COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 13 EXERCISES
INDEXING:
HASH INDEX & BITMAP INDEX

EXERCISE 1

A company database has the following file and sizes of each field

Employee(employeeId: 6 bytes, employeeName: 10 bytes, departmentId: 4 bytes)

where departmentId is the id of the department where the employee works.

There are 100,000 employee records and 1,000 departments (each department has 100 employees).

A page is 1,000 bytes and a pointer is 4 bytes.

Assume that the file is sorted on departmentId and there is no index.

We want to build a hash index on employeeId on the above file where each entry has the form <employeeId, *pointer*>.

EXERCISE I (CONT'D)

Employee records: 100,000
Departments: 1,000
Page size: 1000 bytes
Record size: 20 bytes
Index entry size: 10 bytes

a) How many index entries are needed?

Index entries needed: 100,000 **Why?**

Explanation: Since a hash index is always secondary, it must be dense.

b) How many pages are required for these index entries (assuming full pages)?

Index entries needed: 100,000

Explanation: It is a hash index so it must be dense.

***bf*$_{employeeIdIndex}$:** ⌊1000 / 10 bytes per entry⌋ = 100 entries per page

Pages needed: ⌈100,000 index entries / 100 index entries per page⌉ = 1000

Employee records: 100,000
Departments: 1,000
Page size: 1000 bytes
Record size: 20 bytes
Index entry size: 10 bytes

EXERCISE I (CONT'D)

c) Using the hash index, what is the page I/O cost of retrieving the record of an employee with a given employeeId, assuming that there are no overflow pages?

Page I/O cost: 2 page I/Os **Why?**

Explanation: 1 access to the hash index + 1 access to retrieve the record.

EXERCISE 2

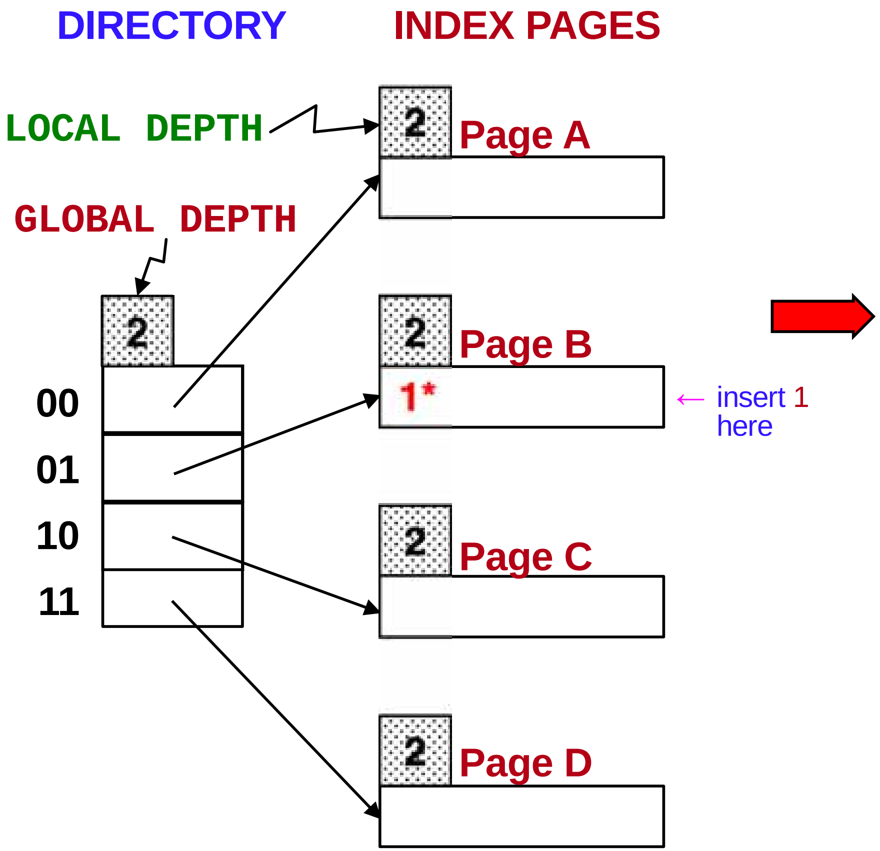
Using the template below, construct a file that contains records with the given search-key values using extendable hashing. Use the hash function $h(x) = x \bmod 8$ and insert records one at a time in order into an empty file. Assume data pages can hold <u>4 records</u>.

key value	1	4	5	7	10	12	15	16	20	24
h(*x*)	1	4	5	7	2	4	7	0	4	0
binary value	001	100	101	111	010	100	111	000	100	000

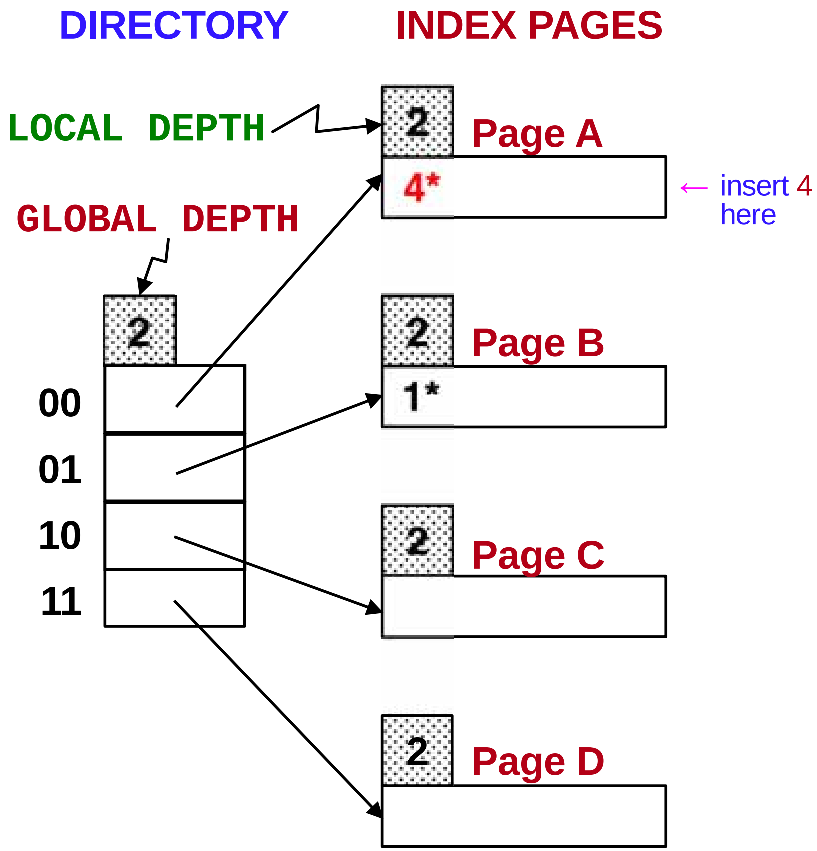
page size: 4

EXERCISE 2 (CONT'D)

Insert: 1 (01)



Insert: 4 (00)

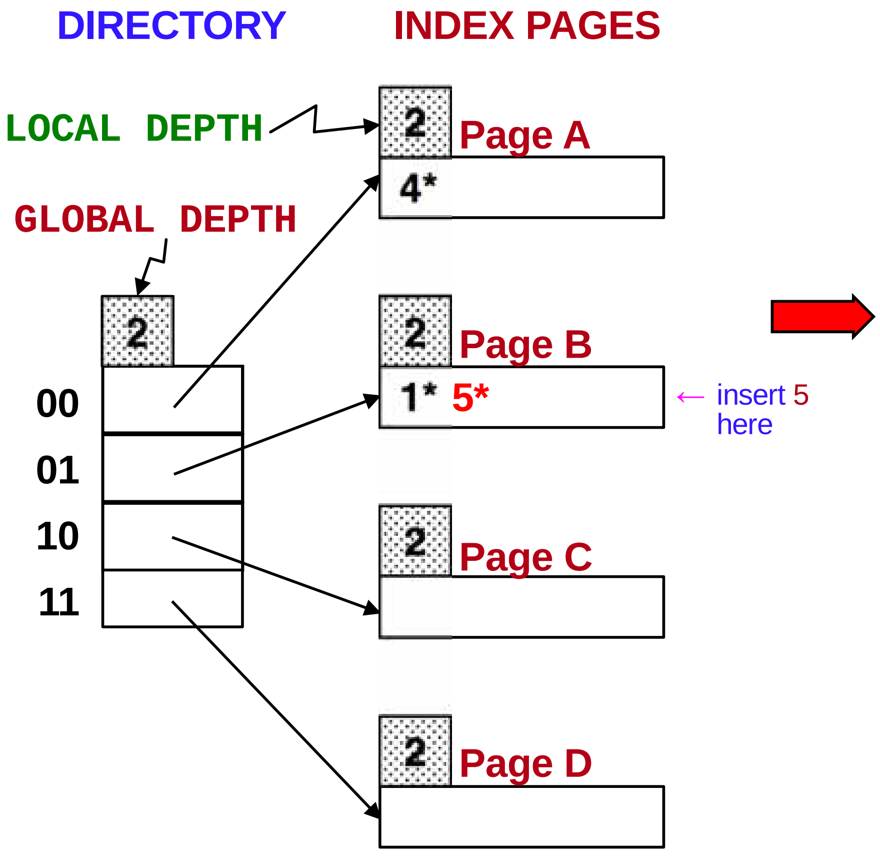


key value	1	4	5	7	10	12	15	16	20	24
h(x)	1	4	5	7	2	4	7	0	4	0
binary value	001	100	101	111	010	100	111	000	100	000

page size: 4

EXERCISE 2 (CONT'D)

Insert: 5 (01)



DIRECTORY

INDEX PAGES

LOCAL DEPTH

GLOBAL DEPTH

2

00

01

10

11

2 Page A

4*

2 Page B

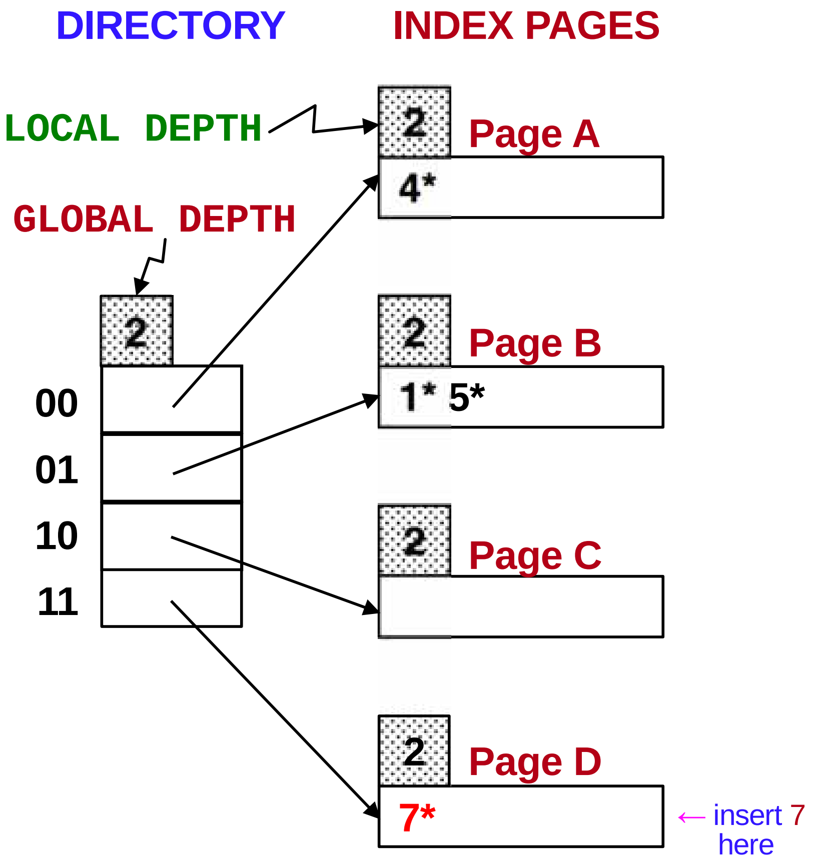
1* 5*

← insert 5 here

2 Page C

2 Page D

Insert: 7 (11)



DIRECTORY

INDEX PAGES

LOCAL DEPTH

GLOBAL DEPTH

2

00

01

10

11

2 Page A

4*

2 Page B

1* 5*

2 Page C

2 Page D

7*

← insert 7 here

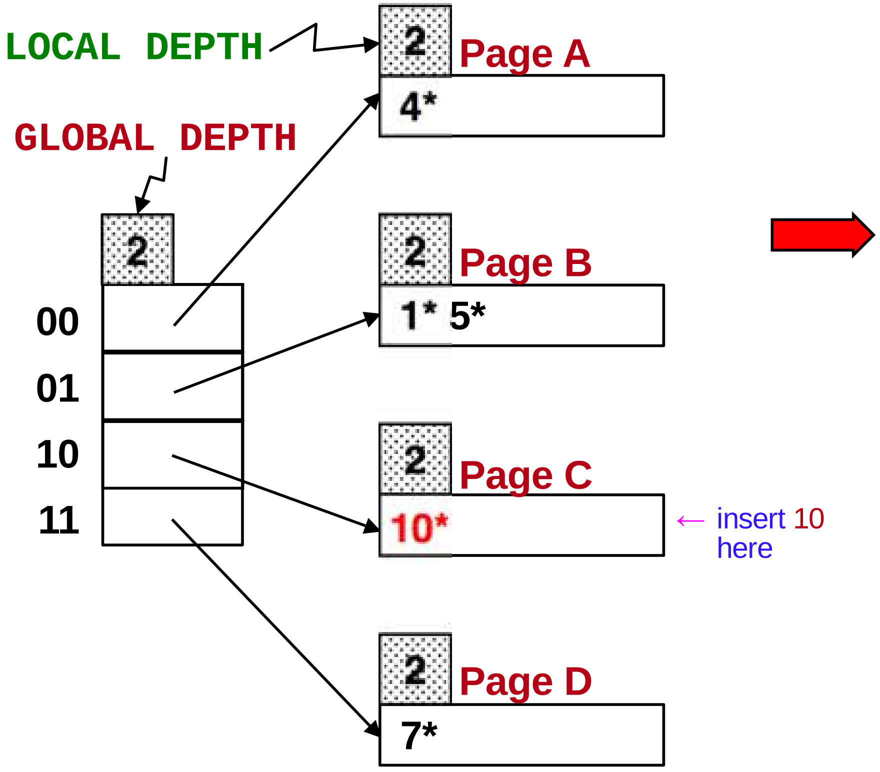
key value	1	4	5	7	10	12	15	16	20	24
h(x)	1	4	5	7	2	4	7	0	4	0
binary value	001	100	101	111	010	100	111	000	100	000

page size: 4

EXERCISE 2 (CONT'D)

Insert: 10 (10)

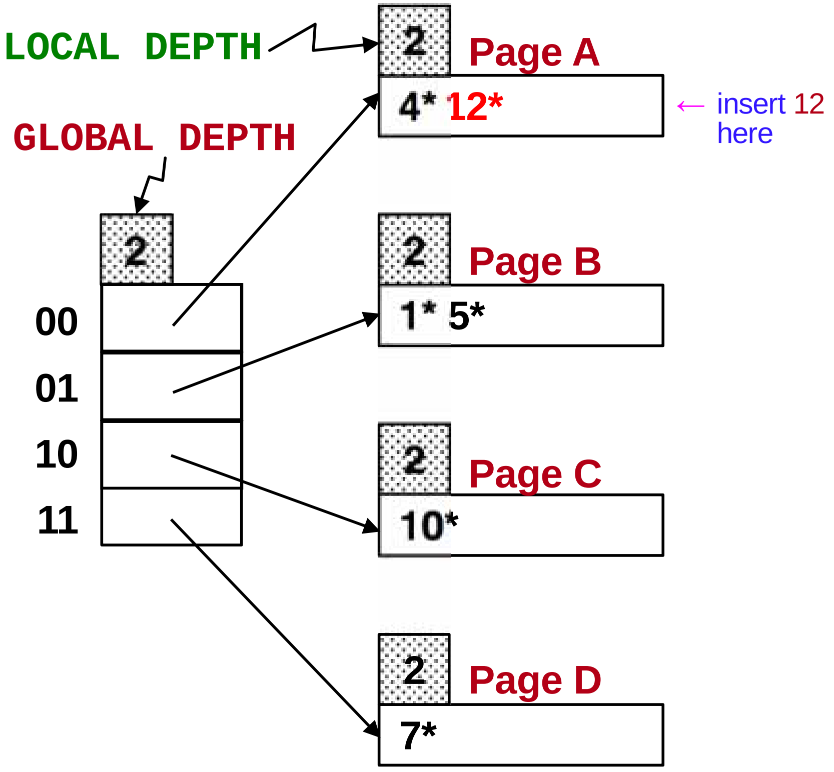
DIRECTORY INDEX PAGES



← insert 10 here

Insert: 12 (00)

DIRECTORY INDEX PAGES



← insert 12 here

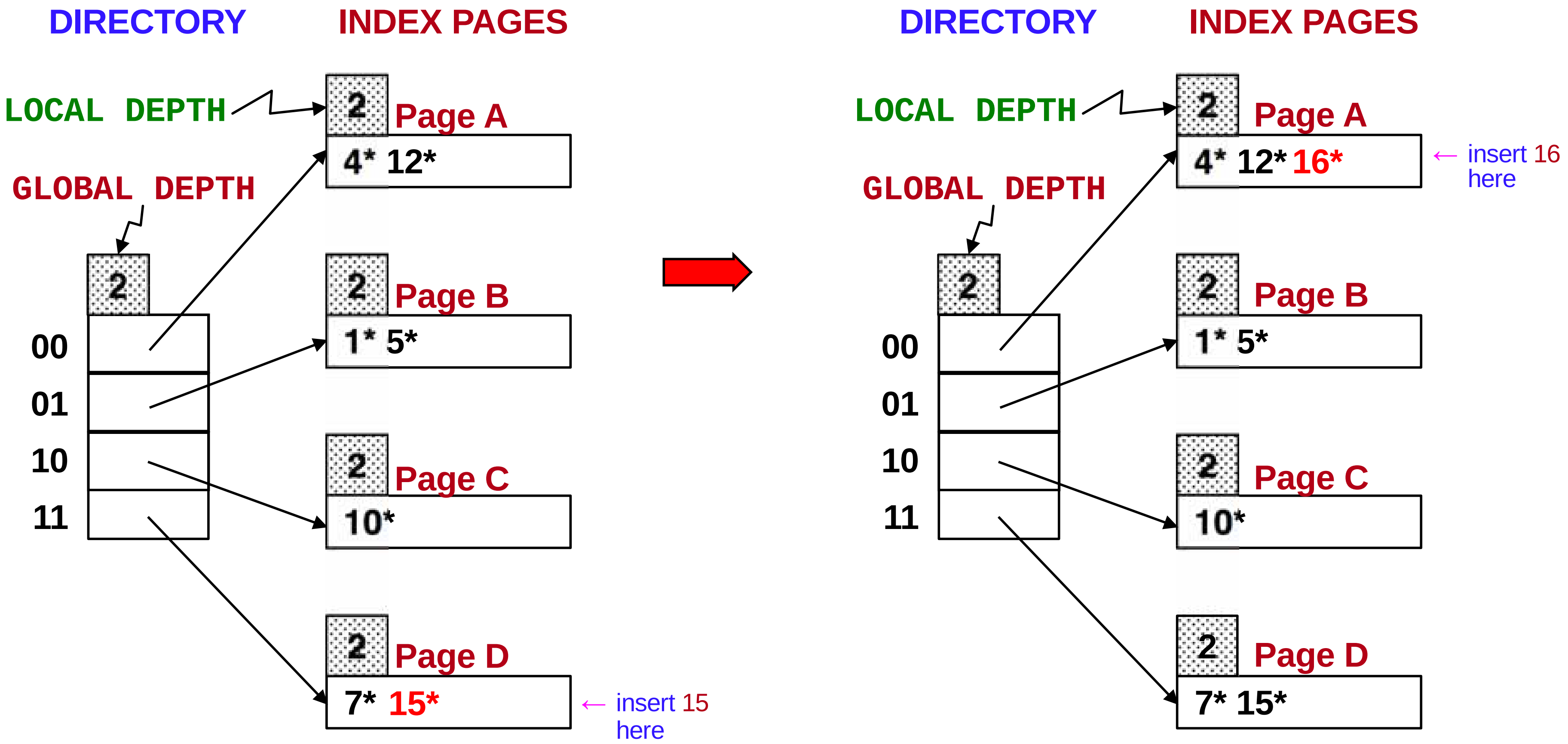
key value	1	4	5	7	10	12	15	16	20	24
h(x)	1	4	5	7	2	4	7	0	4	0
binary value	001	100	101	111	010	100	111	000	100	000

page size: 4

EXERCISE 2 (CONT'D)

Insert: 15 (11)

Insert: 16 (00)



DIRECTORY

INDEX PAGES

LOCAL DEPTH

GLOBAL DEPTH

2

00

01

10

11

2 Page A

4* 12*

2 Page B

1* 5*

2 Page C

10*

2 Page D

7* 15*

← insert 15 here

DIRECTORY

INDEX PAGES

LOCAL DEPTH

GLOBAL DEPTH

2

00

01

10

11

2 Page A

4* 12* 16*

← insert 16 here

2 Page B

1* 5*

2 Page C

10*

2 Page D

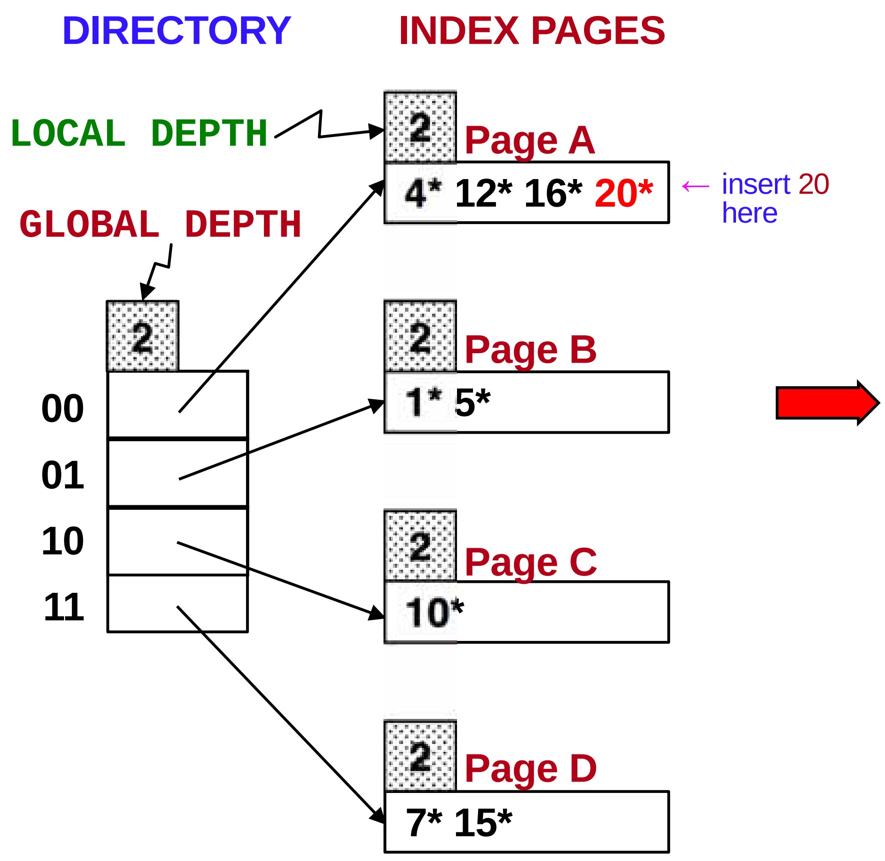
7* 15*

key value	1	4	5	7	10	12	15	16	20	24
h(x)	1	4	5	7	2	4	7	0	4	0
binary value	001	100	101	111	010	100	111	000	100	000

page size: 4

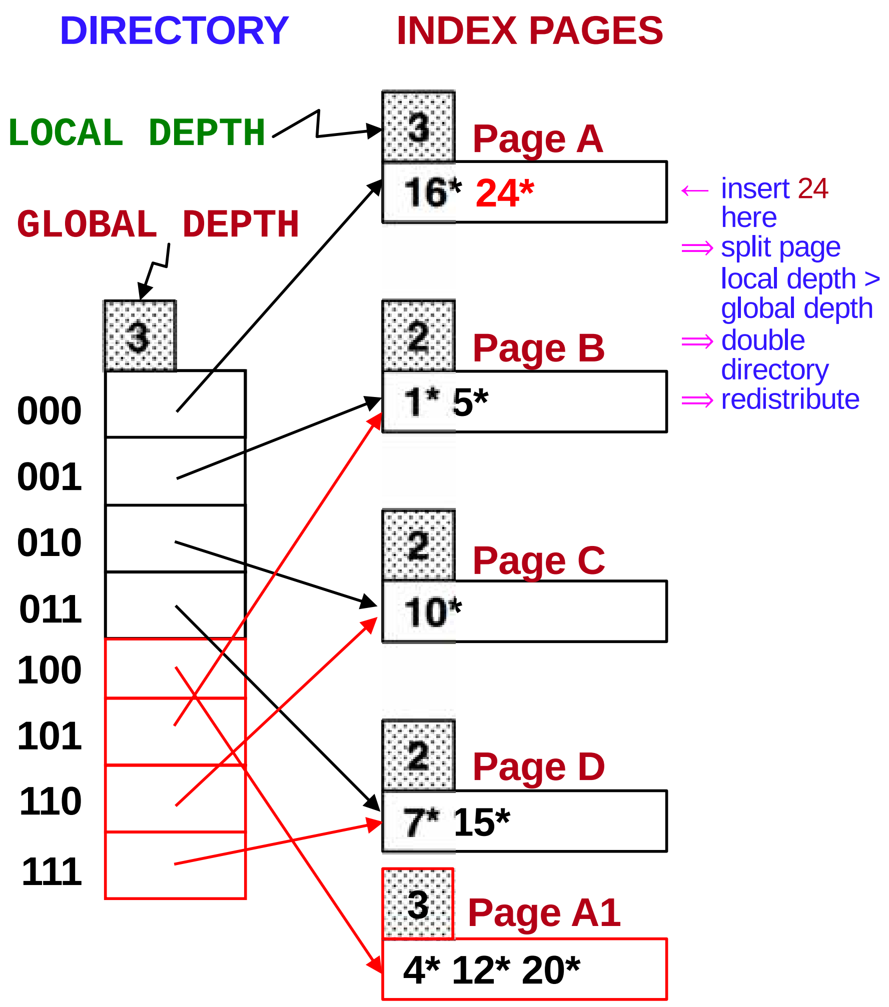
EXERCISE 2 (CONT'D)

Insert: 20 (00)



← insert 20 here

Insert: 24 (000)



← insert 24 here
⇒ split page local depth > global depth
⇒ double directory
⇒ redistribute

key value	1	4	5	7	10	12	15	16	20	24
h(x)	1	**4**	5	7	2	**4**	7	**0**	**4**	**0**
binary value	001	**100**	101	111	010	**100**	111	**000**	**100**	**000**

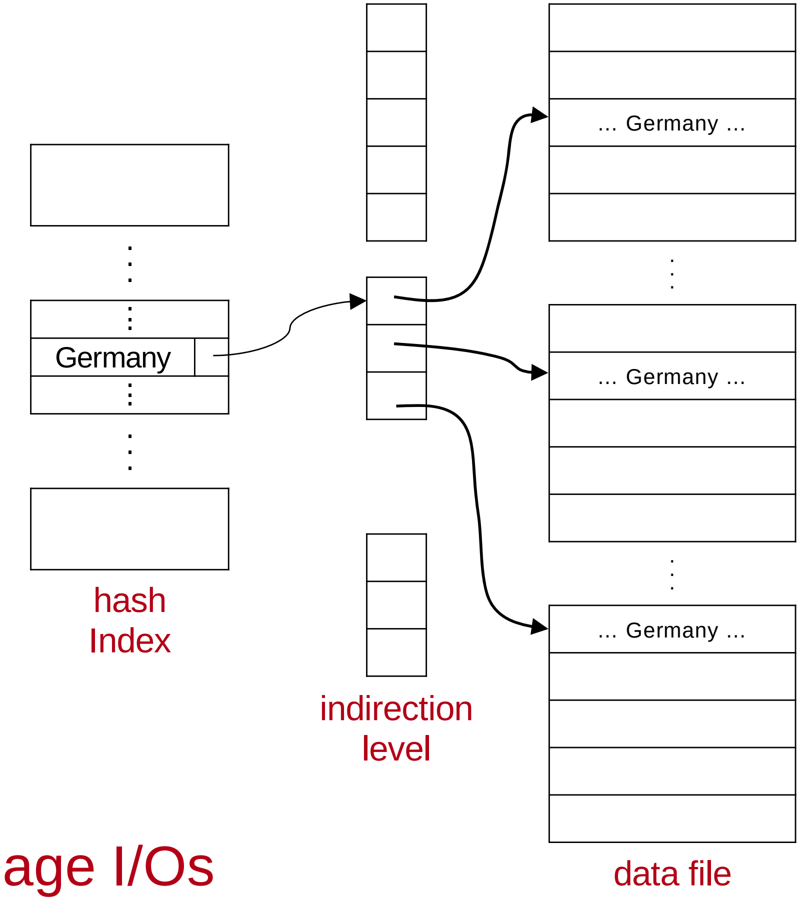
EXERCISE 3

A global e-commerce website maintains a Customer file with attributes customerId, name, address, email and country. The file is organized as a hash file on customerId. There is also a secondary hash index on country. For each country there is only one index entry in the hash index. Assume that on average there are 9,000 customers for each country, there are 90 countries and that a page can hold 100 record pointers. How many page I/Os are required to retrieve the records of all the customers for a given country using the hash index on country?

# records/country: 9,000
# countries: 90
bf$_{\text{record pointers}}$: 100

EXERCISE 3 (CONT'D)

From the problem description, the secondary hash index uses a level of indirection to store all the record pointers to the records for a given country as shown in the figure.



Consequently, the number of page I/Os required to retrieve the records for all the customers in each county is:

Hash index: 1 page I/O

Indirection level: $\lceil 9000 / 100 \rceil$ = 90 page I/Os

Record retrieval: 9000 page I/Os (one for each indirection pointer)

Total page I/Os: 1 + 90 + 9000 = 9091

EXERCISE 4

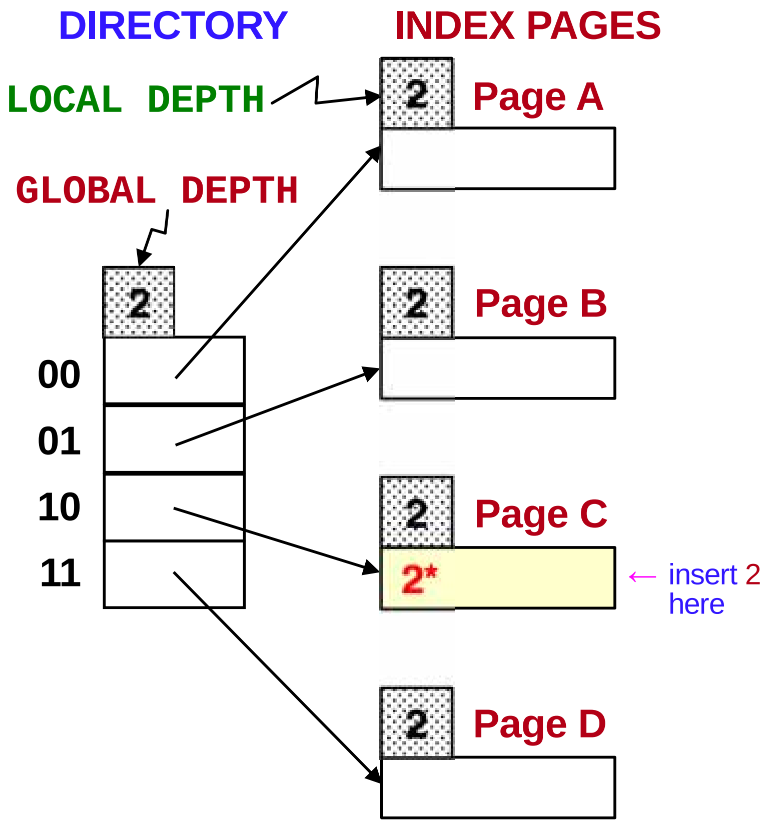
Using the template below, construct a file that contains records with the given search-key values using extendable hashing. Use the hash function *h(x)* = *x* mod 8 and insert records one at a time in order into an empty file. Assume data pages can hold 3 records.

key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
binary value	010	011	101	111	011	001	011	111	101	111

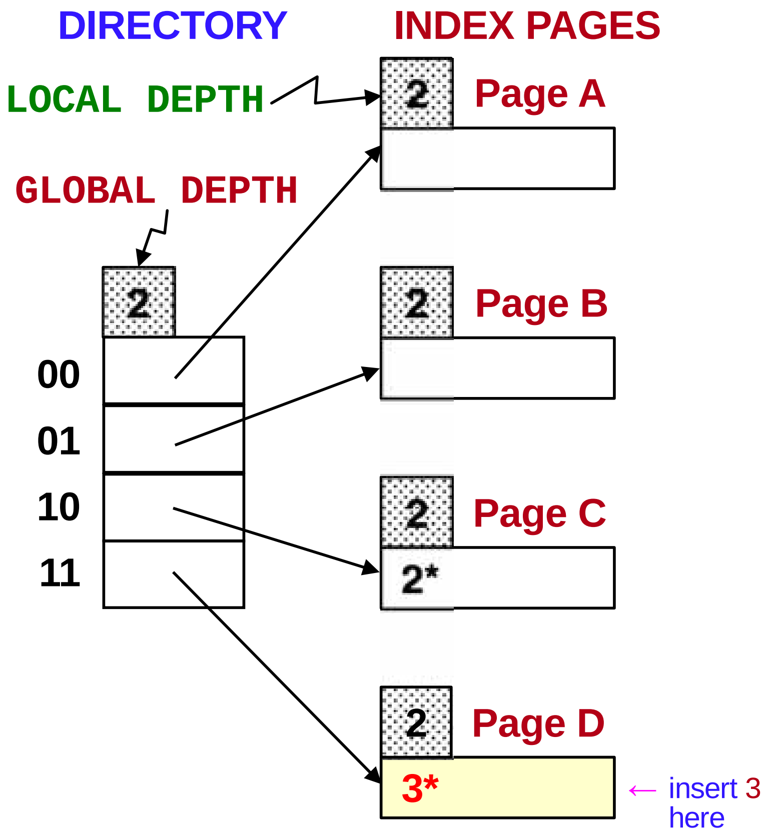
page size: 3

EXERCISE 4 (CONT'D)

Insert: 2 (10)



Insert: 3 (11)

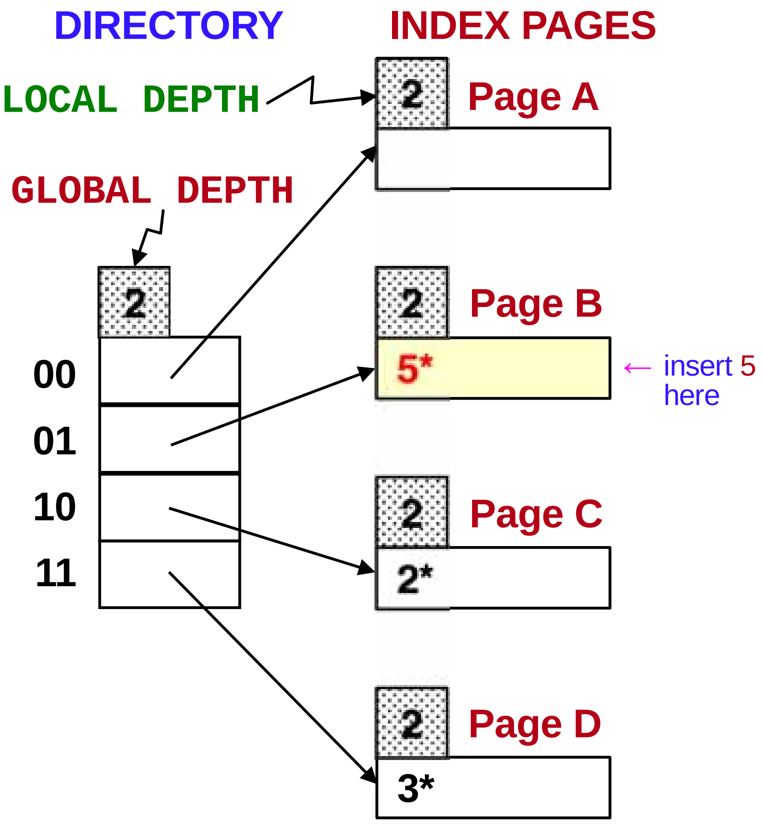


key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
binary value	010	011	101	111	011	001	011	111	101	111

page size: 3

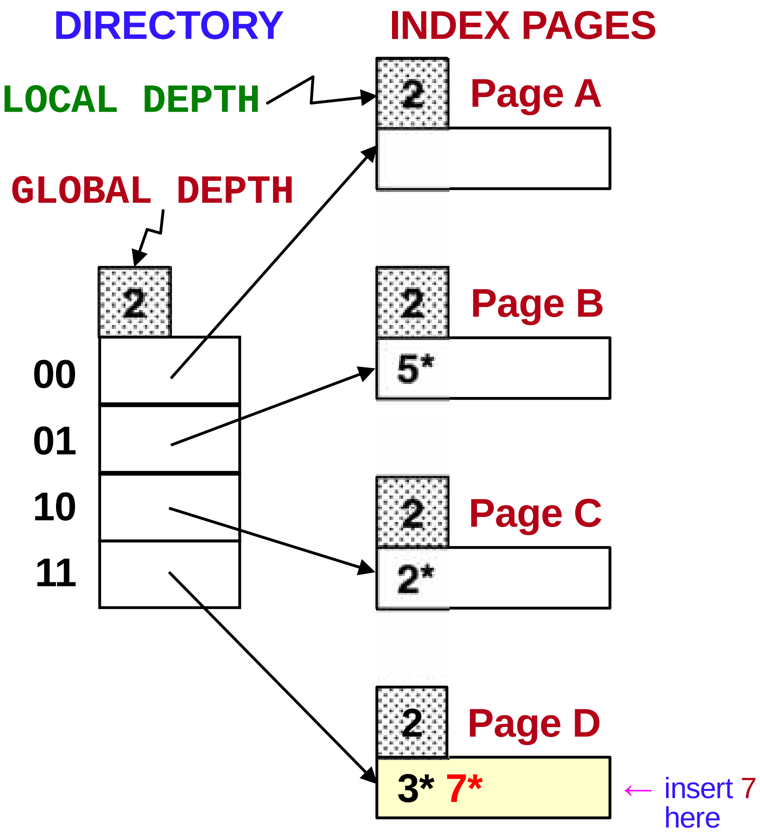
EXERCISE 4 (CONT'D)

Insert: 5 (01)



← insert 5 here

Insert: 7 (11)



← insert 7 here

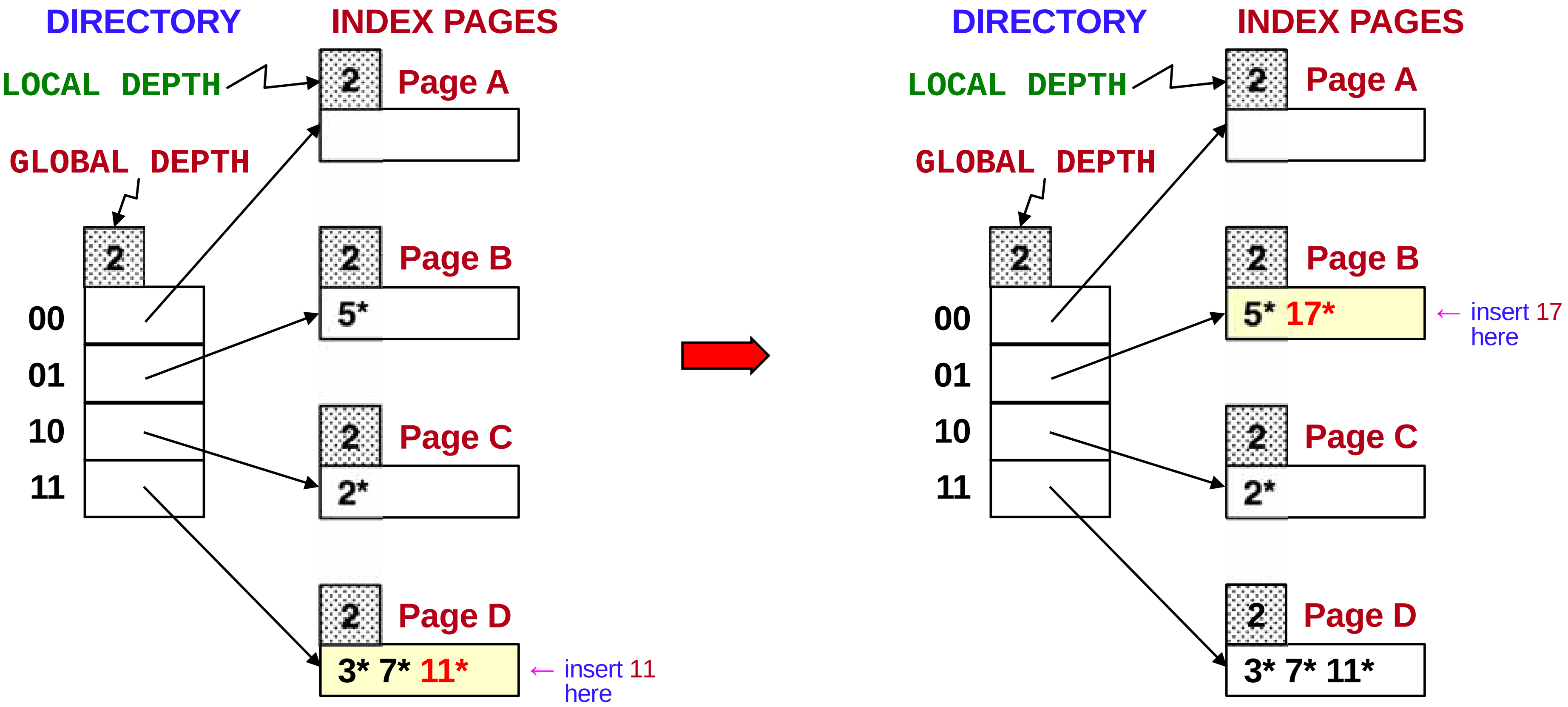
key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
binary value	010	011	101	111	011	001	011	111	101	111

page size: 3

EXERCISE 4 (CONT'D)

Insert: 11 (11)

Insert: 17 (01)

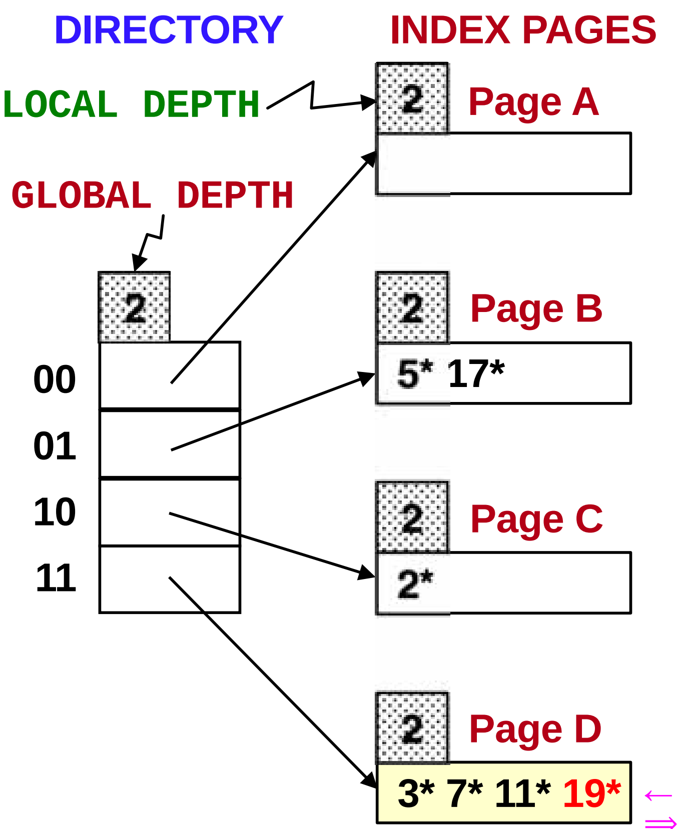


key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
binary value	010	011	101	111	011	001	011	111	101	111

page size: 3

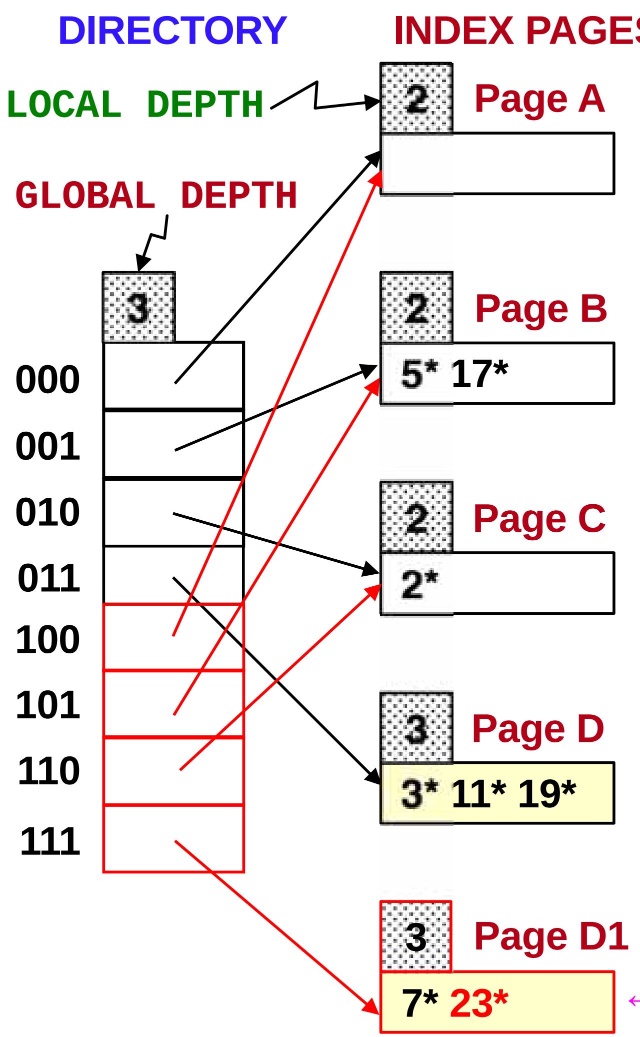
EXERCISE 4 (CONT'D)

Insert: 19 (11)



← insert 19 here
⟹ split page
⟹ local depth > global depth
⟹ double directory
⟹ redistribute

Insert: 23 (111)



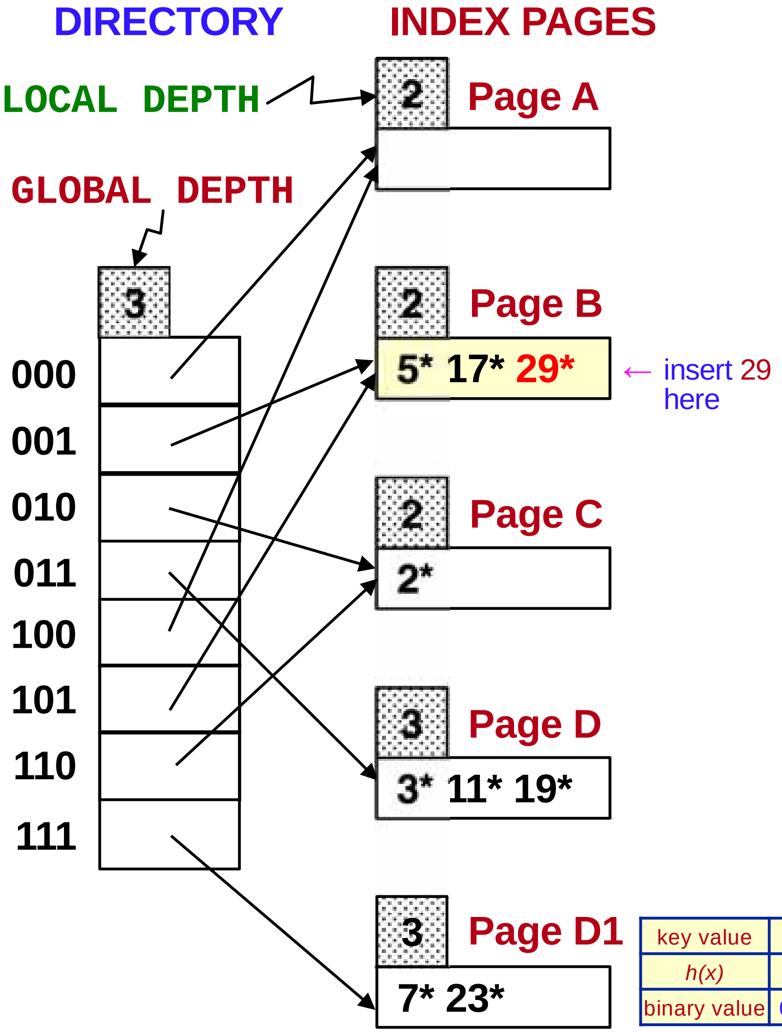
← insert 23 here

key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	**3**	5	**7**	**3**	1	**3**	7	5	7
binary value	010	**011**	101	**111**	**011**	001	**011**	111	101	111

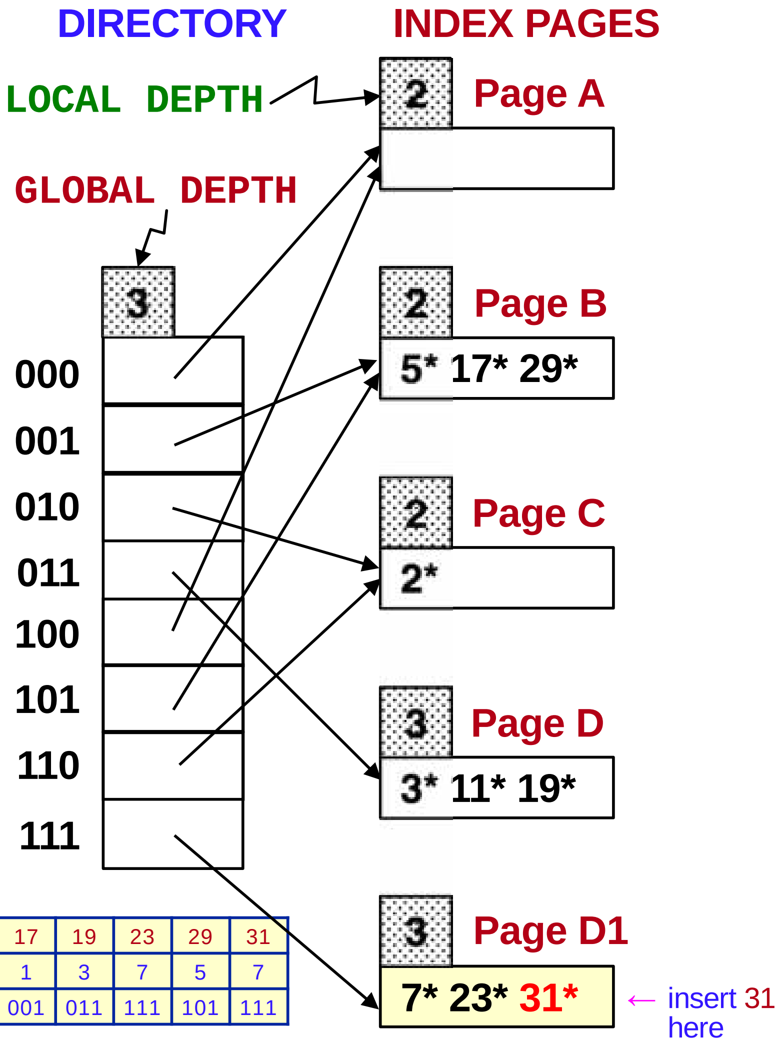
page size: 3

EXERCISE 4 (CONT'D)

Insert: 29 (101)



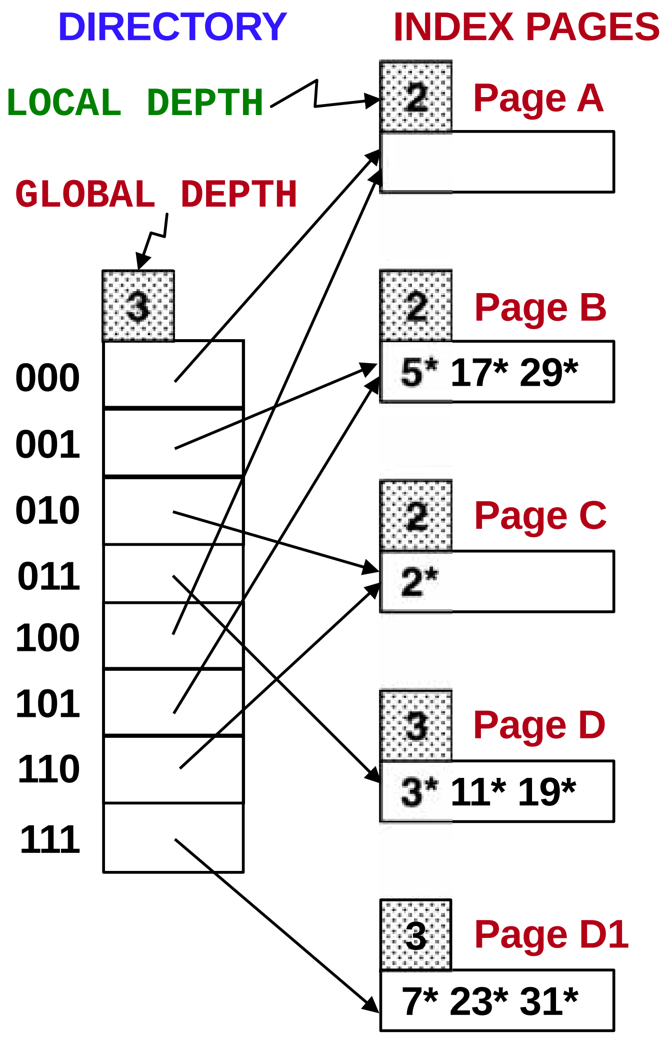
Insert: 31 (111)



key value	2	3	5	7	11	17	19	23	29	31
h(x)	2	3	5	7	3	1	3	7	5	7
binary value	010	011	101	111	011	001	011	111	101	111

page size: 3

EXERCISE 4 (CONT'D)



EXERCISE 5

Given the Customer relation and only the two bitmap indexes on gender and rating shown below, explain how you would use the bitmap indexes to answer the following queries? If the bitmap indexes are not useful, explain why.

Do not calculate the result of a query.
Explain how to obtain the result using the bitmaps.

gender index

male	female
1	0
1	0
0	1
1	0

Customer

id	name	gender	rating
112	Joe	m	2
115	Ram	m	5
119	Sue	f	5
112	Woo	m	1

rating index

1	2	3	4	5
0	1	0	0	0
0	0	0	0	1
0	0	0	0	1
1	0	0	0	0

EXERCISE 5 (cont'd)

a) How many customers with a rating less than 3 are male?

i. or the rating 1 and 2 bitmaps.
ii. and the result with the male bitmap.
iii. count the number of 1 bits in the result.

b) What percentage of customers are male?

gender index

male	female
1	0
1	0
0	1
1	0

Customer

id	name	gender	rating
112	Joe	m	2
115	Ram	m	5
119	Sue	f	5
112	Woo	m	1

rating index

1	2	3	4	5
0	1	0	0	0
0	0	0	0	1
0	0	0	0	1
1	0	0	0	0

EXERCISE 5 (cont'd)

c) How many customer there are?

count the total number of bits in any bitmap *or* use the length of any bitmap.

d) How many customer are named Woo?

The bitmaps are not useful for this query.

gender index

male	female
1	0
1	0
0	1
1	0

Customer

id	name	gender	rating
112	Joe	m	2
115	Ram	m	5
119	Sue	f	5
112	Woo	m	1

rating index

1	2	3	4	5
0	1	0	0	0
0	0	0	0	1
0	0	0	0	1
1	0	0	0	0