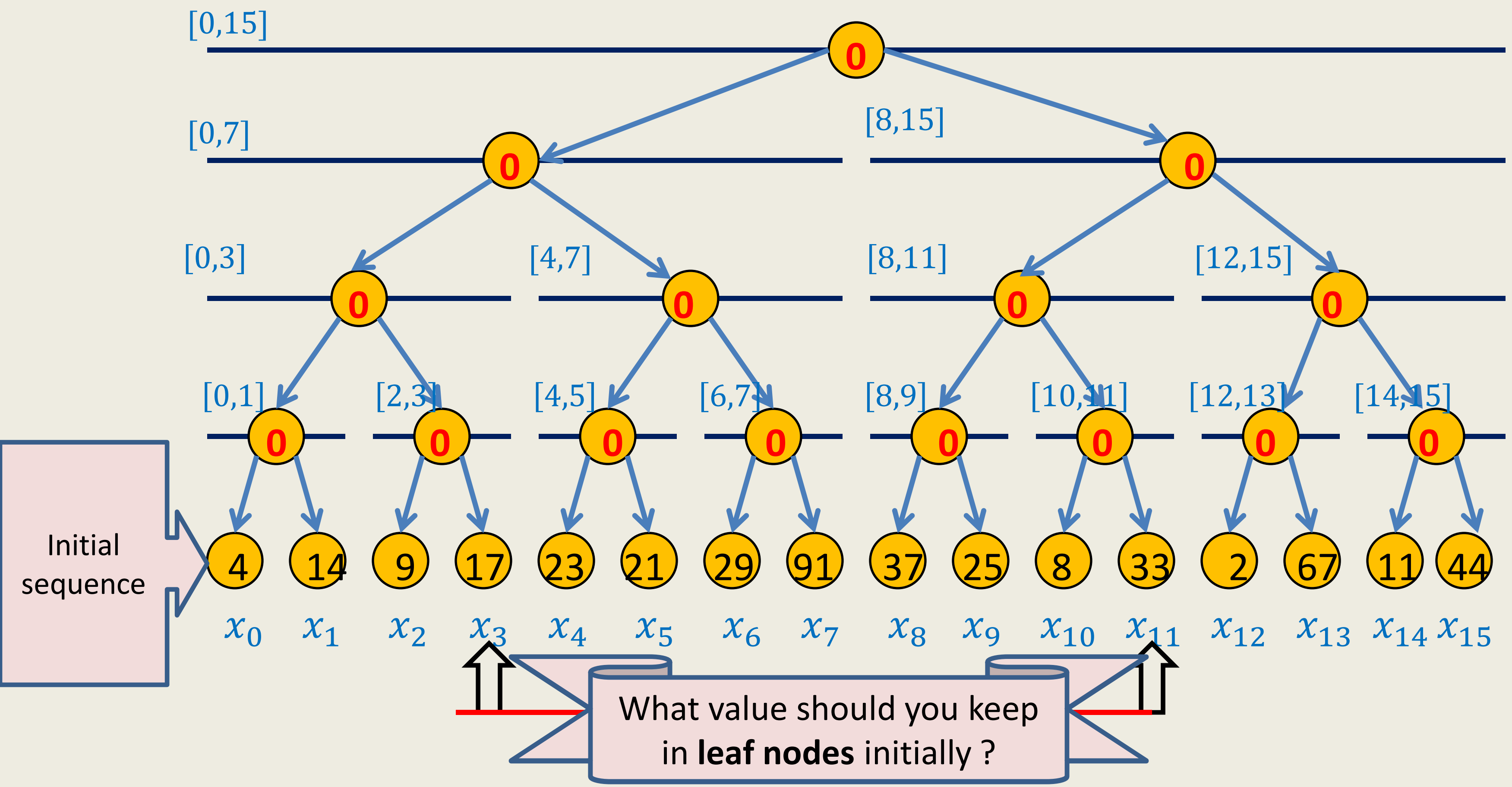
think for a while …

and then only proceed.

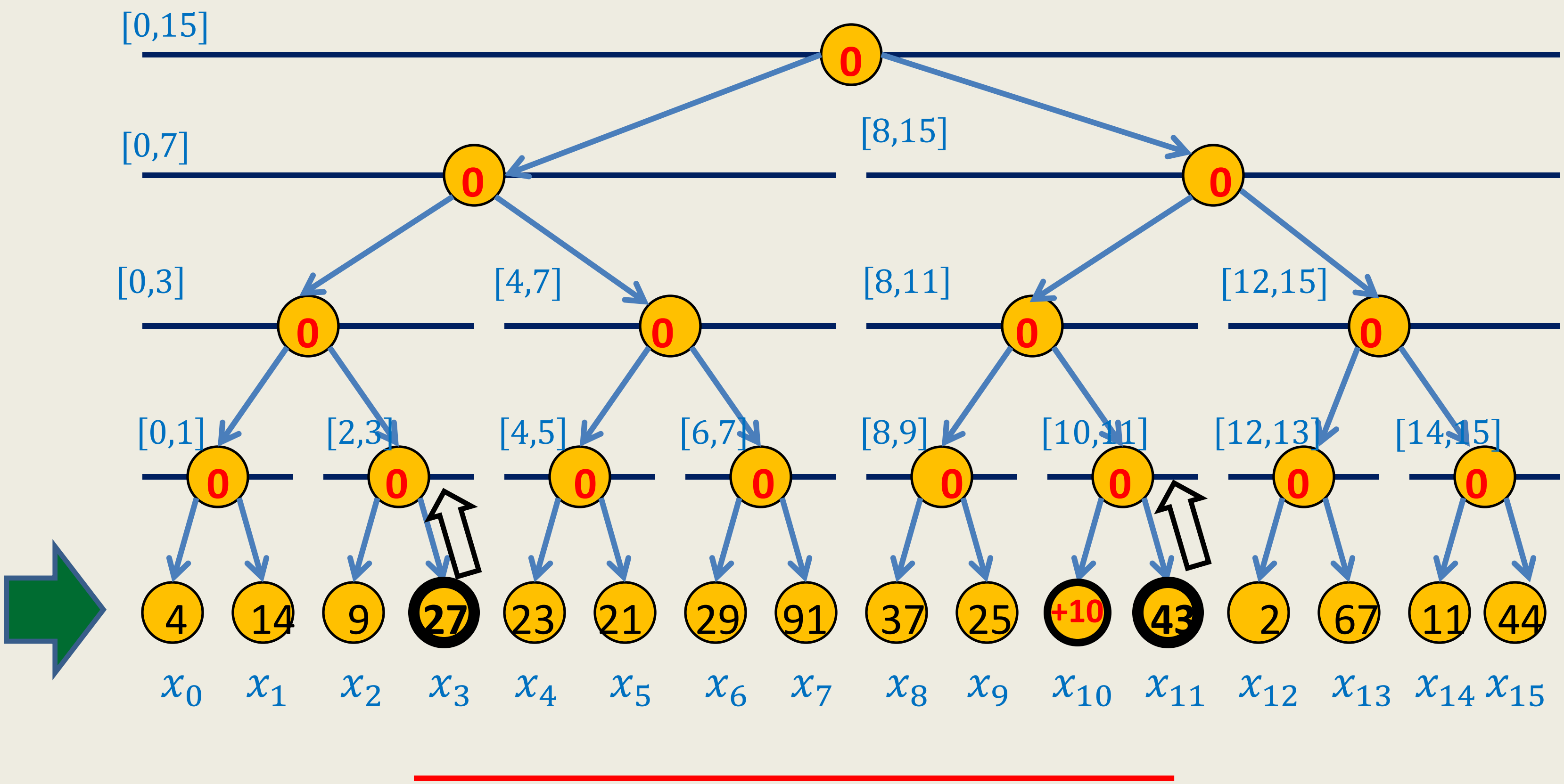
Which data structure emerges ?



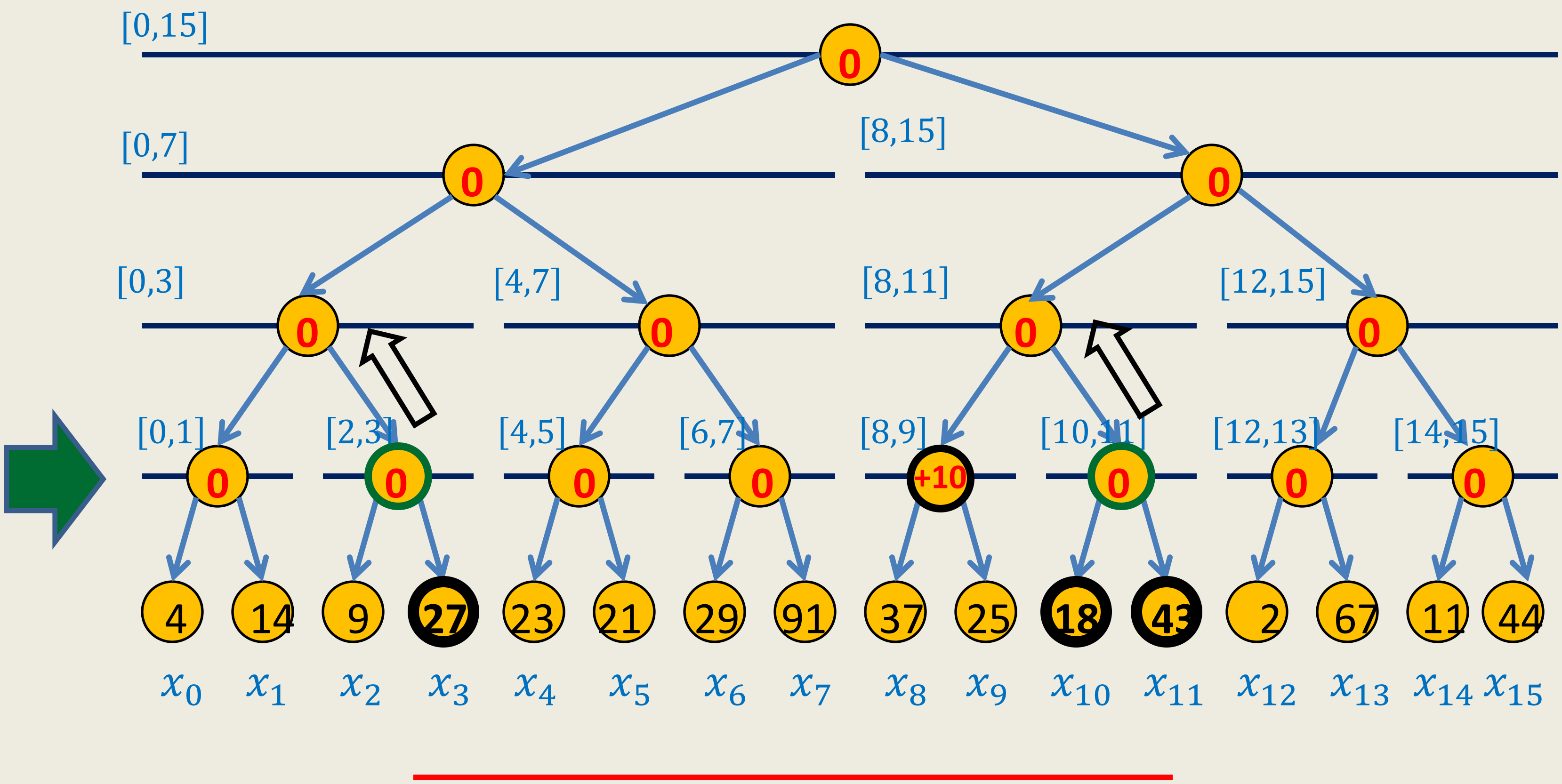
Isn't it a **Binary tree** that you thought ?



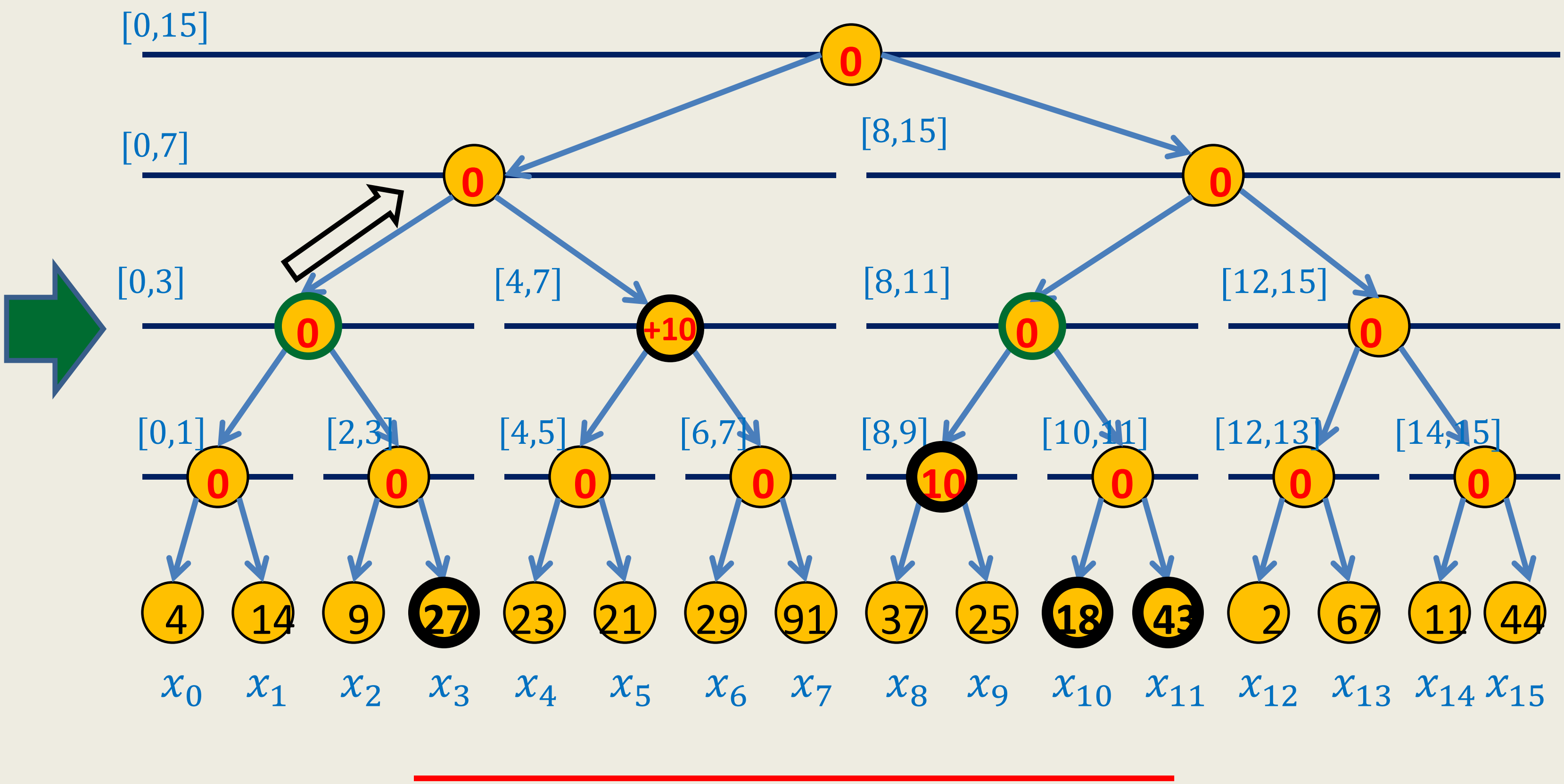
How to do Multi-Increment(3,11,10) ?



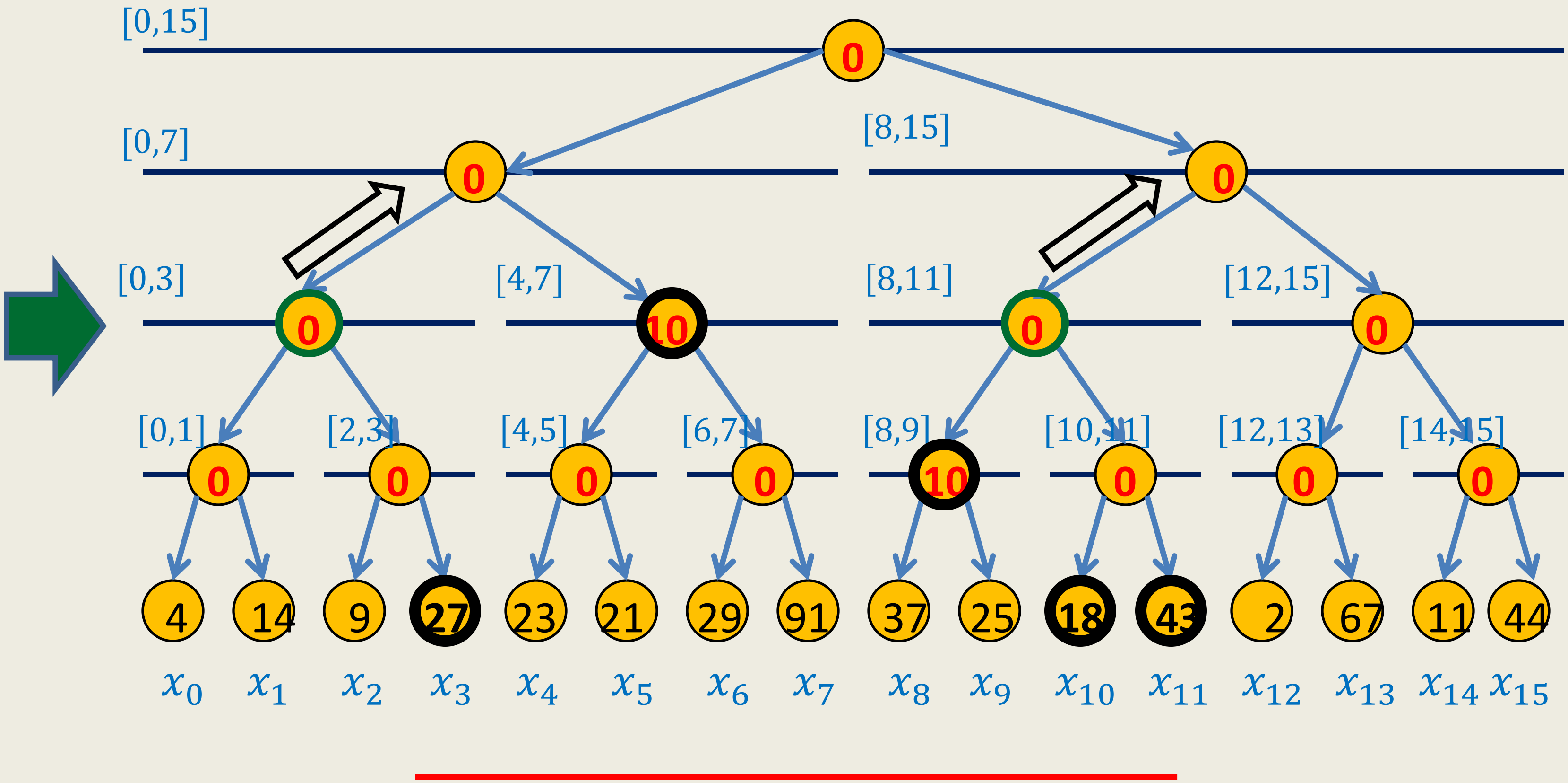
How to do Multi-Increment(3,11,10) ?



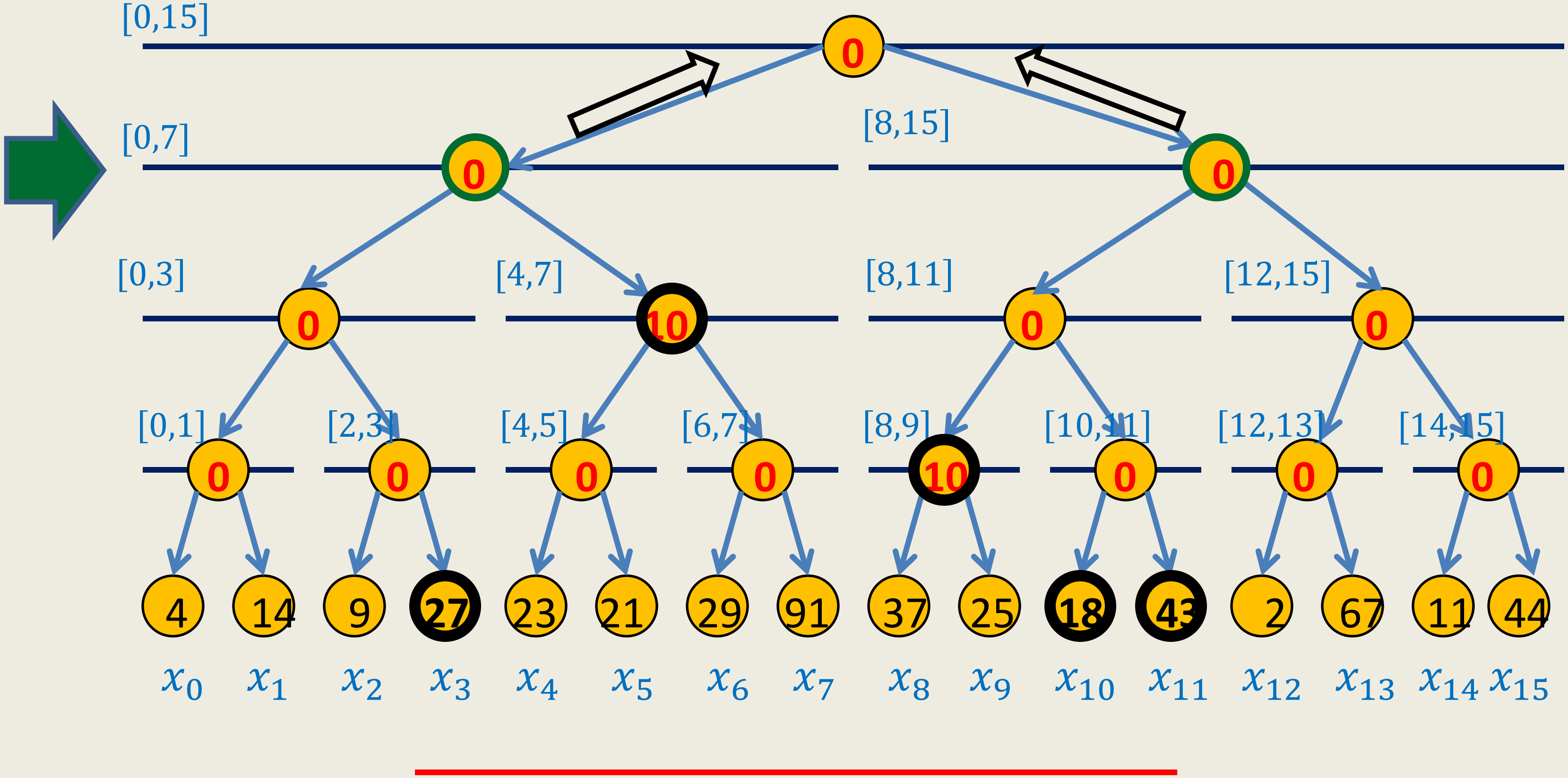
How to do Multi-Increment(3,11,10) ?



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

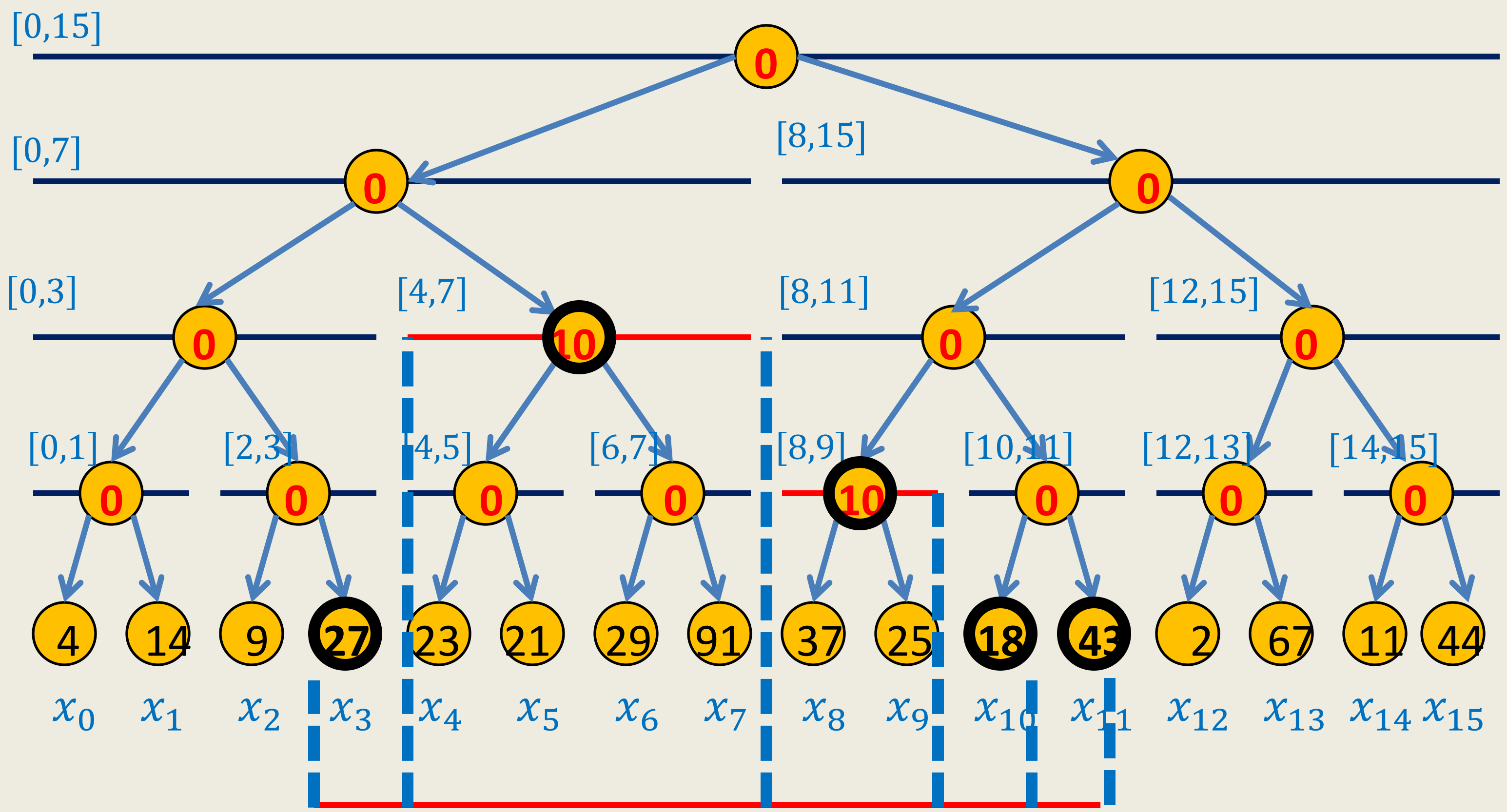


How to do Multi-Increment(3,11,10) ?



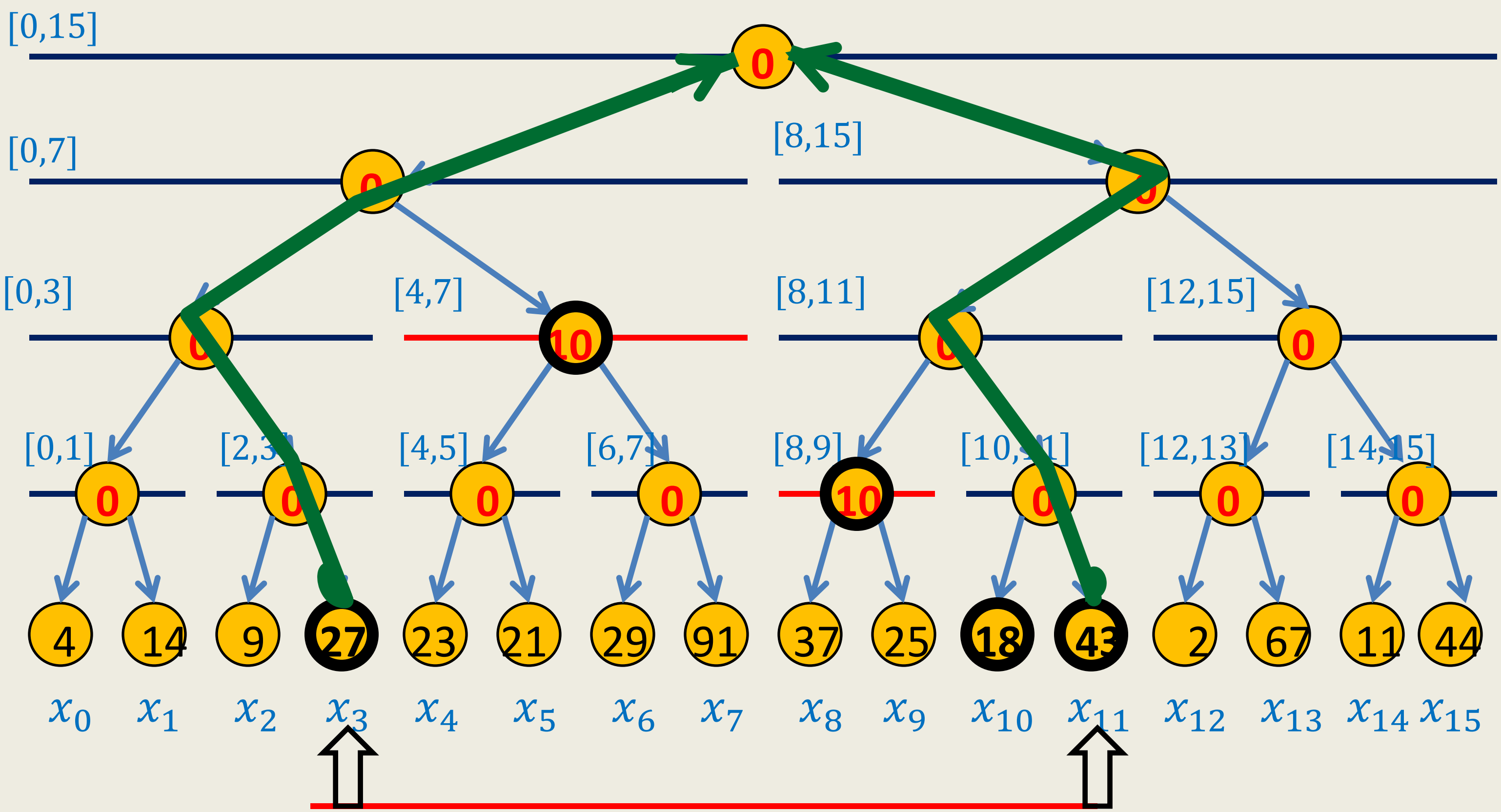
How to do Multi-Increment(3,11,10) ?

Are we done ?



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

Yes



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

What path was followed ?

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 parent(**u**) <> 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

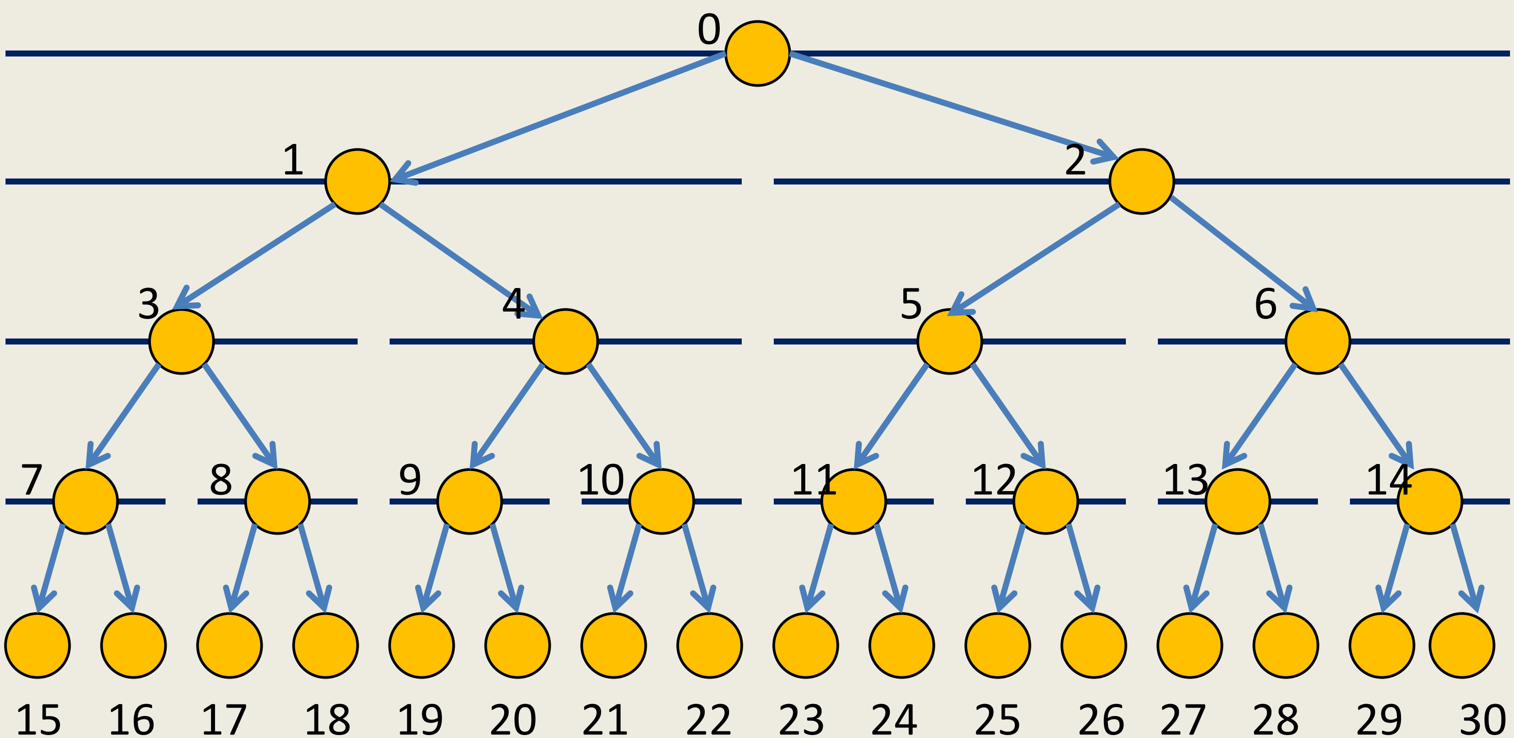
Sketch:

1. Let **u** be the leaf nodes corresponding to x_i.
2. **val** ← **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 2**n**-1.

Copy the sequence **S** = $\prec x_0, \ldots, x_{n-1} \succ$ into **A**$[n-1]$…**A**$[2n-2]$

Leaf node corresponding to x_i = **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)

MultiIncrement(i, j, Δ)

```
MultiIncrement(i, j,Δ)
    i ← (n − 1) + i ;
    j ← (n − 1) + j ;
    A(i) ← A(i) + Δ;
    If (j > i)
    {       A(j) ← A(j) + Δ;
            While( ⌊(i − 1)/2⌋ <> ⌊(j − 1)/2⌋ )
            {
                If(i%2=1)   A(i + 1) ← A(i + 1) + Δ;
                If(j%2=0)   A(j − 1) ← A(j − 1) + Δ;
                i ← ⌊(i − 1)/2⌋ ;
                j ← ⌊(j − 1)/2⌋ ;
            }
    }
```

Report(i)

```
Report(i )
    i ← (n − 1) + i ;
    val ← 0;
    While(            i > 0            )
    {
        val ← val + A[i];
        i ← ⌊(i − 1)/2⌋ ;
    }
    return val;
```

The solution of Multi-Increment Problem

Theorem: There exists a data structure of size **O**(n) for maintaining a sequence **S** = $\prec x_0, \ldots, x_{n-1} \succ$ such that each **Multi-Increment**() and **Report**() operation takes **O**(log n) time.

Problem 2

Dynamic Range-minima

Problem 2

Given an initial sequence **S** = $\prec x_0, \ldots, x_{n-1} \succ$ of numbers, maintain a compact data structure to perform the following operations efficiently for any $0 \le \boldsymbol{i} < \boldsymbol{j} < \boldsymbol{n}$.

- **ReportMin**($\boldsymbol{i}$, $\boldsymbol{j}$):

  Report the minimum element from $\{x_k \mid \text{for each } \boldsymbol{i} \le \boldsymbol{k} \le \boldsymbol{j}\}$

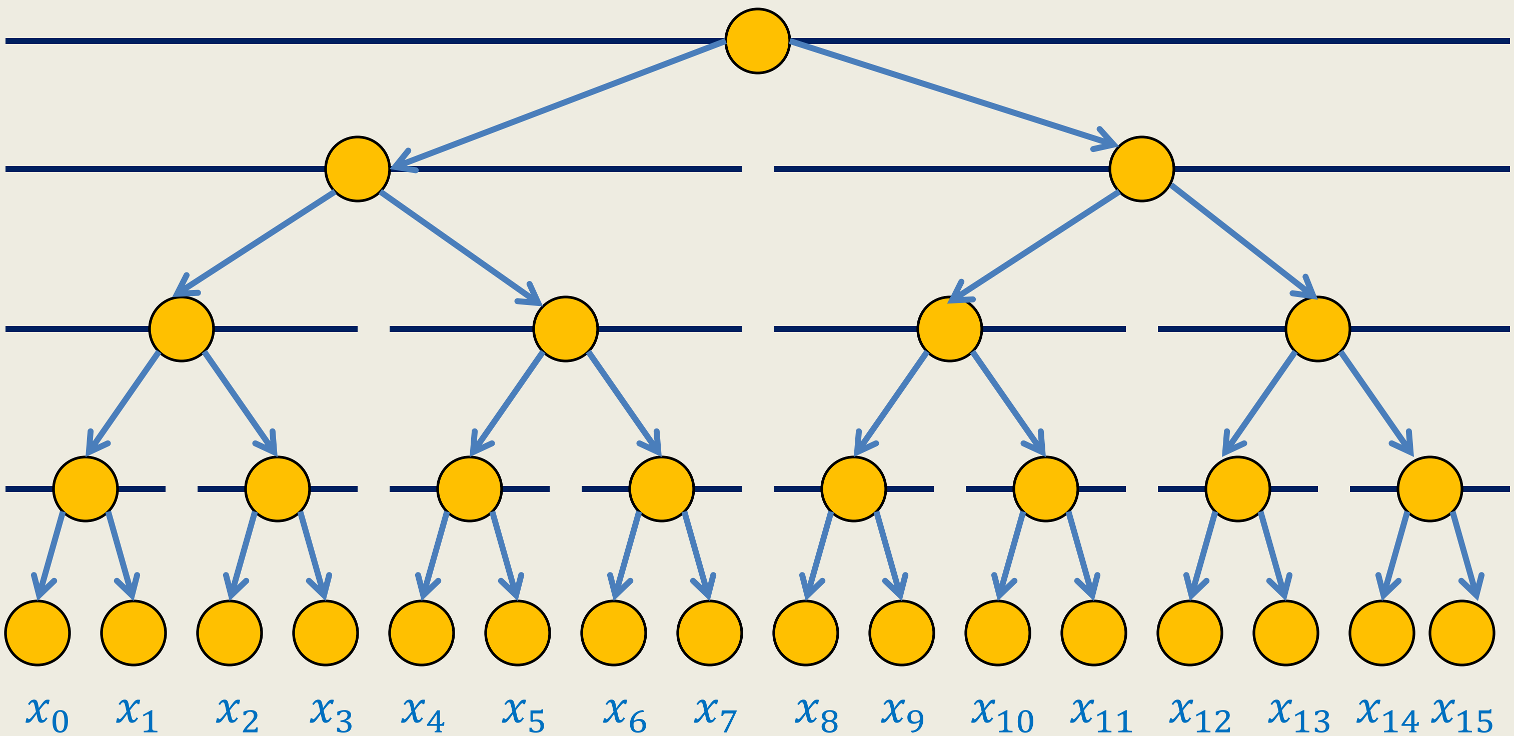
- **Update**($\boldsymbol{i}$, **a**):

  a becomes the new value of x_i.

AIM:

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

Efficient dynamic range minima



What **to store** at internal nodes ?

How to perform **ReportMin**($\boldsymbol{i}$, $\boldsymbol{j}$) ?

How to perform **Update**($\boldsymbol{i}$, **a**) ?

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