Complete binary trees

Outline

Introducing complete binary trees

- Background
- Definitions
- Examples
- Logarithmic height
- Array storage

Background

A perfect binary tree has ideal properties but restricted in the number of nodes: $n = 2^{h+1} - 1$

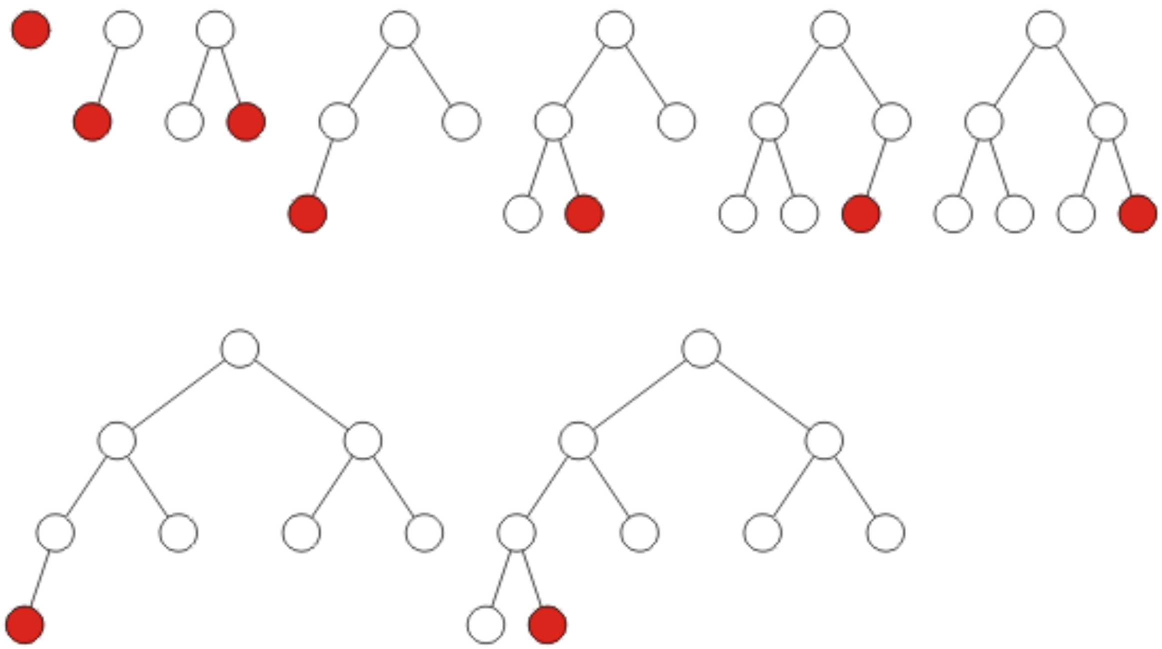
$1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, ...$

We require binary trees which are

- Similar to perfect binary trees, but
- Defined for all n

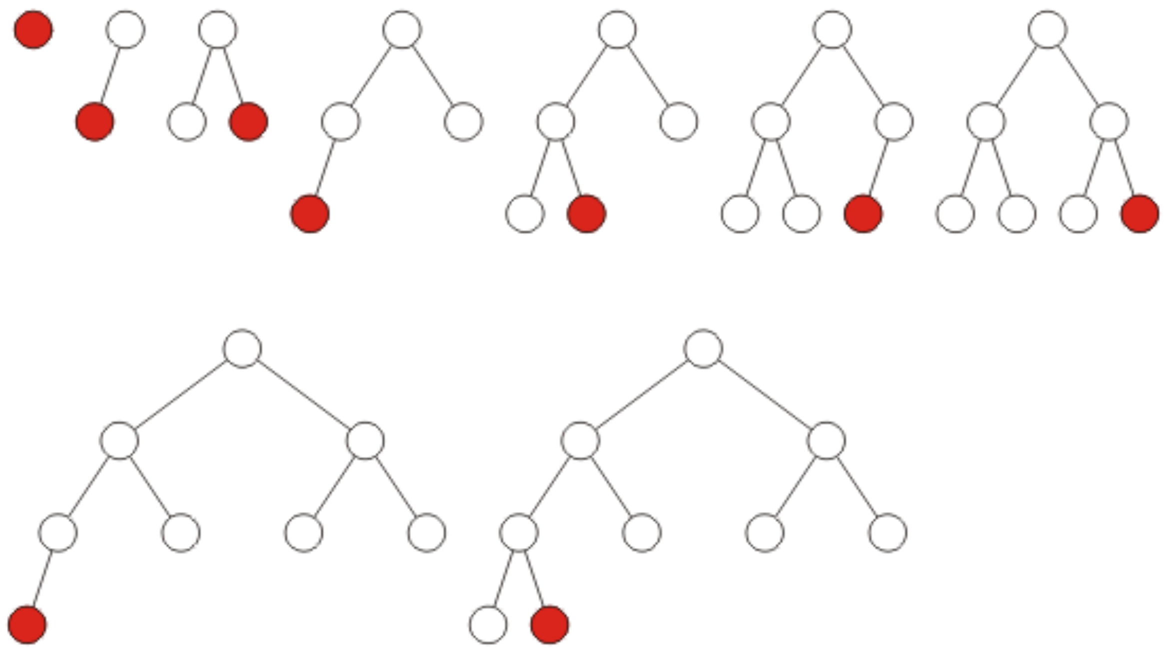
Definition

A complete binary tree filled at each depth from left to right:



Definition

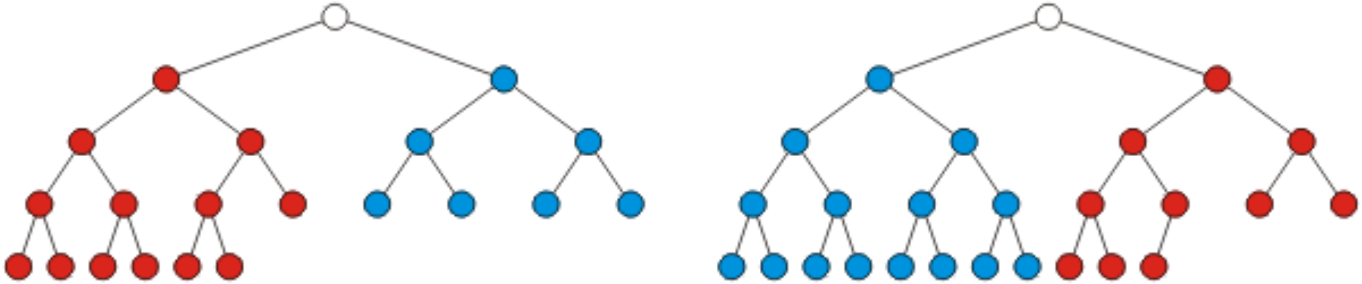
The order is identical to that of a breadth-first traversal



Recursive Definition

Recursive definition: a binary tree with a single node is a complete binary tree of height $h = 0$ and a complete binary tree of height h is a tree where either:

- The left sub-tree is a **complete tree** of height $h - 1$ and the right sub-tree is a **perfect tree** of height $h - 2$, or
- The left sub-tree is **perfect tree** with height $h - 1$ and the right sub-tree is **complete tree** with height $h - 1$



Height

Theorem

The height of a complete binary tree with n nodes is $h = \lfloor \log(n) \rfloor$

Proof:

- Base case:
  - When $n = 1$ then $\lfloor \lg(1) \rfloor = 0$ and a tree with one node is a complete tree with height $h = 0$
- Inductive step:
  - Assume that a complete tree with n nodes has height $\lfloor \log(n) \rfloor$
  - Must show that $\lfloor \log(n + 1) \rfloor$ gives the height of a complete tree with $n + 1$ nodes
  - Two cases:
    - If the tree with n nodes is perfect, and
    - If the tree with n nodes is complete but not perfect

Height

Following proof is optional material

Note that **lg means log base 2**

Case 1 (the tree is perfect):

- If it is a perfect tree then
  - Adding one more node must increase the height
- Before the insertion, it had $n = 2^{h+1} - 1$ nodes:

$$2^h < 2^{h+1} - 1 < 2^{h+1}$$

$$h = \lg\left(2^h\right) < \lg\left(2^{h+1} - 1\right) < \lg\left(2^{h+1}\right) = h + 1$$

$$h \le \left\lfloor \lg\left(2^{h+1} - 1\right) \right\rfloor < h + 1$$

- Thus, $\left\lfloor \lg(n) \right\rfloor = h$
- However, $\left\lfloor \lg(n+1) \right\rfloor = \left\lfloor \lg\left(2^{h+1} - 1 + 1\right) \right\rfloor = \left\lfloor \lg\left(2^{h+1}\right) \right\rfloor = h + 1$

Height

Following proof is optional material

Case 2 (the tree is complete but not perfect):

– If it is not a perfect tree then

$$2^h \le n < 2^{h+1} - 1$$

$$2^h + 1 \le n + 1 < 2^{h+1}$$

$$h < \lg\left(2^h + 1\right) \le \lg\left(n+1\right) < \lg\left(2^{h+1}\right) = h + 1$$

$$h \le \left\lfloor \lg\left(2^h + 1\right) \right\rfloor \le \left\lfloor \lg\left(n+1\right) \right\rfloor < h + 1$$

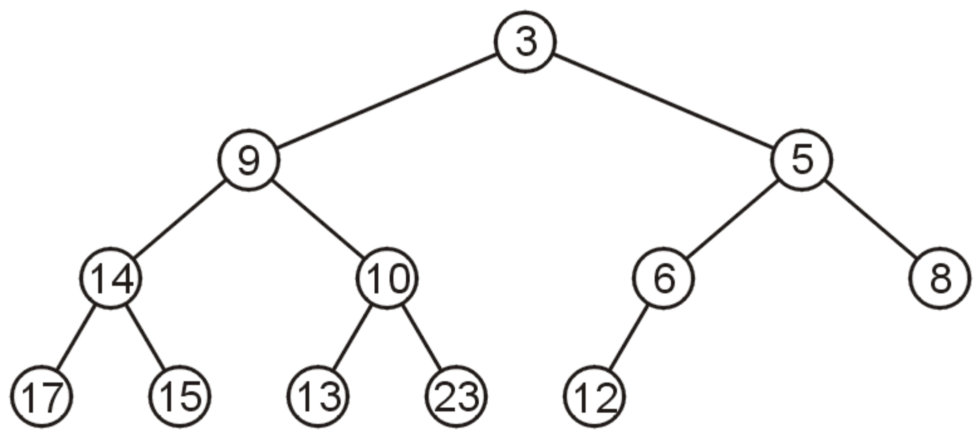
– Consequently, the height is unchanged: $\lfloor \lg(n + 1)\rfloor = h$

By mathematical induction, the statement must be true for all $n \ge 1$

Array storage

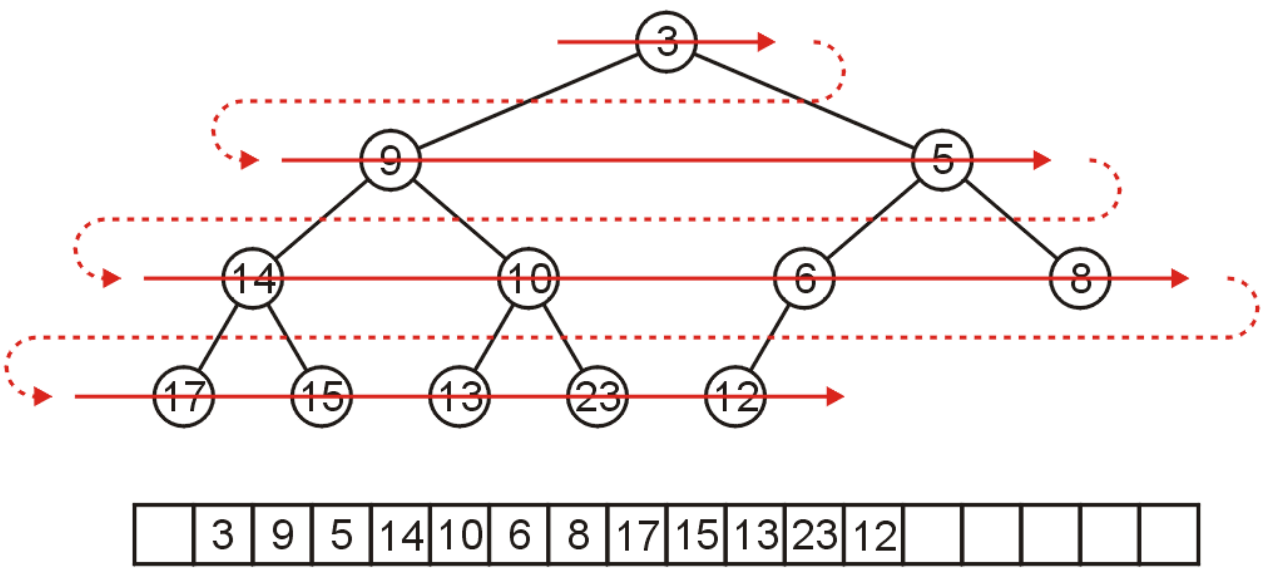
We are able to store a complete tree as an array

– Traverse the tree in breadth-first order, placing the entries into the array



Array storage

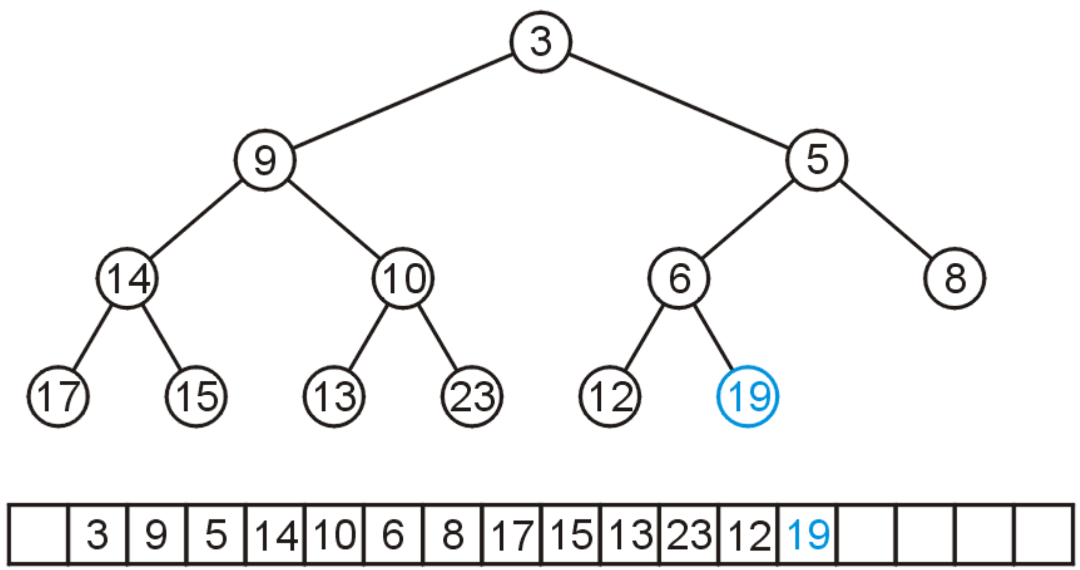
We can store this in an array after a quick traversal:



Please note that 0^{th} index is left null for convenience

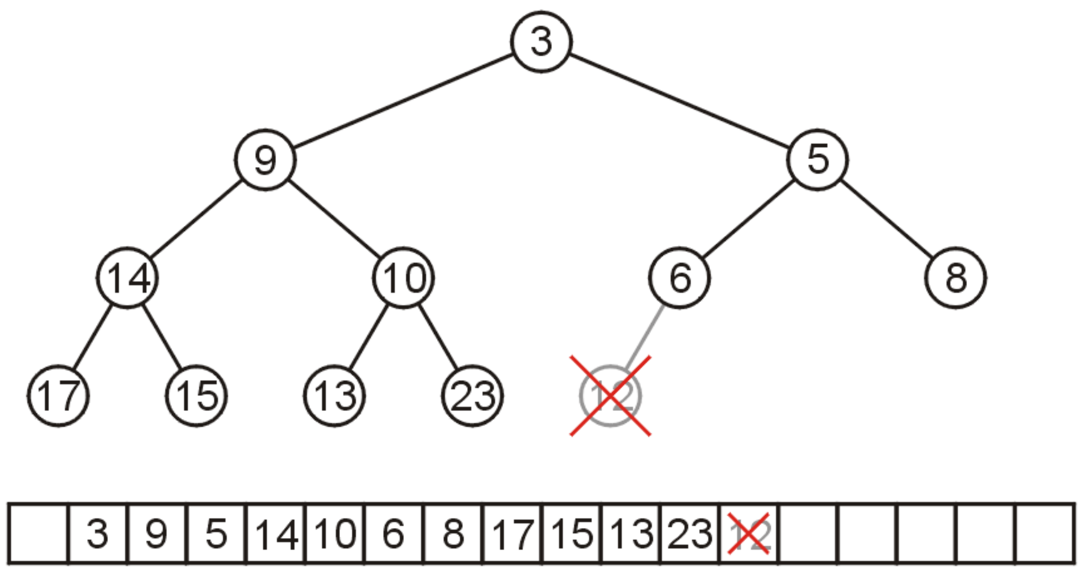
Array storage

To insert another node while maintaining the complete-binary-tree structure, we must insert into the next array location



Array storage

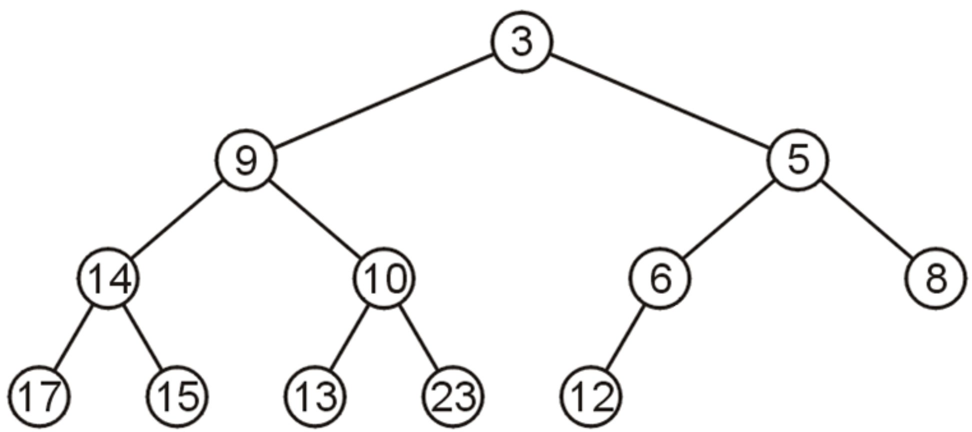
To remove a node while keeping the complete-tree structure, we must remove the last element in the array



Array storage

Leaving the first entry blank yields a bonus:

- The children of the node with index k are in $2k$ and $2k + 1$
- The parent of node with index k is in $k \div 2$

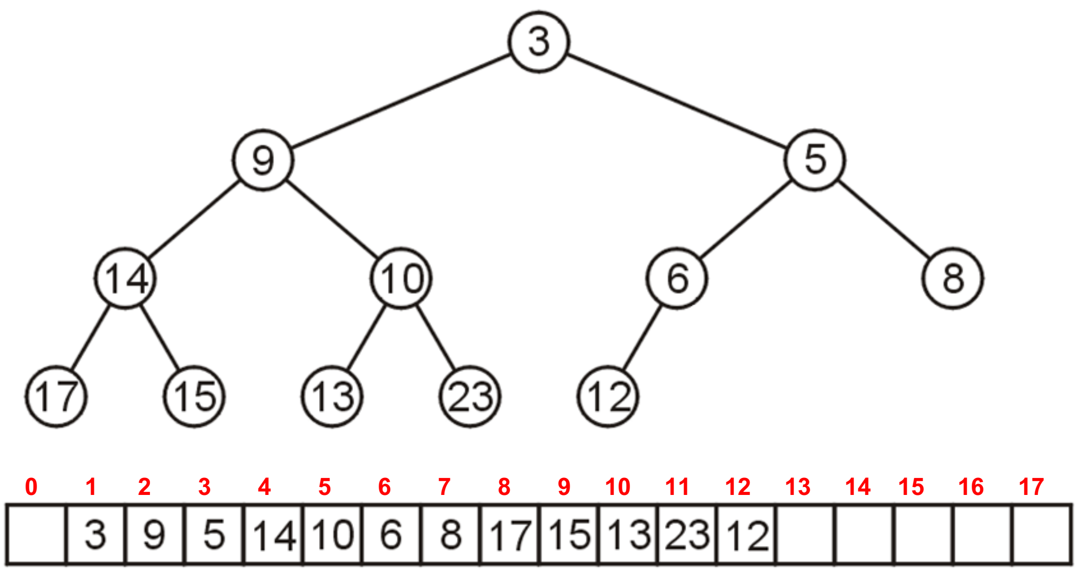


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	3	9	5	14	10	6	8	17	15	13	23	12					

Array storage

Leaving the first entry blank yields a bonus:

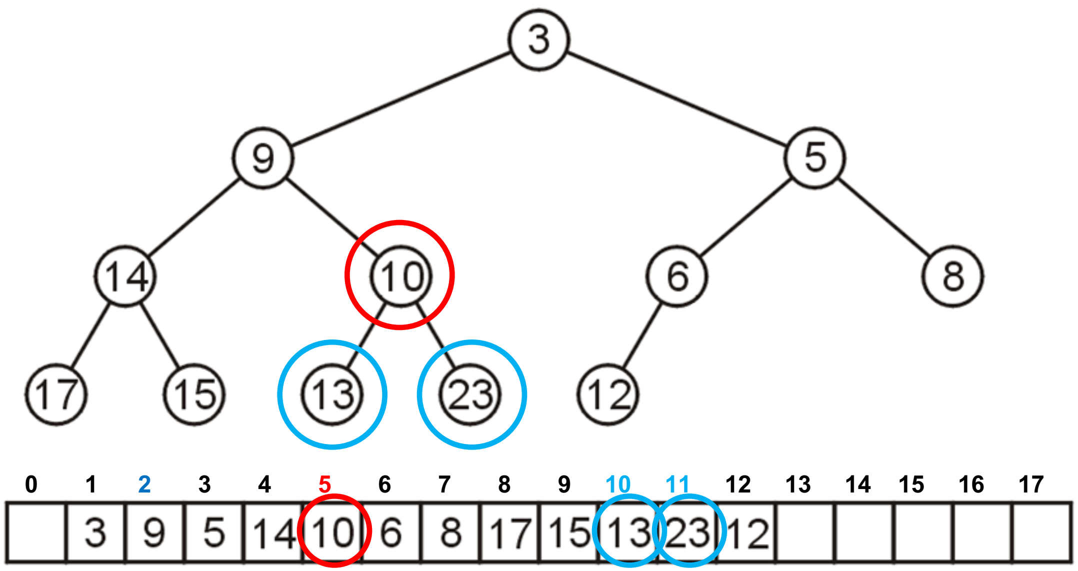
– It simplifies the calculations:



Array storage

For example, node 10 has index 5:

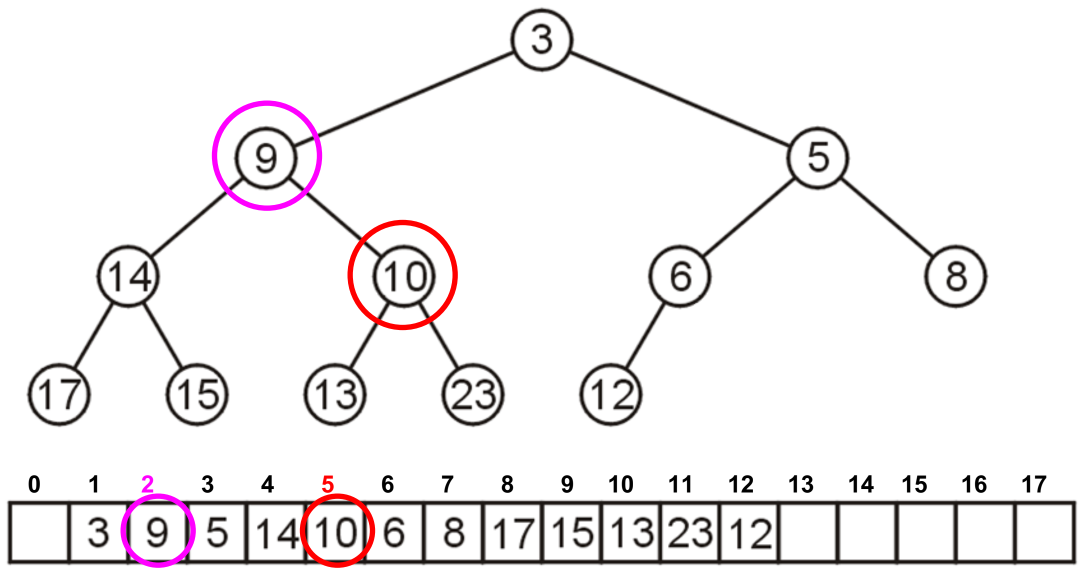
– Its children 13 and 23 have indices 10 and 11, respectively



Array storage

For example, node 10 has index 5:

- Its children 13 and 23 have indices 10 and 11, respectively
- Its parent is node 9 with index 5/2 = **2**



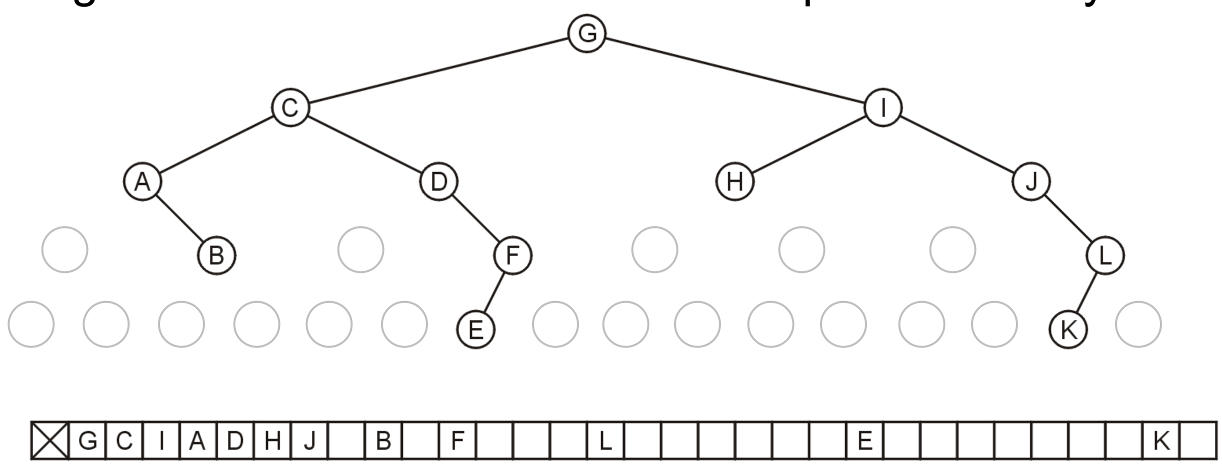
Array storage

Question: why not store any tree as an array using breadth-first traversals?

– There is a significant potential for a lot of wasted memory

Consider this tree with 12 nodes would require an array of size 32

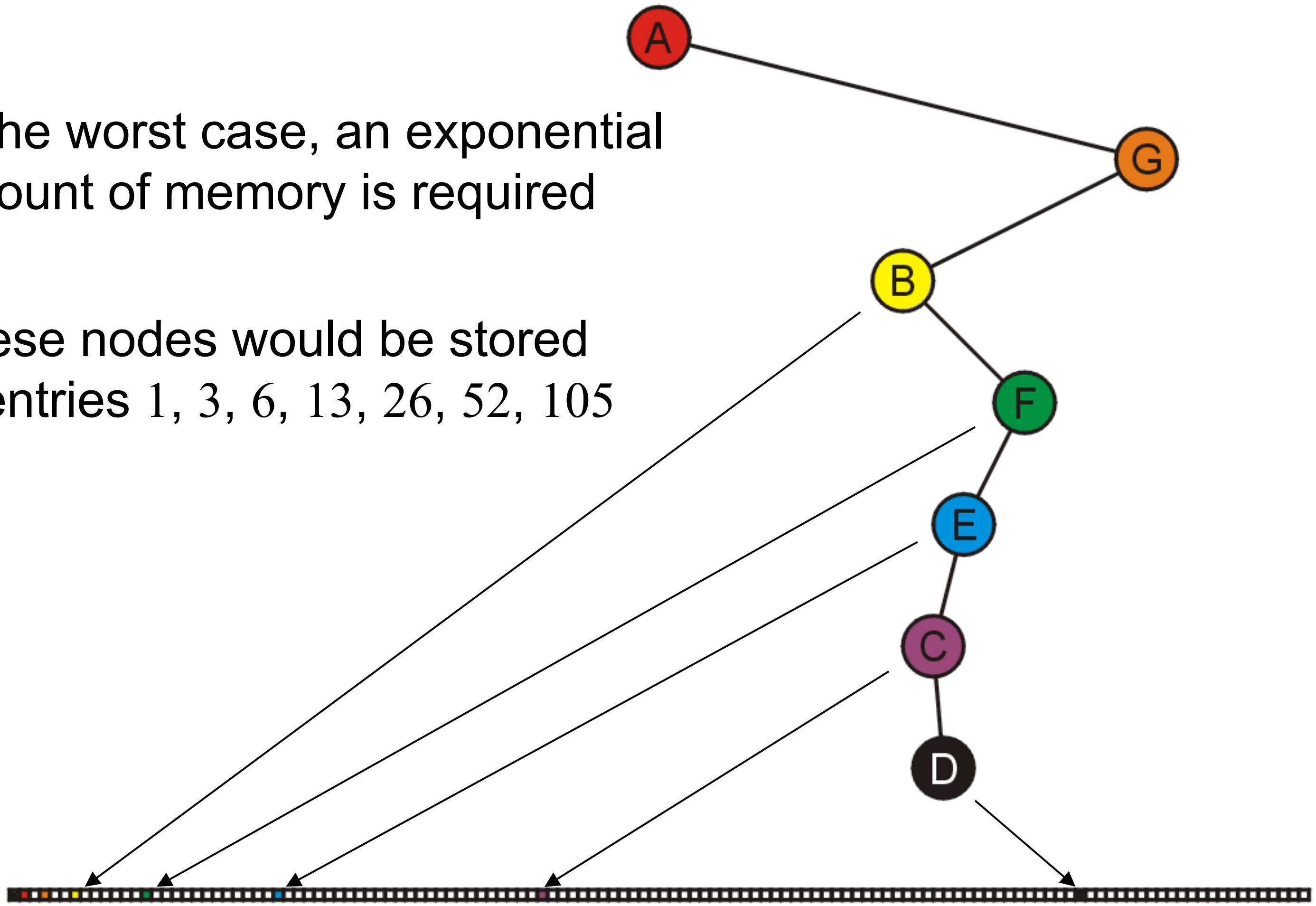
– Adding a child to node K doubles the required memory



Array storage

In the worst case, an exponential amount of memory is required

These nodes would be stored in entries 1, 3, 6, 13, 26, 52, 105



Summary

In this topic, we have covered the concept of a complete binary tree:

- A useful relaxation of the concept of a perfect binary tree
- It has a compact array representation