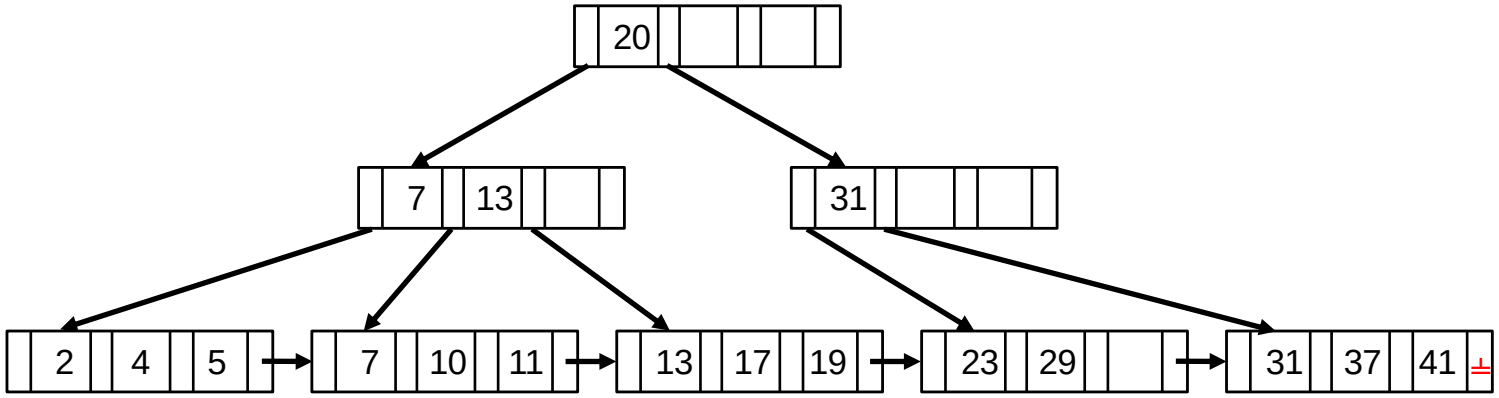
COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 13 EXERCISES
INDEXING: B+-TREE

EXERCISE 1

For the B$^+$-tree below with order 2 and fan out 4, show the tree that would result after *successively* applying the following operations in order.

i. insert 3 ii. insert 8



Non-leaf nodes: min $\lceil 4/2 \rceil$ = 2 pointers; min $\lceil 4/2 \rceil$-1 = 1 value

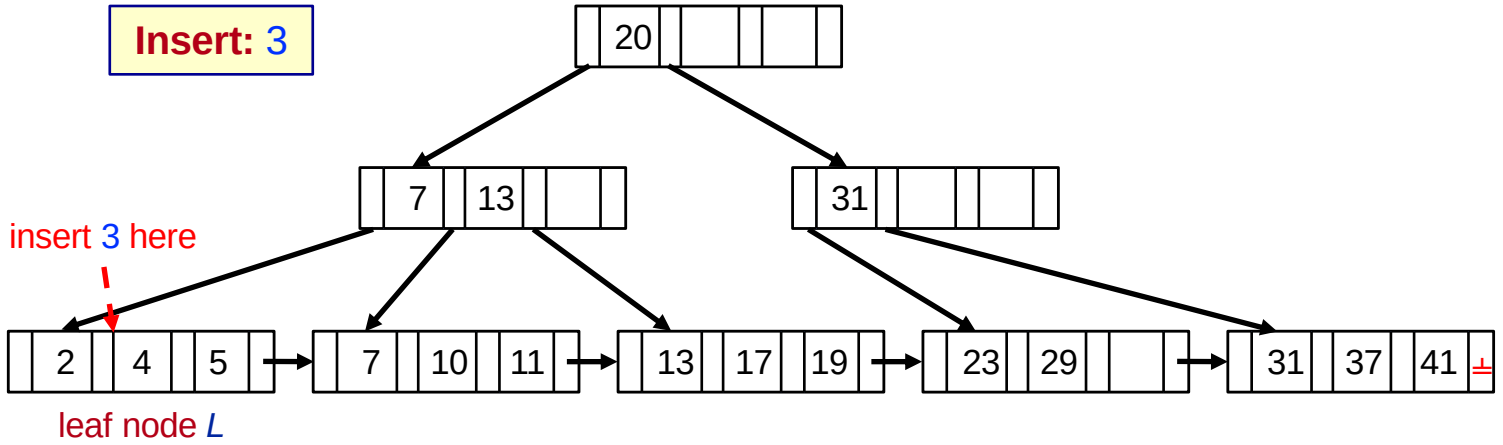
Leaf nodes: min $\lceil (4-1)/2 \rceil$ +1 = 3 pointers; min $\lceil (4-1)/2 \rceil$ = 2 values

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 1 (cont'd)

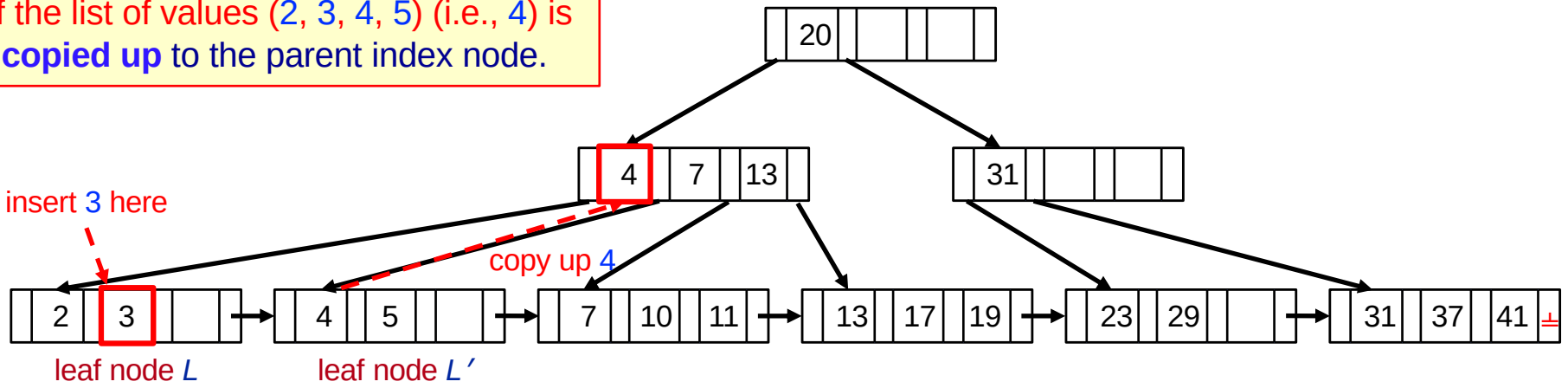
The insertion causes the first leaf node to become overfull.

Insert: 3



B+-tree *before* insertion of 3.

The leaf node is split and the search-key value of the entry in position $\lceil n/2 \rceil + 1 = 3$ of the list of values (2, 3, 4, 5) (i.e., 4) is **copied up** to the parent index node.



B+-tree *after* insertion of 3.

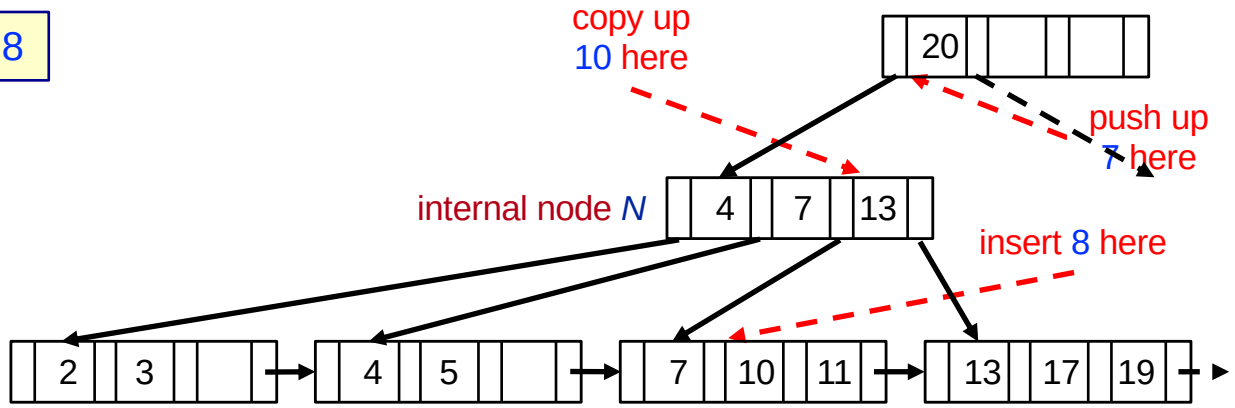
n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 1 (cont'd)

The insertion causes the third leaf node to become overfull.

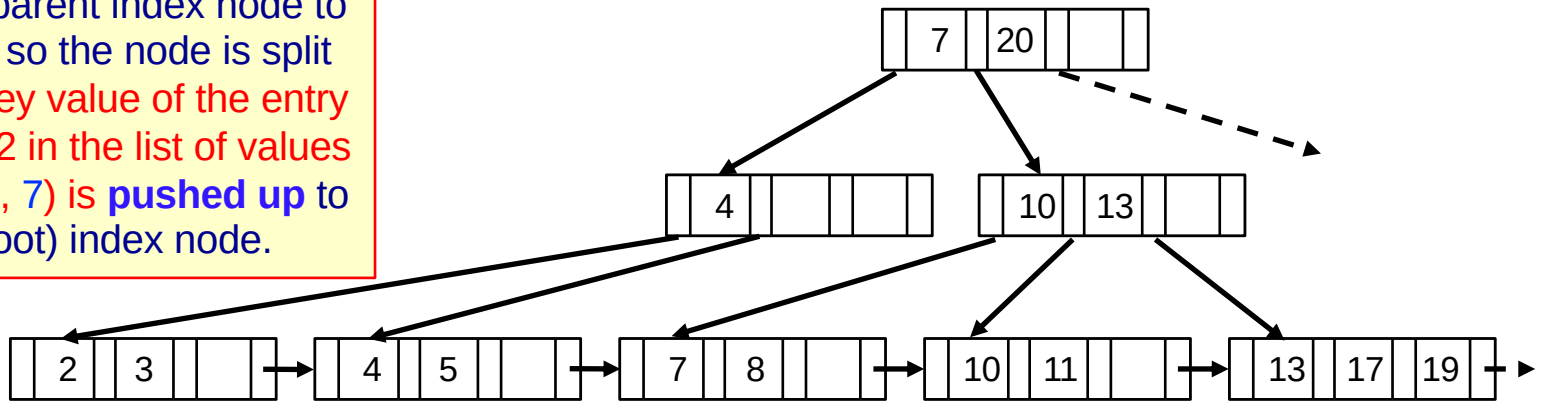
Insert: 8

The leaf node is split and the search-key value of the entry in position $\lceil n/2 \rceil+1=3$ of the list of values (7, 8, 10, 11) (i.e., 10) is **copied up** to the parent index node.



B+-tree *before* insertion of 8.

This causes the parent index node to become overfull so the node is split and the search-key value of the entry in position $\lceil n/2 \rceil=2$ in the list of values (4, 7, 10, 13) (i.e., 7) is **pushed up** to the parent (root) index node.

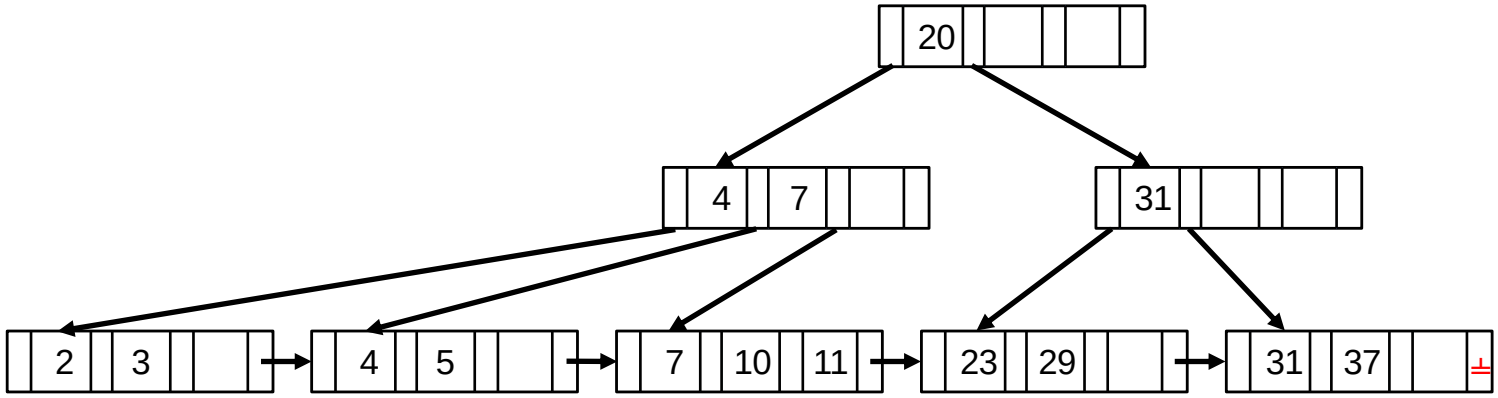


B+-tree *after* insertion of 8.

EXERCISE 2

For the B$^+$-tree below with order 2 and fan out 4, show the tree that would result after *successively* applying the following operations in order.

i. delete 5 ii. delete 3 iii. delete 11



Non-leaf nodes: min $\lceil 4/2 \rceil$ = 2 pointers; min $\lceil 4/2 \rceil$-1 = 1 value

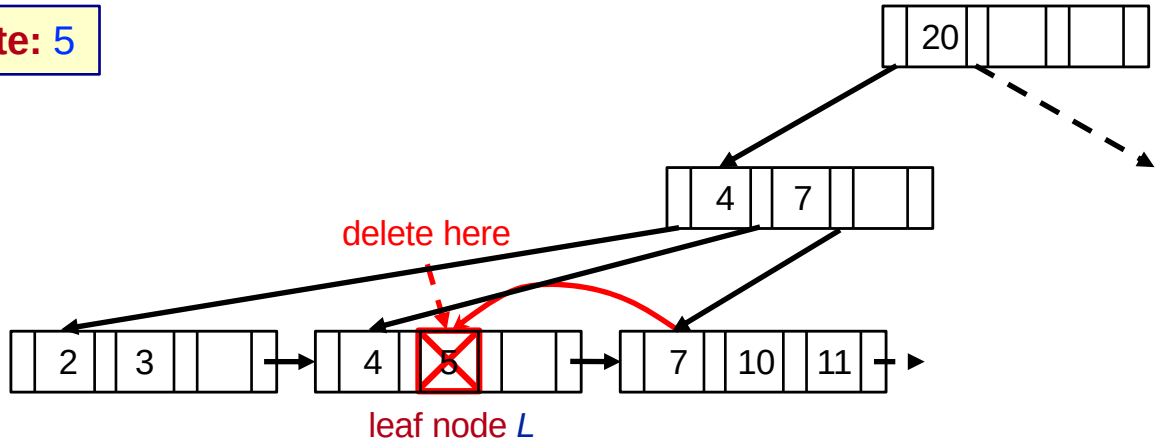
Leaf nodes: min $\lceil (4-1)/2 \rceil$ +1 = 3 pointers; min $\lceil (4-1)/2 \rceil$ = 2 values

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 2 (CONT'D)

The deletion causes the second leaf node to become underfull (less than $\lceil (n\text{-}1)/2 \rceil = 2$ values).

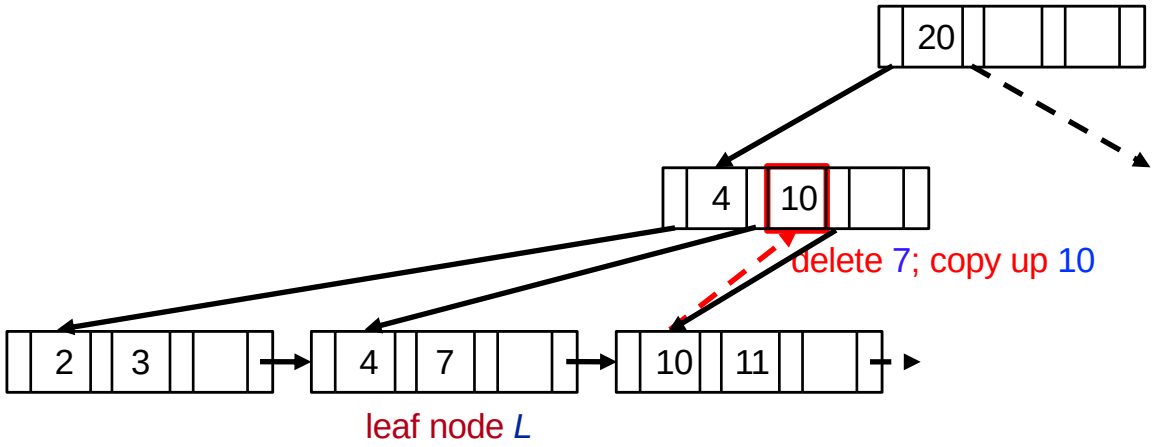
Delete: 5



B^+-tree *before* deletion of 5.

The node can borrow a value (7) from its right sibling.

The parent index node is adjusted accordingly by deleting 7 and copying up 10.



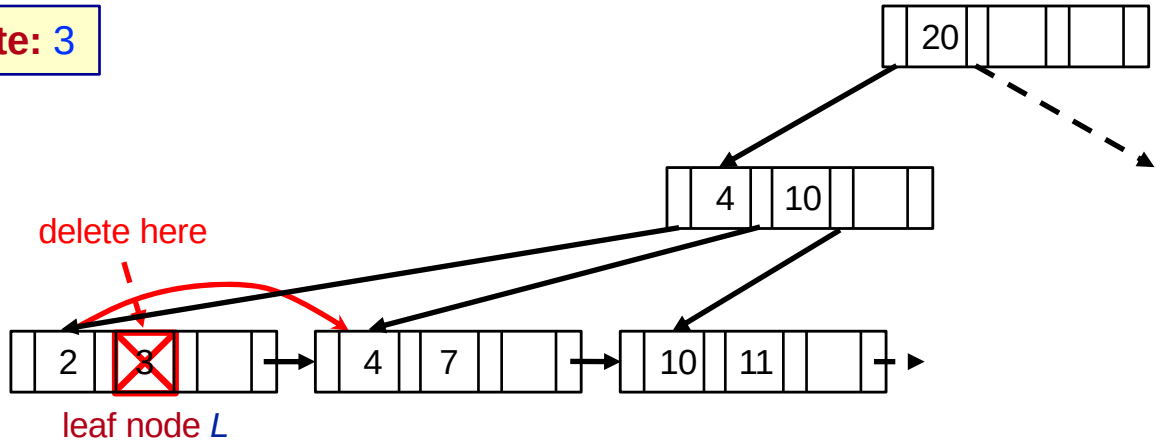
B^+-tree *after* deletion of 5.

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 2 (CONT'D)

The deletion causes the first leaf node to become underfull (less than $\lceil (n\text{-}1)/2 \rceil$ =2 values).

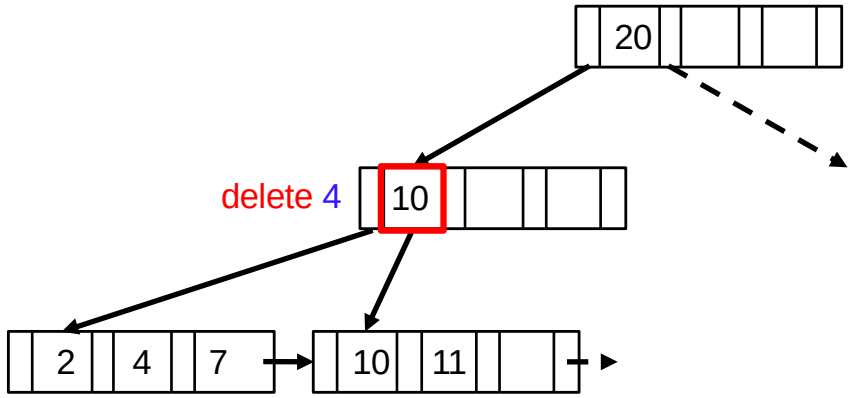
Delete: 3



B+-tree *before* deletion of 3.

The node cannot borrow a value from its right sibling, so it must be merged with it.

The parent index node is adjusted accordingly by deleting 4.



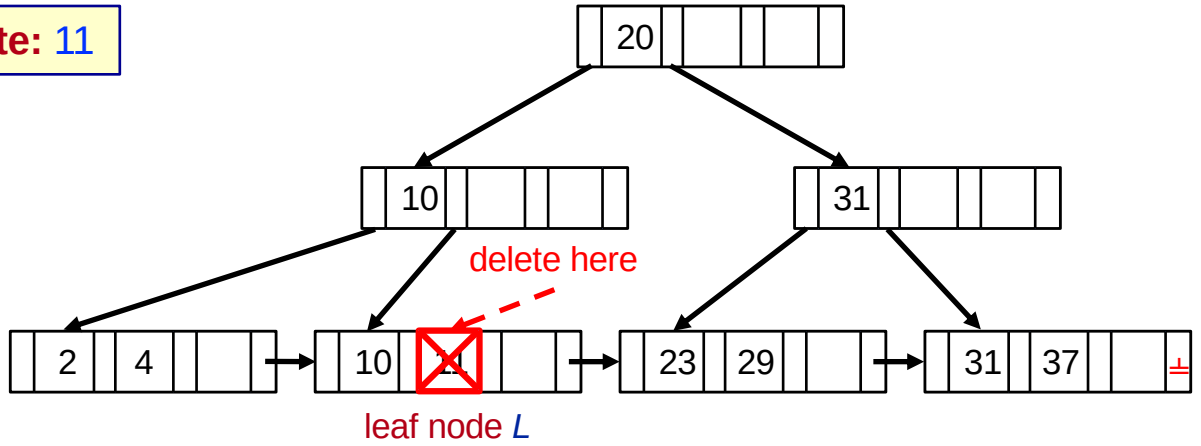
B+-tree *after* deletion of 3.

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 2 (CONT'D)

The deletion causes the second leaf node to become underfull (less than $\lceil (n\text{-}1)/2 \rceil$ =2 values).

Delete: 11

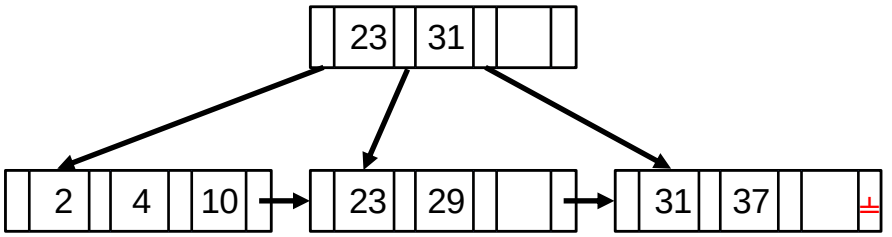


B$^+$-tree *before* deletion of 11.

The node cannot borrow a value from either of its siblings, so it must be merged (pick left sibling).

This causes the parent index node to now have only 1 pointer, but it needs 2. Therefore, it must be merged with its sibling and the index values adjusted.

This merge causes the root index node to now have only 1 pointer, so it can be deleted and the tree shrinks one level.

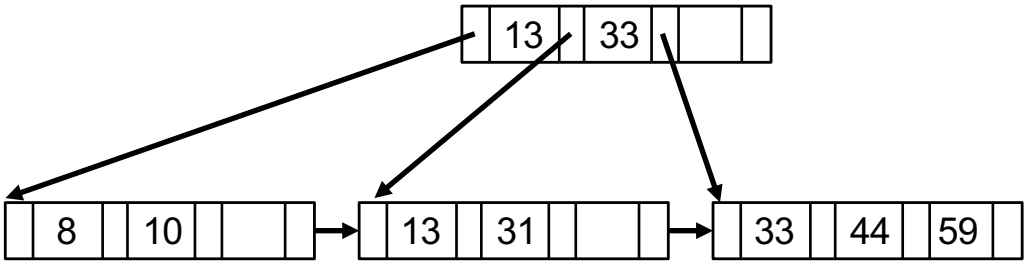


B$^+$-tree *after* deletion of 11.

EXERCISE 3

For the B$^+$-tree below with order 2 and fan out 4, show the tree that would result after *successively* applying the following operations in order. Add nodes to or cross out nodes in the empty B$^+$-tree below as necessary.

i. insert 45 ii. insert 35 iii. insert 40 iv. delete 59

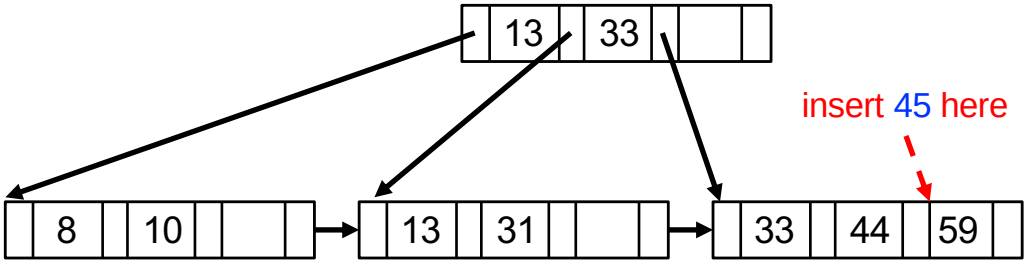


Non-leaf nodes: min $\lceil 4/2 \rceil$ = 2 pointers; min $\lceil 4/2 \rceil$-1 = 1 value

Leaf nodes: min $\lceil (4-1)/2 \rceil$ +1 = 3 pointers; min $\lceil (4-1)/2 \rceil$ = 2 values

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)

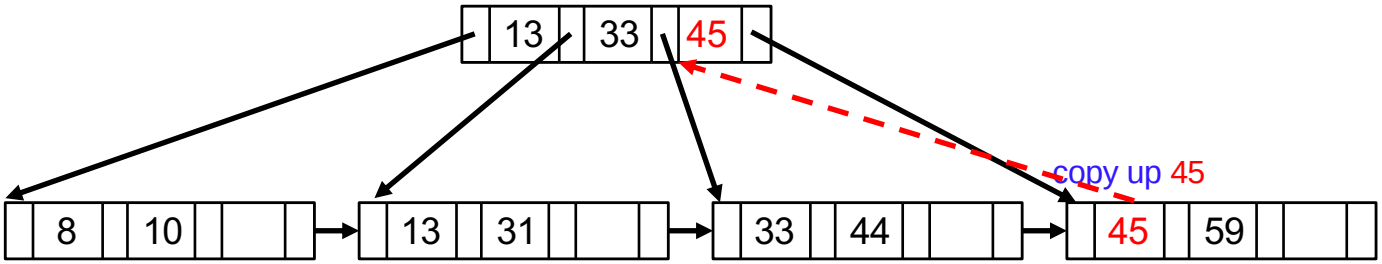


Insert: 45

The value is inserted in the right-most leaf node.

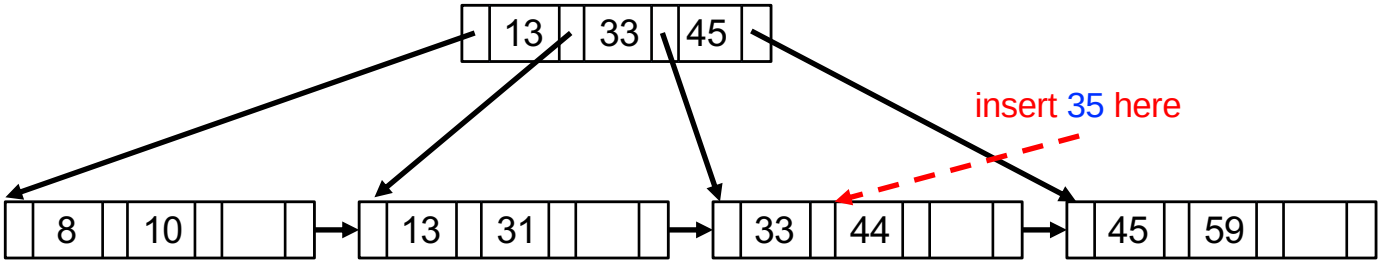
This causes the node to become overfull and split.

The search-key value at position $\lceil n/2 \rceil+1=3$ (i.e., 45) in the list of values (33, 44, 45, 59) is **copied up** to the parent index node.



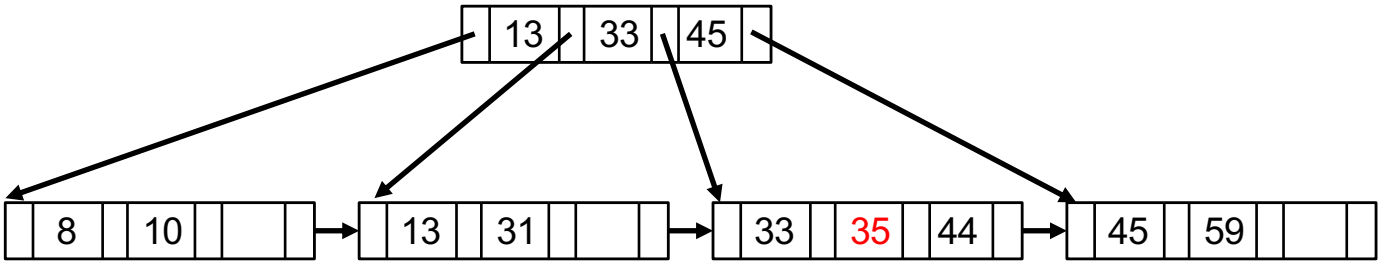
n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)



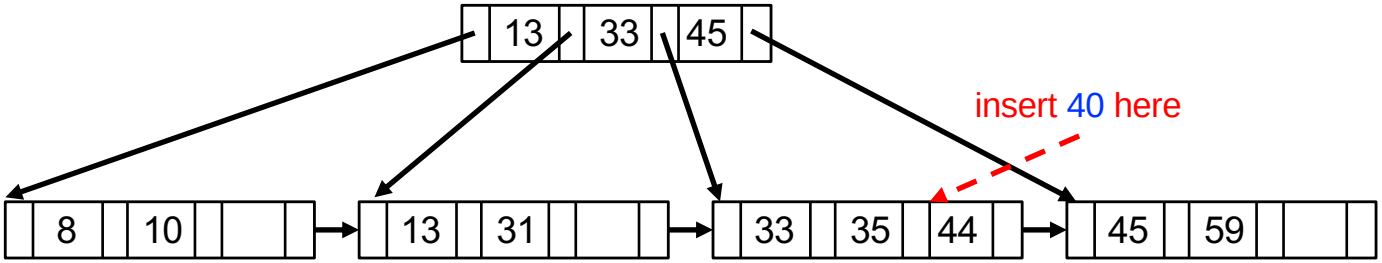
Insert: 35

The value is inserted in the third leaf node from the left in order.



n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)

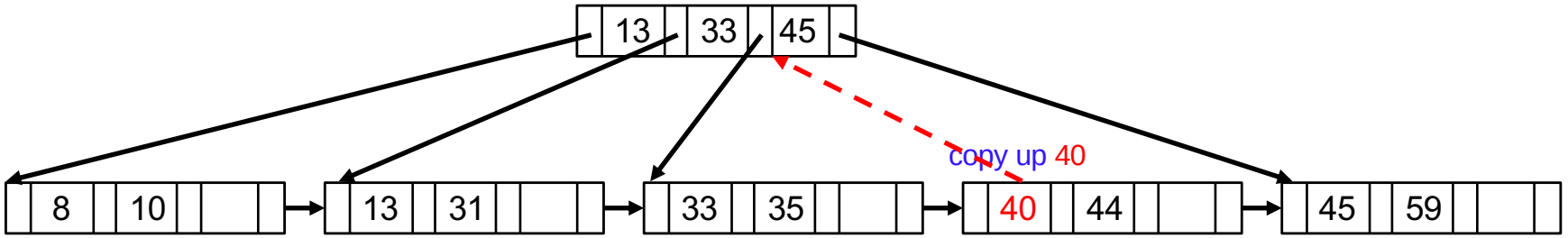


Insert: 40

The value is inserted in the third leaf node from the left.

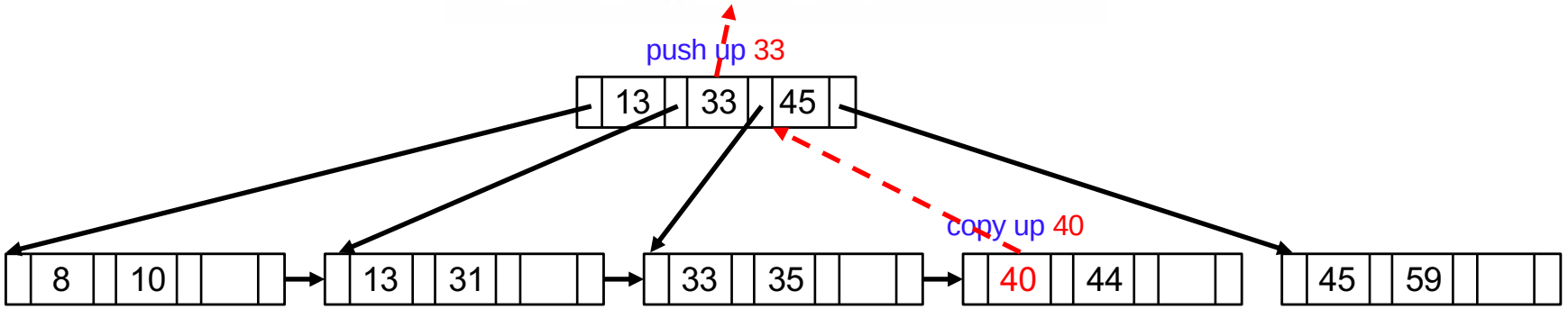
This causes the node to become overfull and split.

The search-key value at position $\lceil n/2 \rceil+1=3$ (i.e., 40) in in the list of values (33, 35, 40, 44) is **copied up** to the parent index node.



n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)

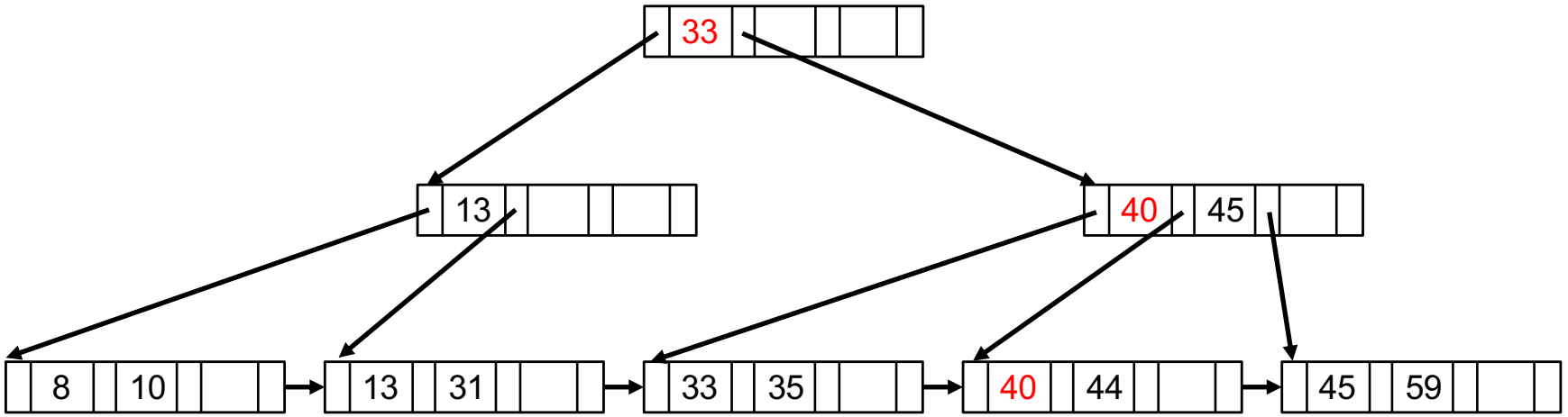


Insert: 40

This causes the parent node to become overfull and split.

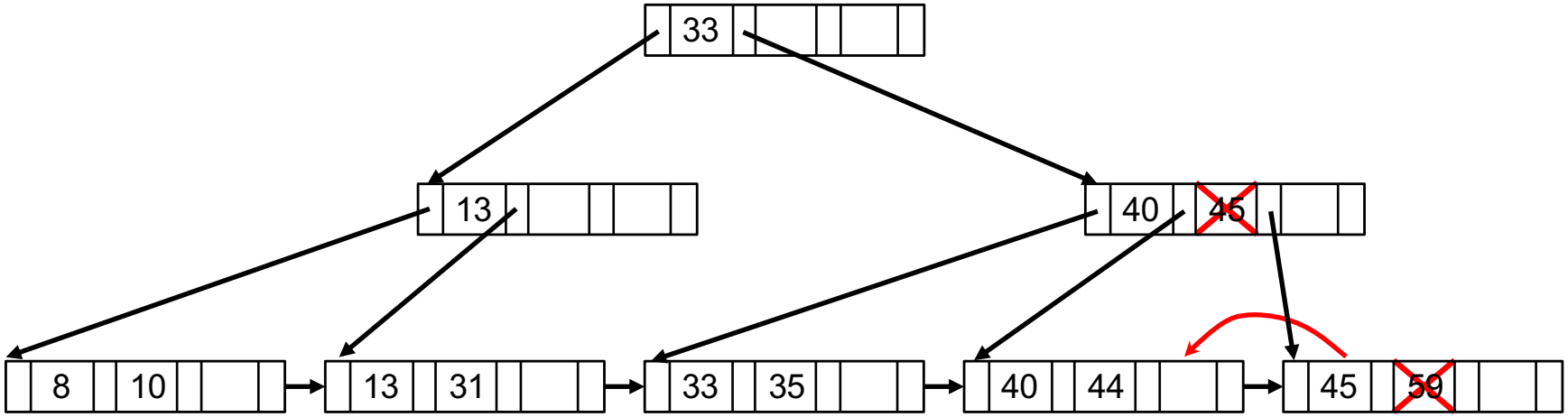
The first $\lceil n/2 \rceil - 1 = 1$ value is placed in the existing node. The search-key value at position $\lceil n/2 \rceil = 2$ (i.e., 33) in the list of values (13, 33, 40, 45) is **pushed up** into the new root node.

The remaining 2 values are placed in a new internal node.



n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)



Delete: 59

Deleting 59 causes the right-most leaf node to become underfull (i.e., it has less than 2 values).

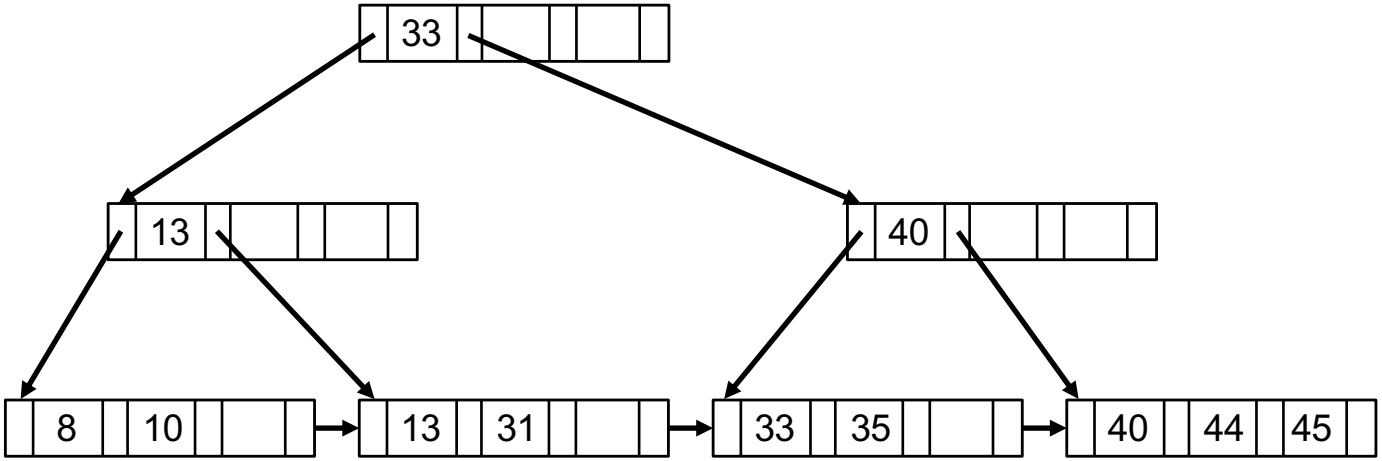
Since a value cannot be borrowed from its right sibling, the left-most leaf node is merged with its right sibling.

The parent node is adjusted by deleting 45.

n = 4
non-leaf nodes: 1 to 3 values
leaf nodes: 2 to 3 values

EXERCISE 3 (CONT'D)

Final B+-tree



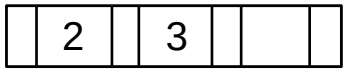
EXERCISE 4

Construct a B^+-tree for the following set of search-key values using bulk loading, which creates leaf nodes from left to right. Assume that each node can hold 4 pointers (i.e., 3 values) and that each leaf node is loaded with the minimum number of values.

2 3 5 7 11 17 19 23 29 31

Since n=4, the minimum number of values in a leaf node is $\lceil n/2 \rceil = 2$.

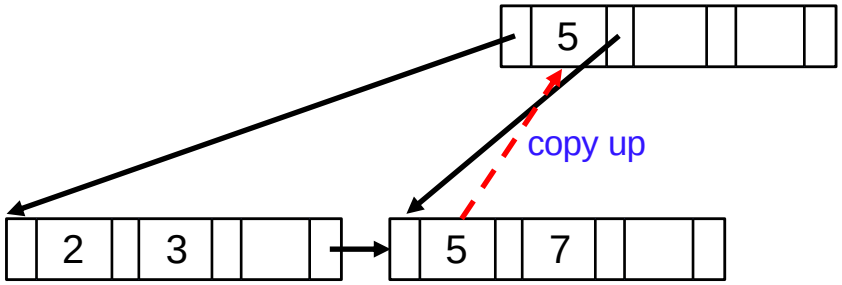
Therefore, load the first two records into the root node.



EXERCISE 4 (cont'd)

~~2~~ ~~3~~ 5 7 11 17 19 23 29 31

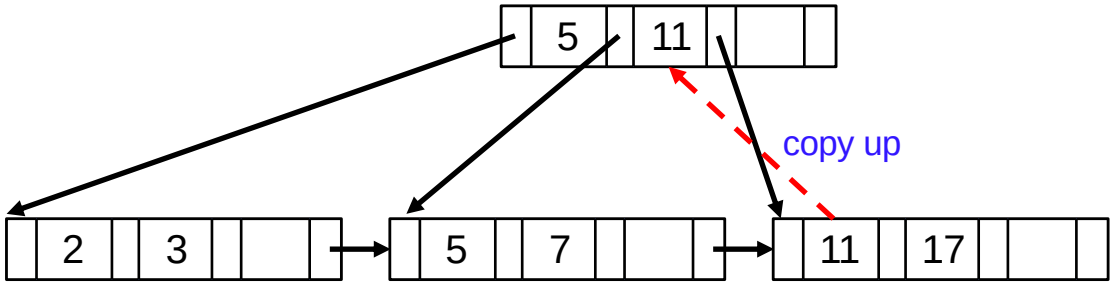
Load the next two records and create a new root node.



EXERCISE 4 (cont'd)

~~2 3 5 7~~ 11 17 19 23 29 31

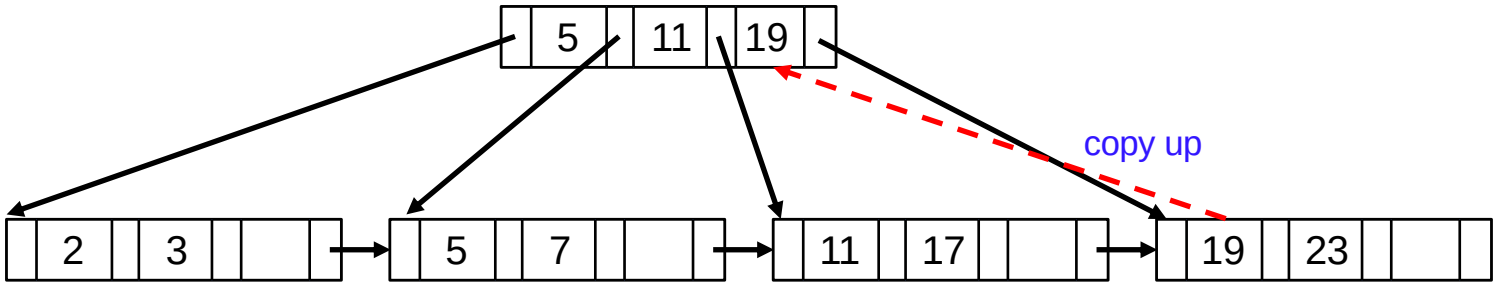
Load the next two records.



EXERCISE 4 (cont'd)

~~2 3 5 7 11 17~~ 19 23 29 31

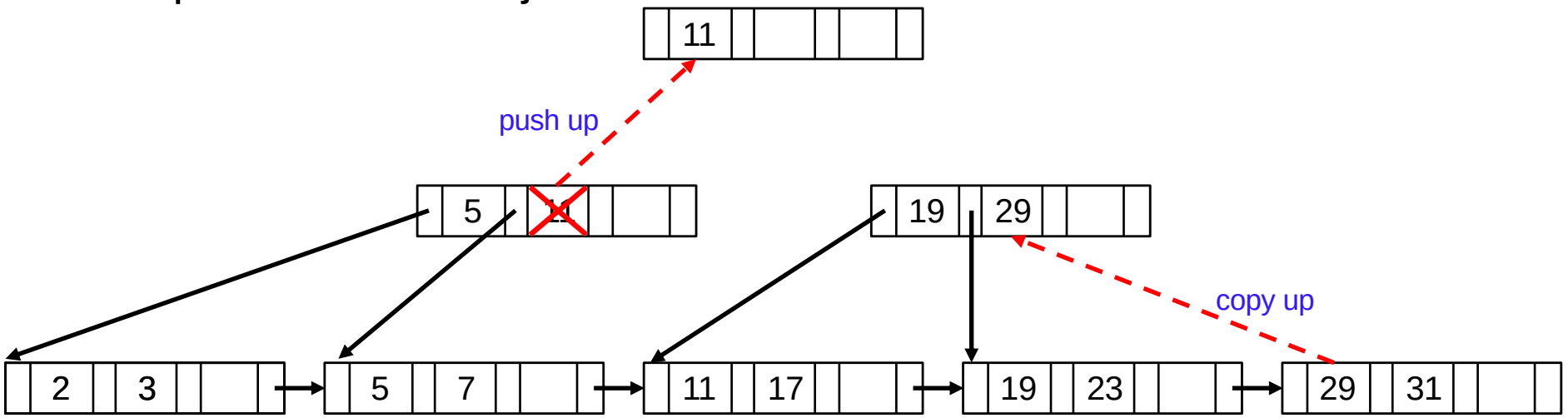
Load the next two records.



EXERCISE 4 (cont'd)

~~2 3 5 7 11 17 19 23~~ 29 31

Load the final two records. This requires the root node to be split, the value at position $\lceil n/2 \rceil = 2$ to be pushed up into a new root node and pointers to be adjusted.



EXERCISE 4 (cont'd)

~~2 3 5 7 11 17 19 23 29 31~~

Done.