Lecture # 18

BuildHeap

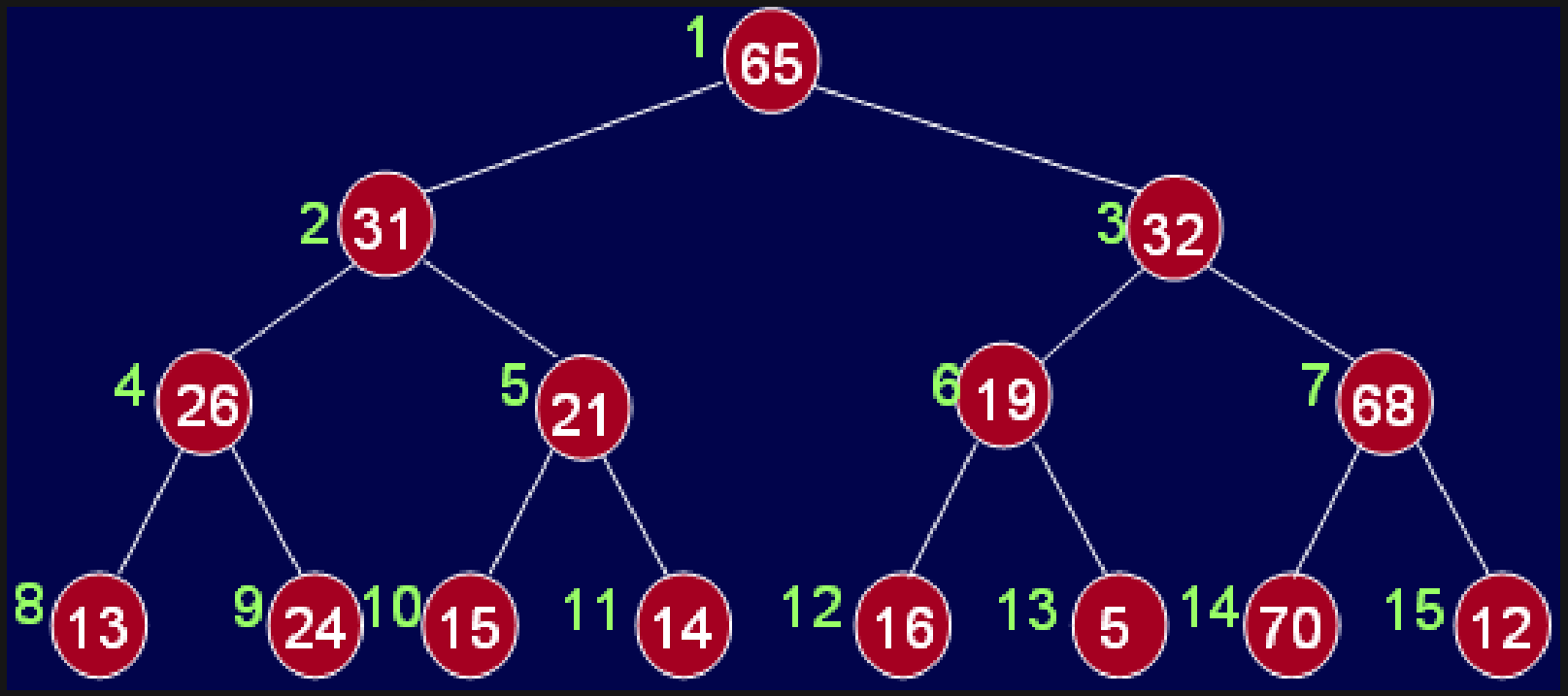
- Suppose we are given as input N keys (or items) and we want to build a heap of the keys.
- Obviously, this can be done with N successive *inserts*.

BuildHeap

- Initial data (N=15)

BuildHeap

- Initial data (N=15)



	65	31	32	26	21	19	68	13	24	15	14	16	5	70	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

BuildHeap

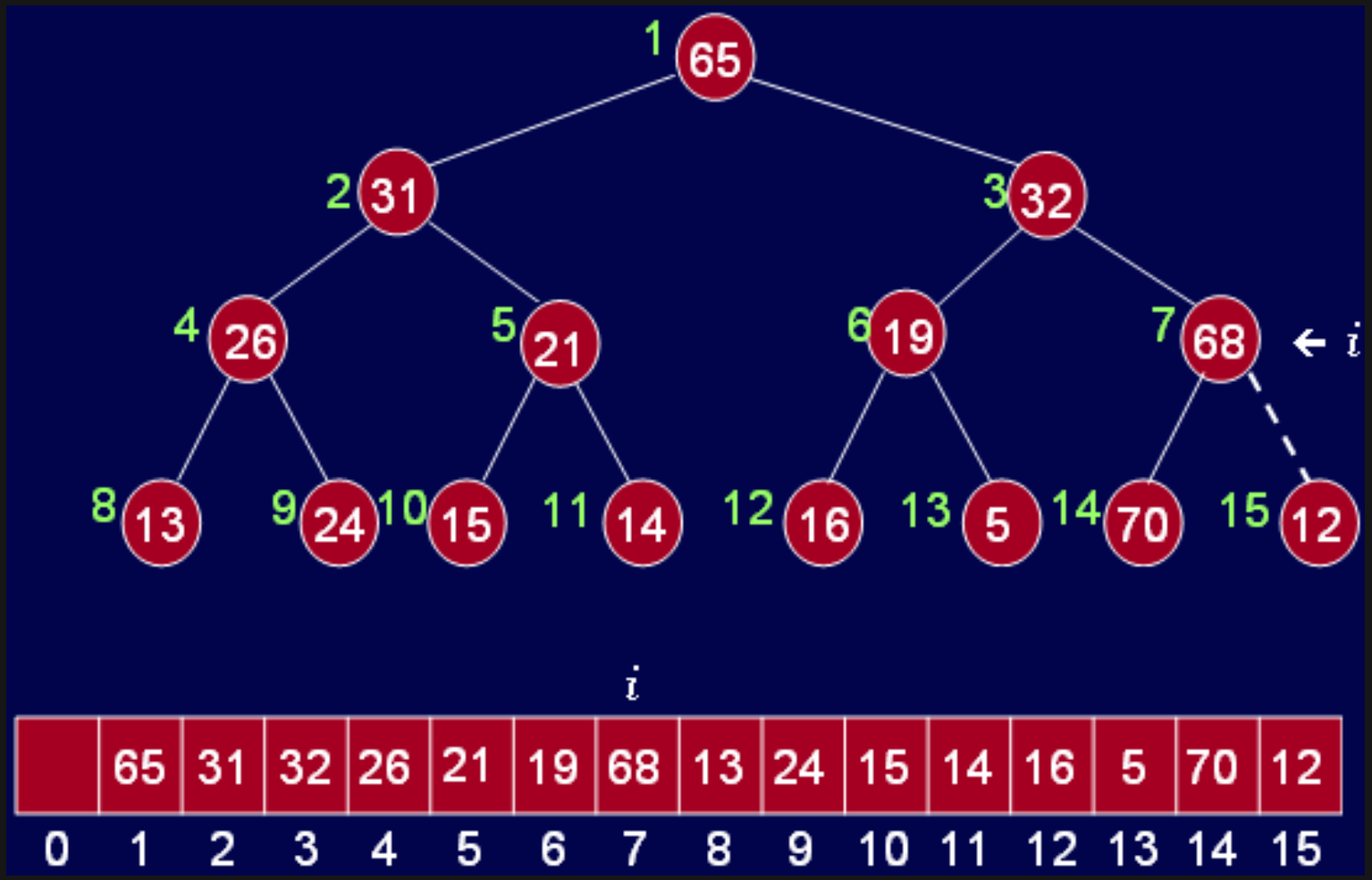
- The general algorithm is to place the N keys in an array and consider it to be an unordered binary tree.
- The following algorithm will build a heap out of N keys.

```
for( i = N/2; i > 0; i-- )
        heapify(i);
```

BuildHeap

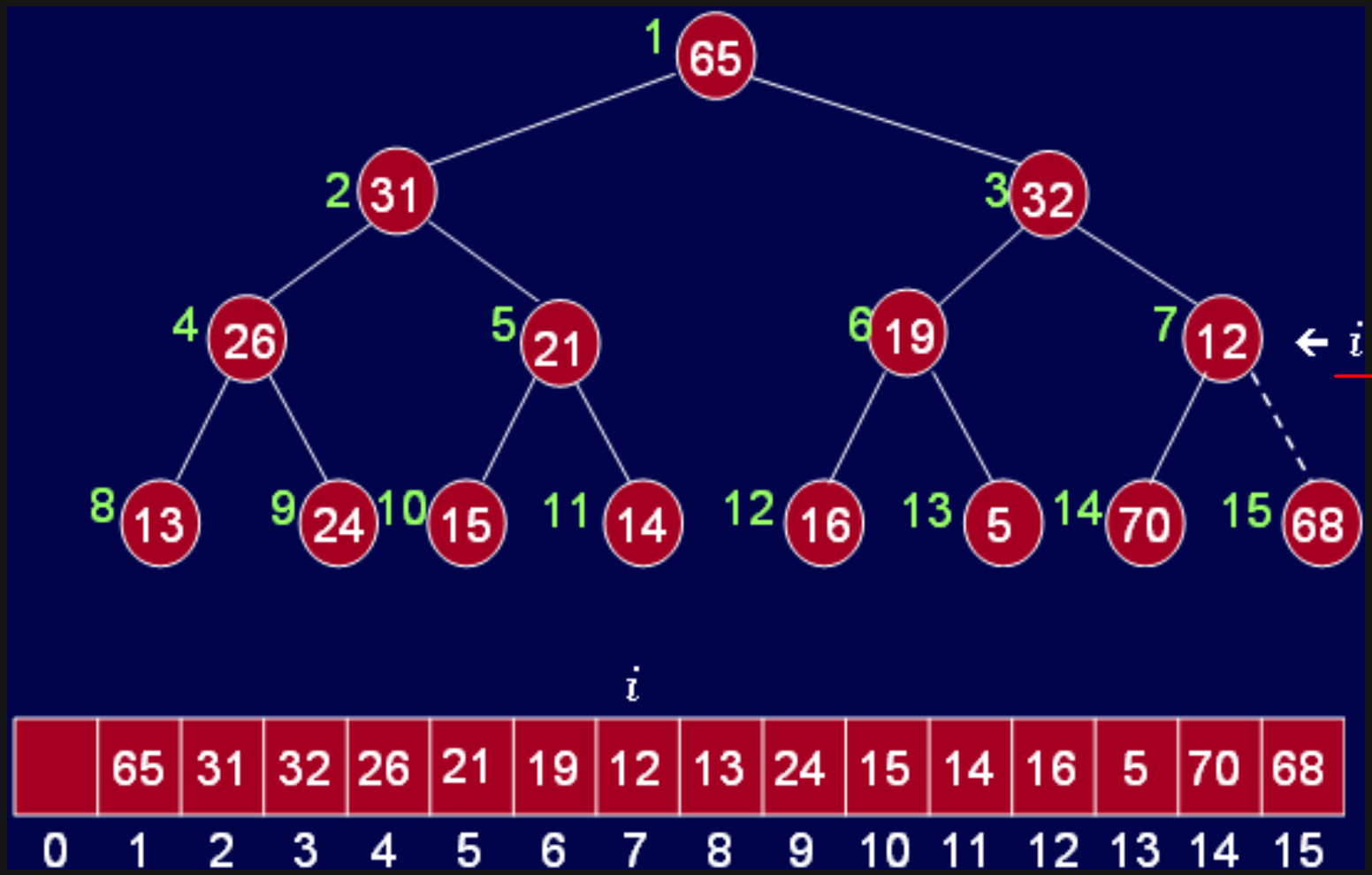
- i = 15/2 = 7

Why i=n/2?



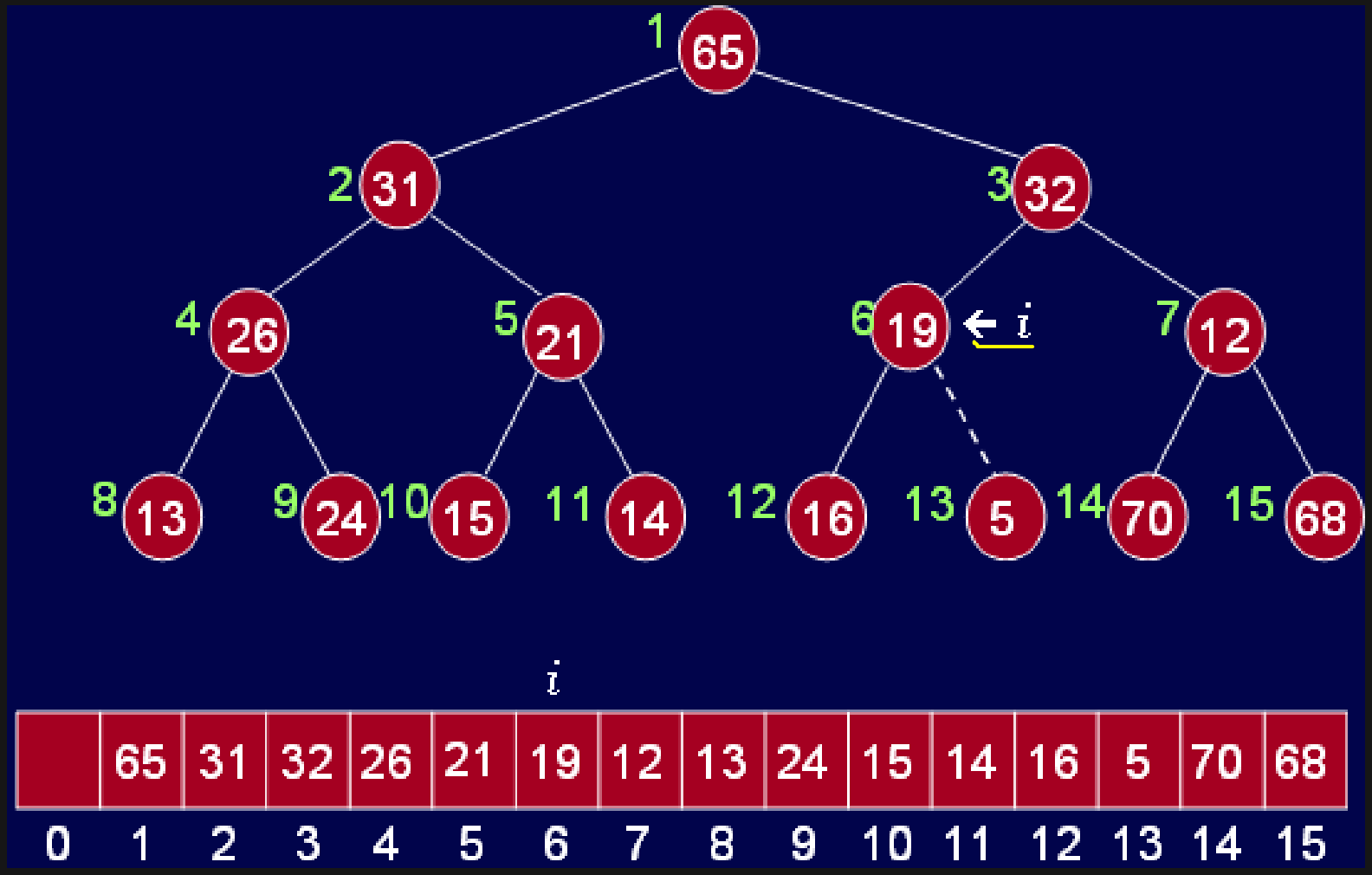
BuildHeap

- i = 15/2 = 7



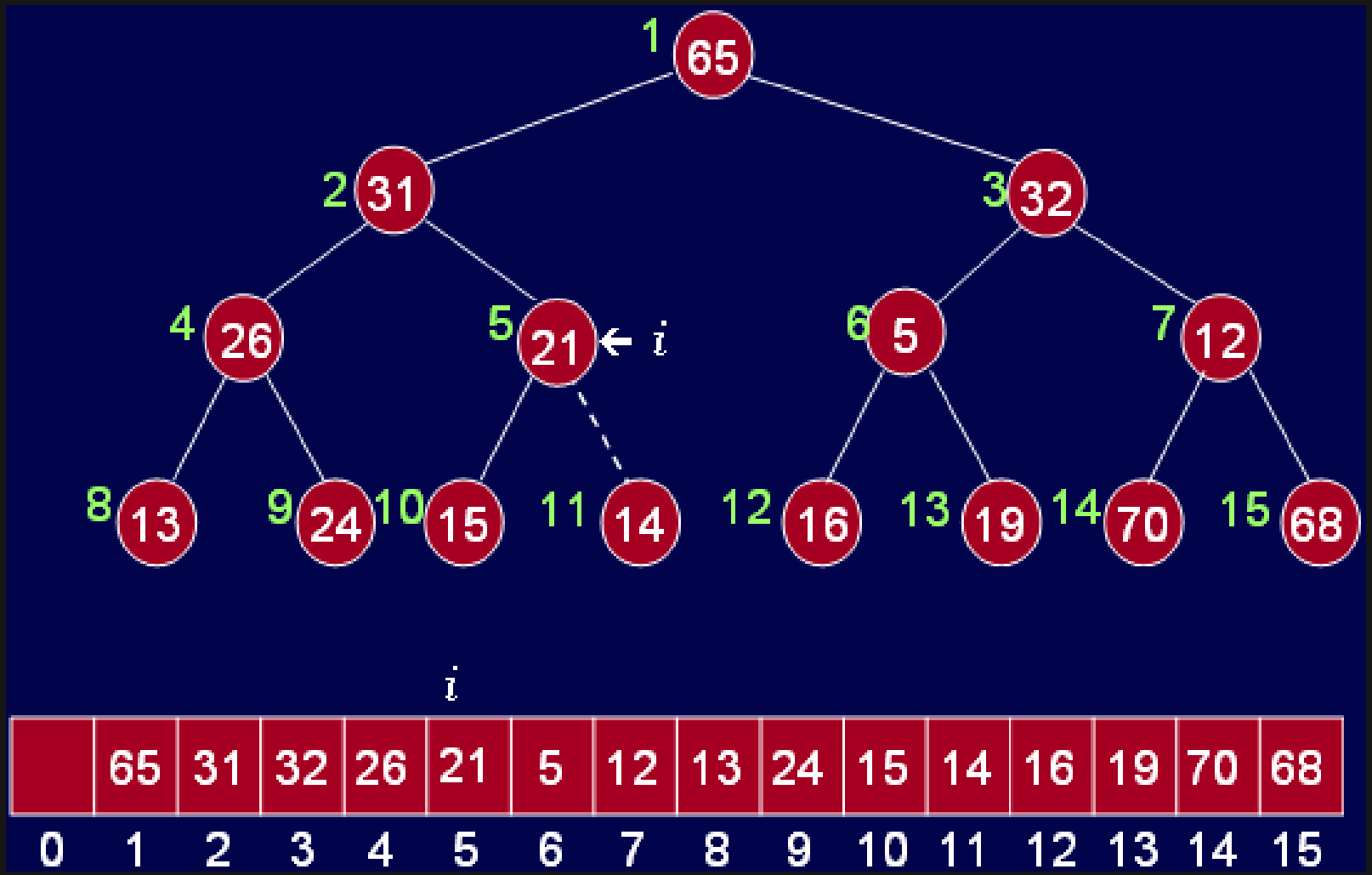
BuildHeap

- i = 6



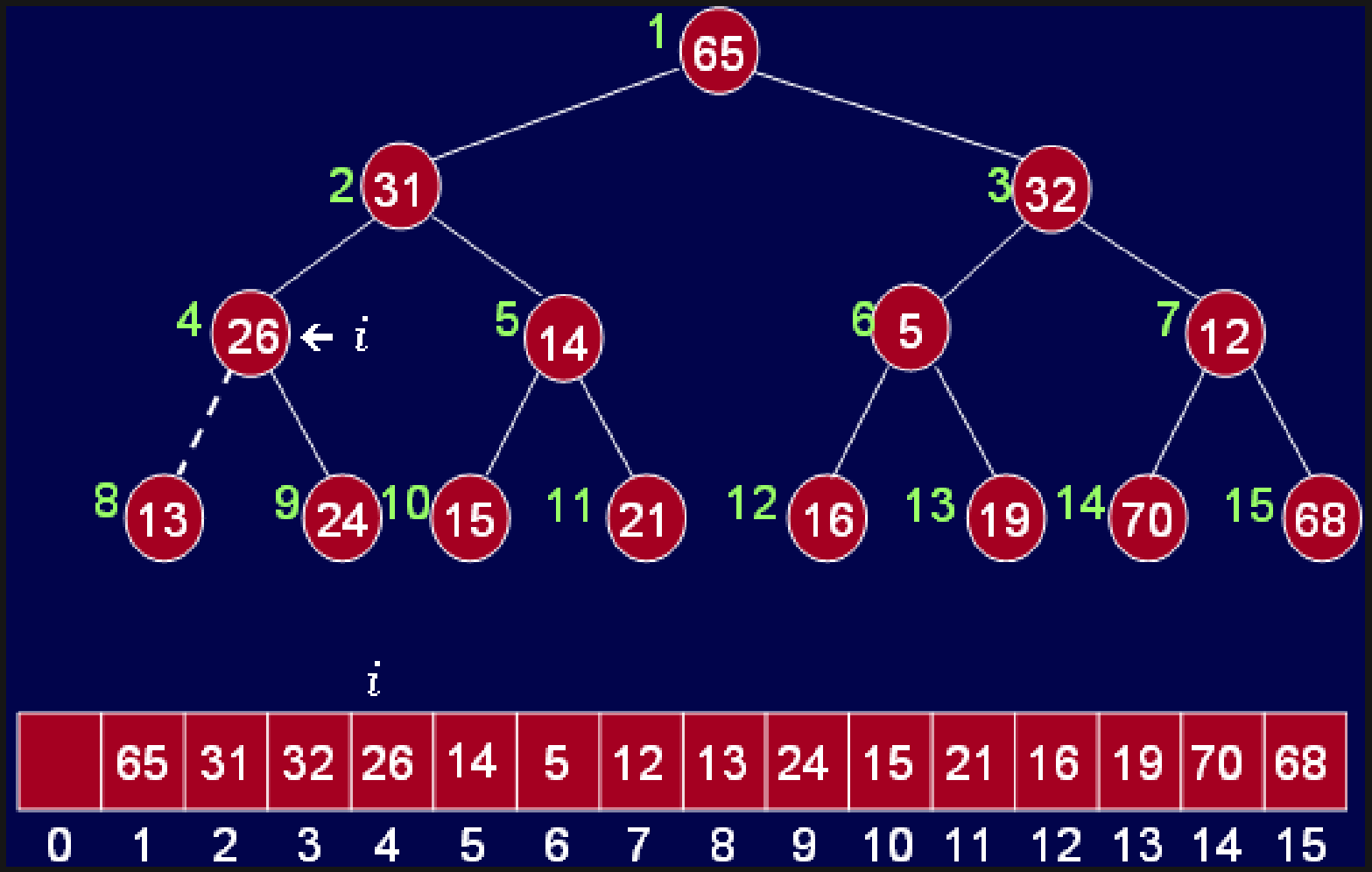
BuildHeap

- i = 5



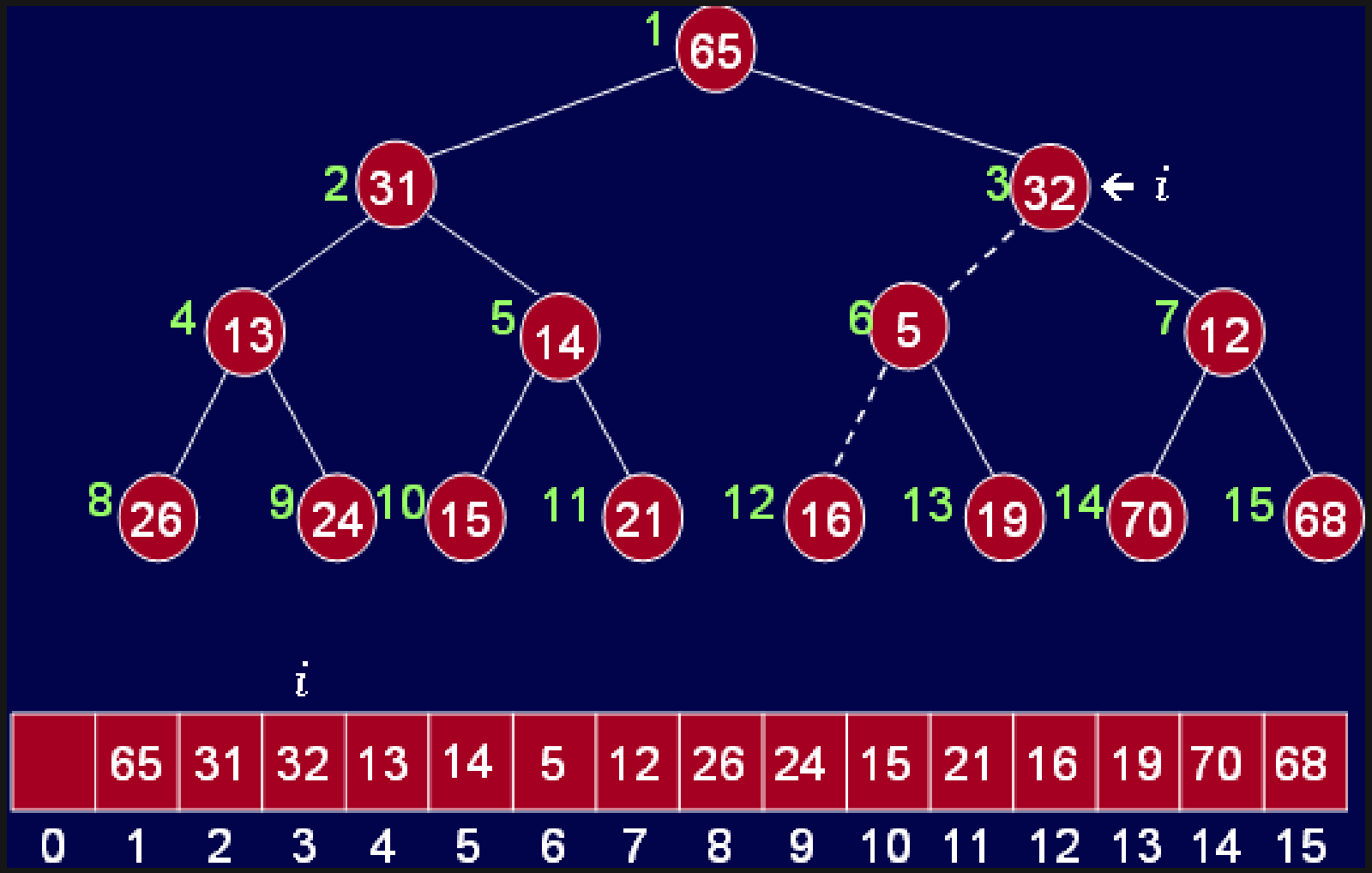
BuildHeap

- i = 4



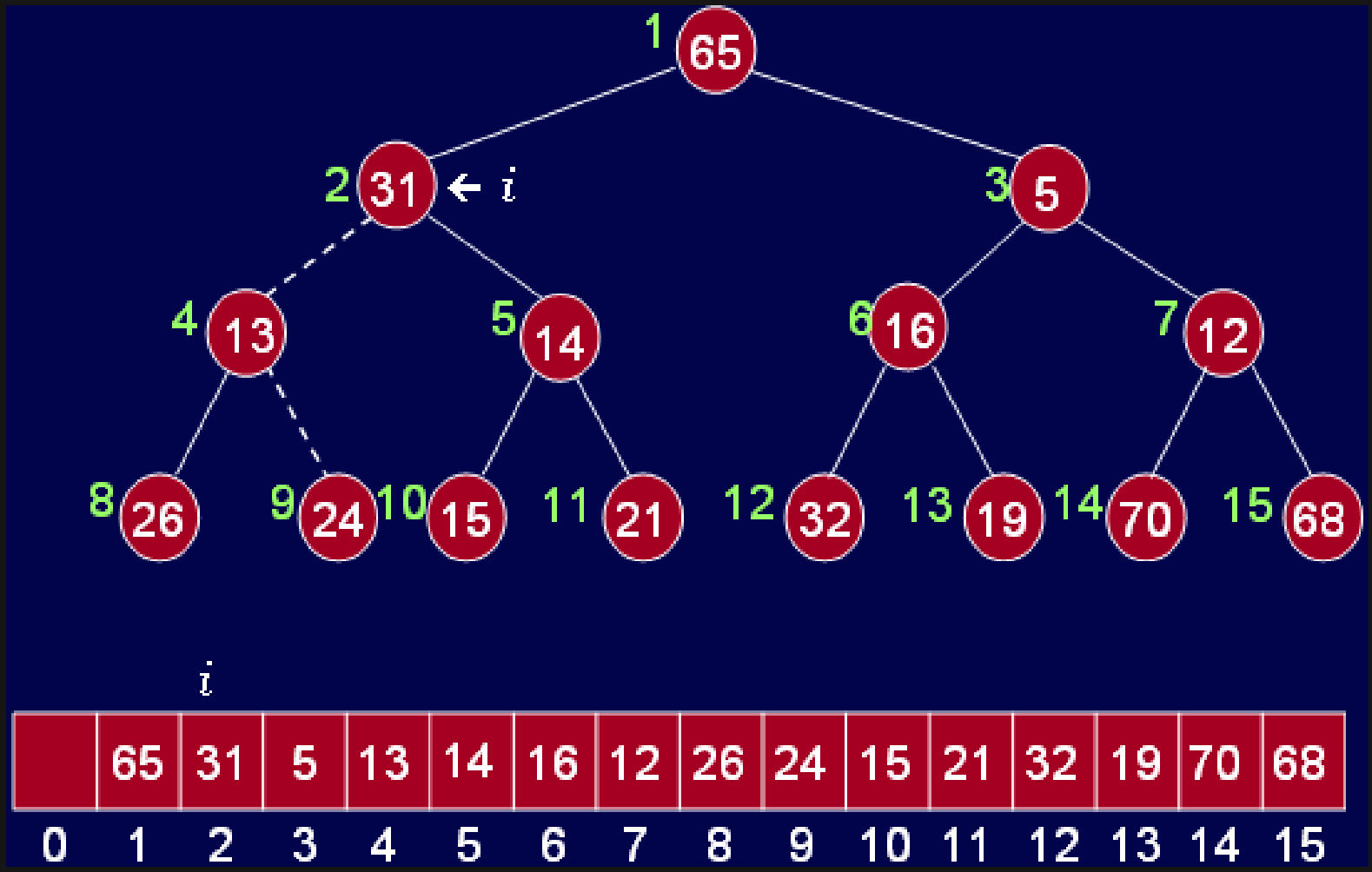
BuildHeap

- i = 3



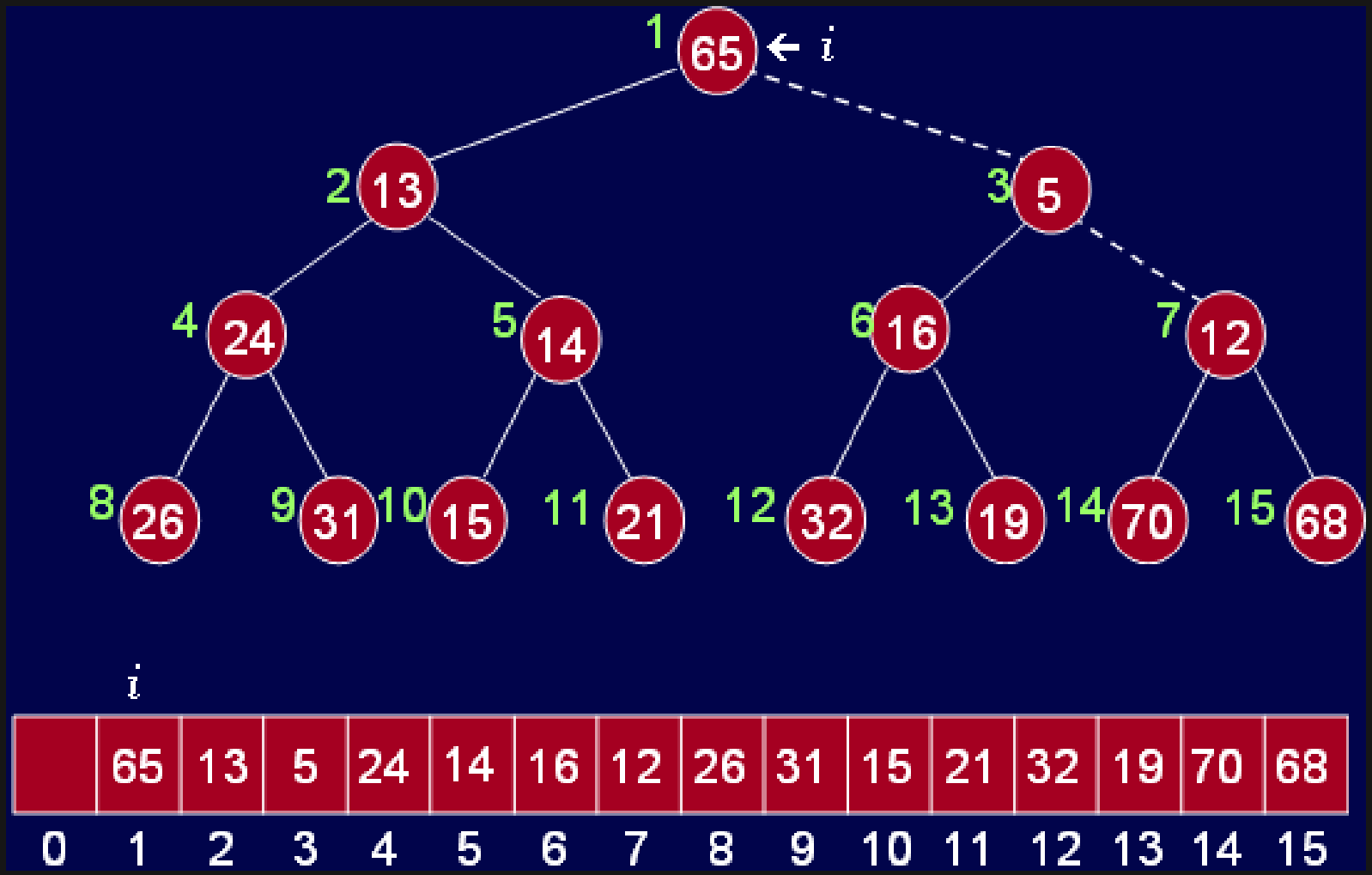
BuildHeap

- i = 2



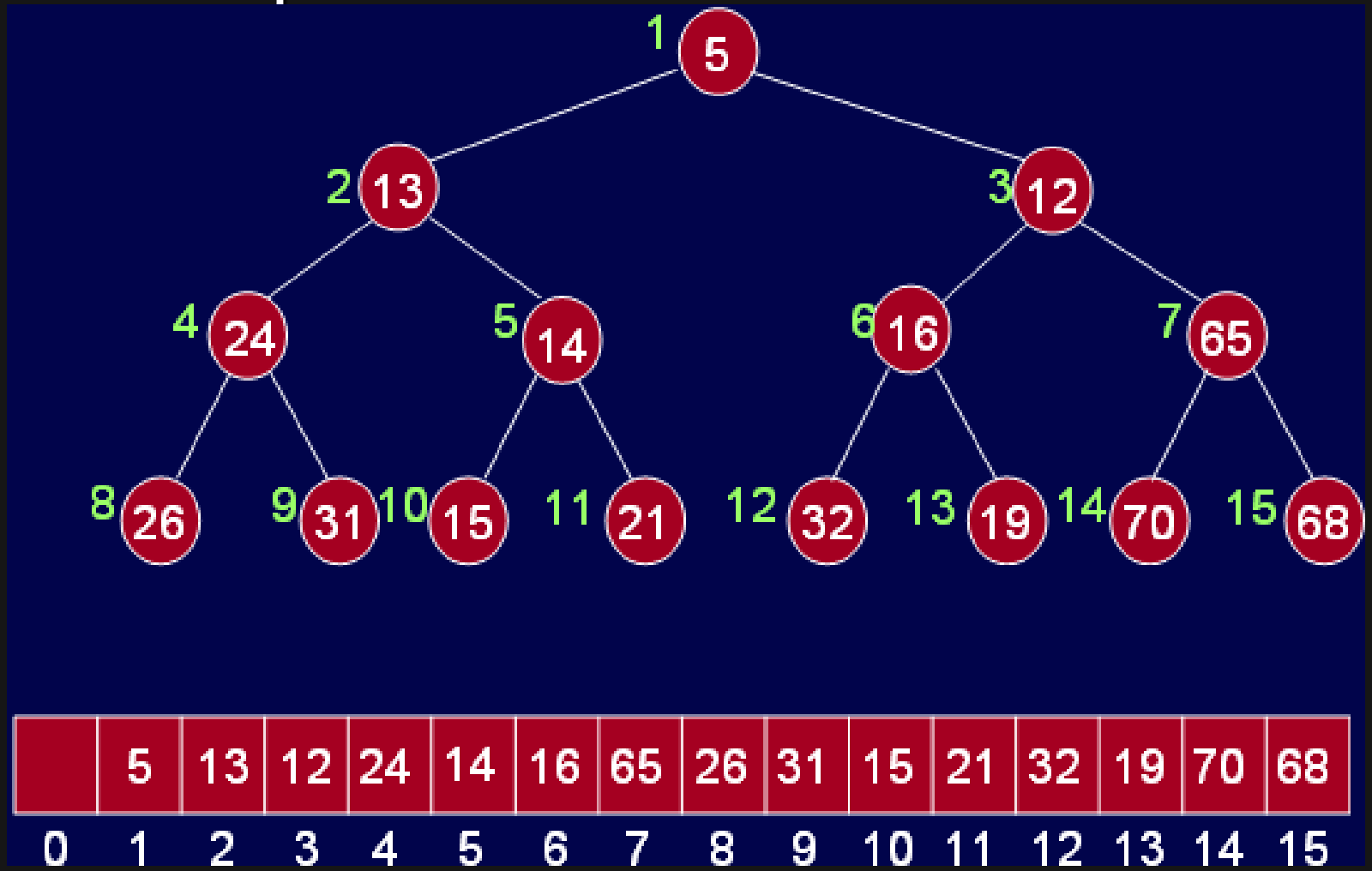
BuildHeap

- i = 1



BuildHeap

- Min heap



Other Heap Operations

- *decreaseKey(p, delta)*:

  lowers the value of the key at position ‘p’ by the amount ‘delta’. Since this might violate the heap order, the heap must be reorganized.
- *increaseKey(p, delta)*:

  opposite of decreaseKey.
- *remove(p)*:

  removes the node at position p from the heap. This is done by first decreaseKey(p, ∞) and then performing deleteMin().

Thank You …