Problem 1

(a) LRU Replacement Policy

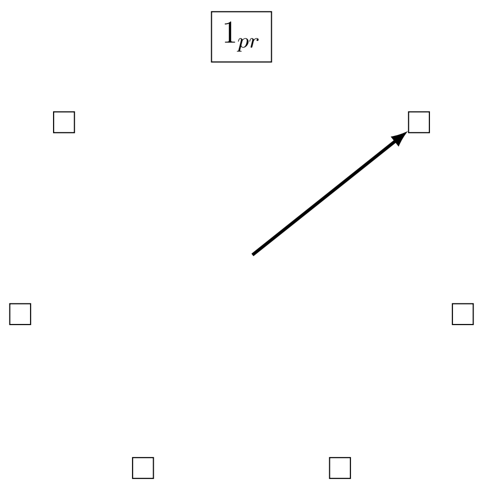
Instruction:	1*	2*	3*	$\underline{1}$	4*	5*	$\underline{3}$	$\underline{4}$	1	6	7	8*	9*	5	10
Frame 1	**1**	**1**	**1**	1	1	1	1	1	1	1	1	1	1	1	10
Frame 2		**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
Frame 3			**3**	**3**	**3**	**3**	3	3	3	3	3	**8**	**8**	**8**	**8**
Frame 4					**4**	**4**	**4**	4	4	4	4	4	**9**	**9**	**9**
Frame 5						**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Frame 6										6	6	6	6	6	6
Frame 7											7	7	7	7	7

Note: pages in bold are pinned.

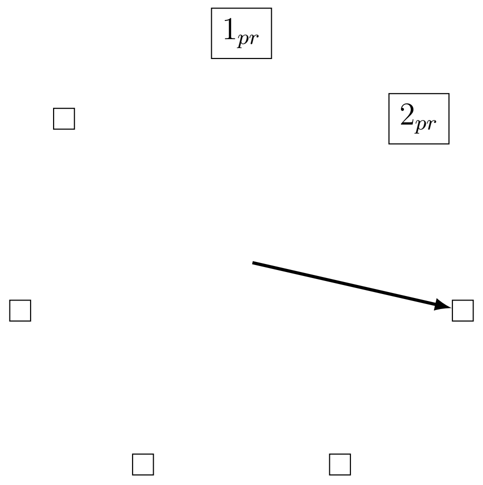
(b) Clock Replacement Policy

Note: a subscrpt p indicates that a page is pinned, and a subscript r indicates that the reference bit is set to 1.

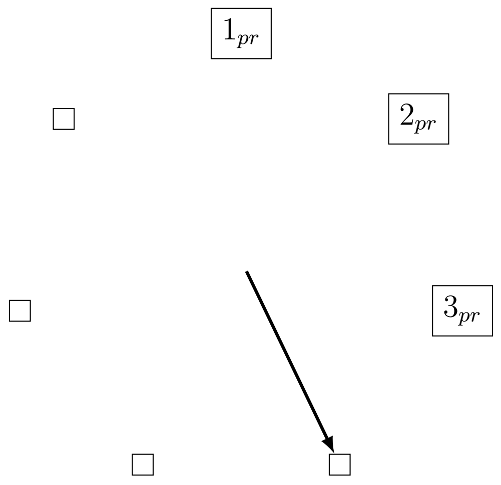
1*



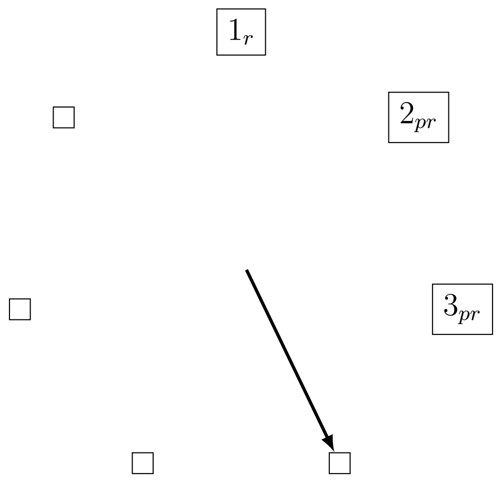
2*



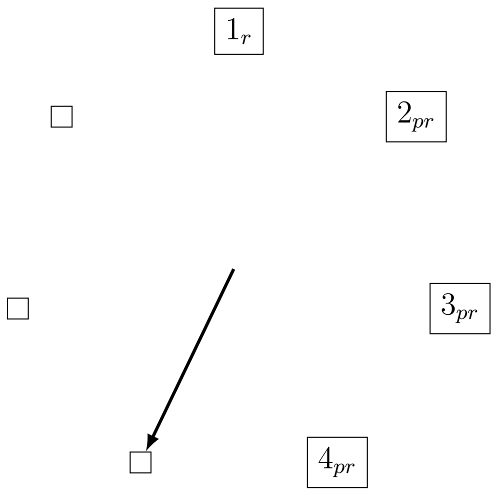
3*



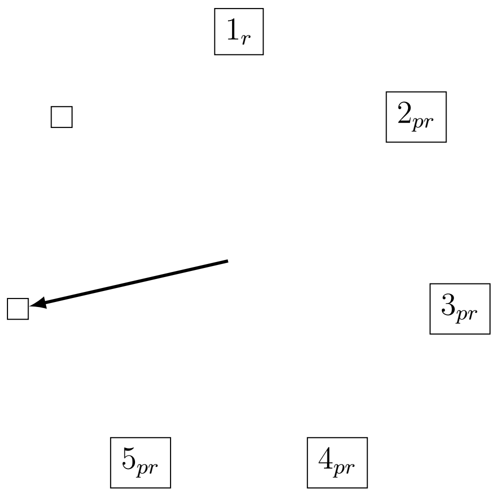
<u>1</u>



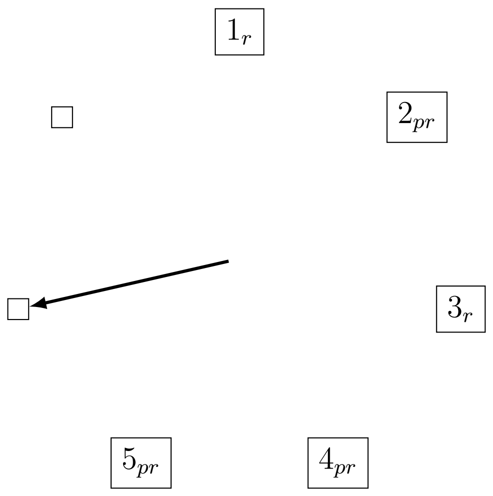
4*



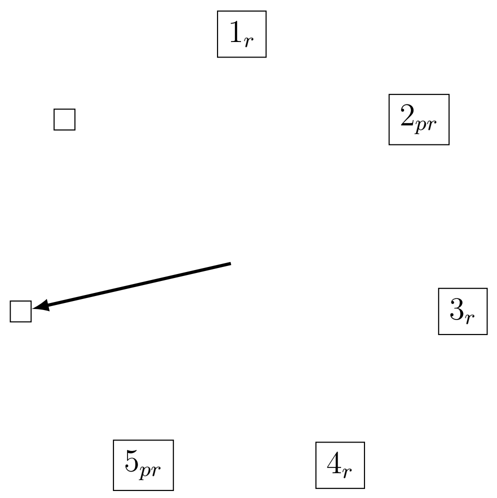
5*



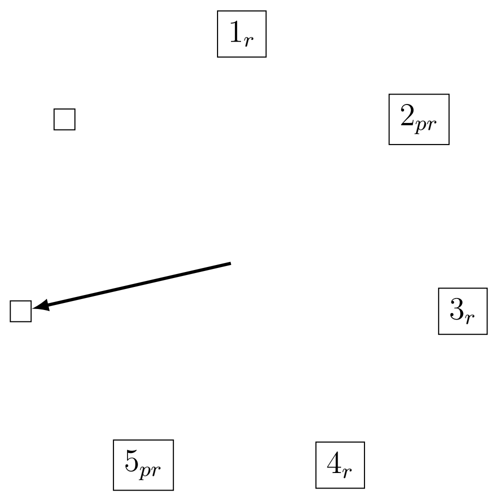
<u>3</u>



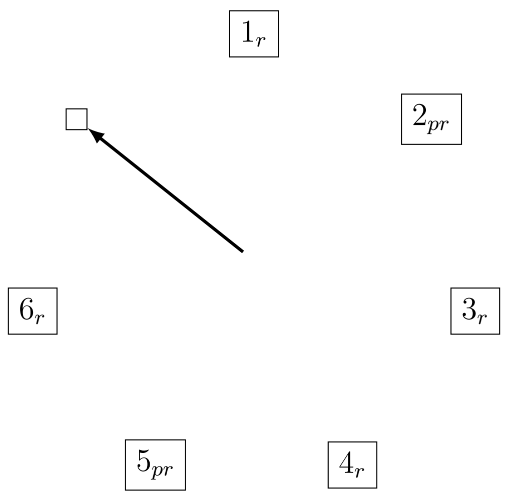
4



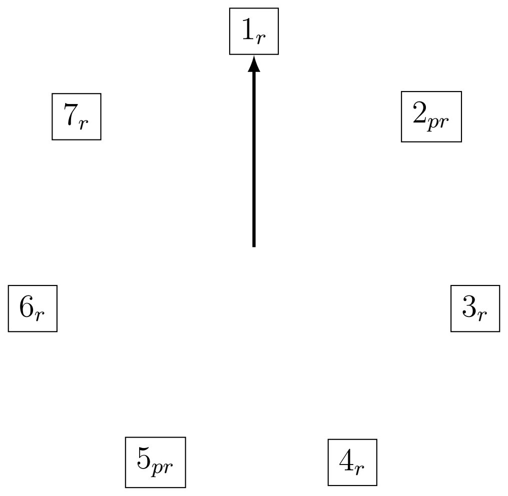
1



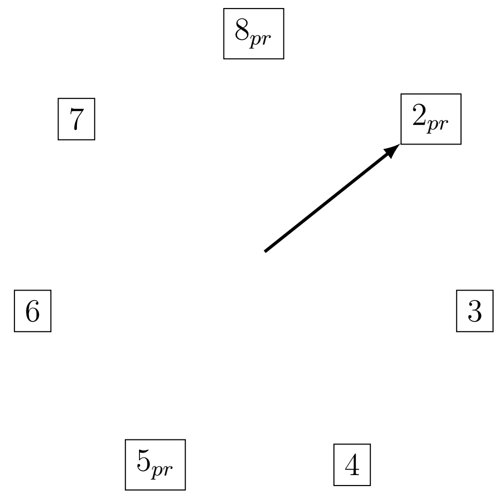
6



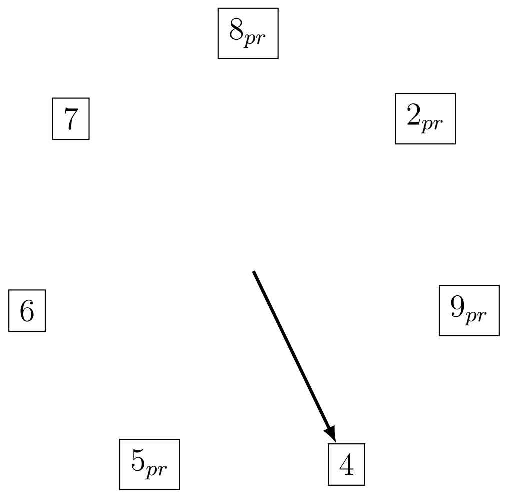
7



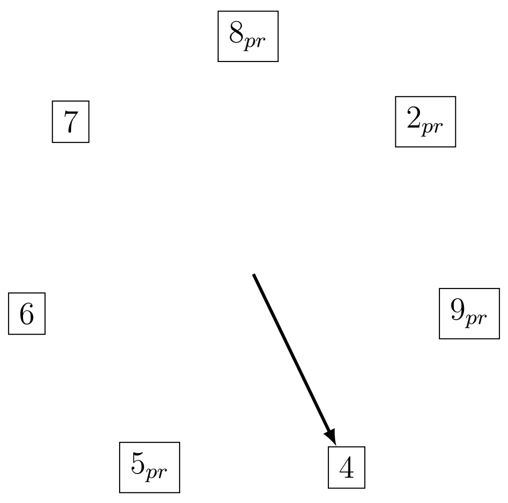
8*



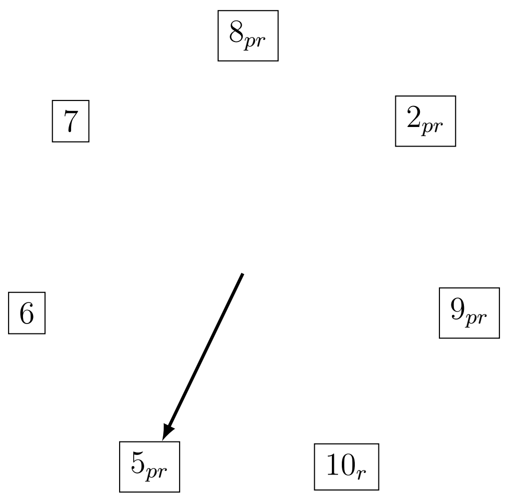
9*



5



10



Problem 2

1. Yes. Each entry in the index would contain the value of K1 (20 bytes) and a pointer to the corresponding record (8 bytes). The entire data file contains 20 million records, so the index file would contain 20 million records, with each record being 28 bytes. Each block can contain $\lfloor 8,192/28 \rfloor = 292$ index records. The entire index would therefore be $\lceil 20,000,000/292 \rceil = 68,494$ blocks.

2. Yes. Each entry in the index would contain a pointer to the start of a block in the data file, and the value of K1 for the first record in that block. Since each block in the data file contains 20 records, you would need $1,000,000/20 = 50,000$ entries in the index, which would take up $\lceil 50,000/292 \rceil = 172$ blocks.

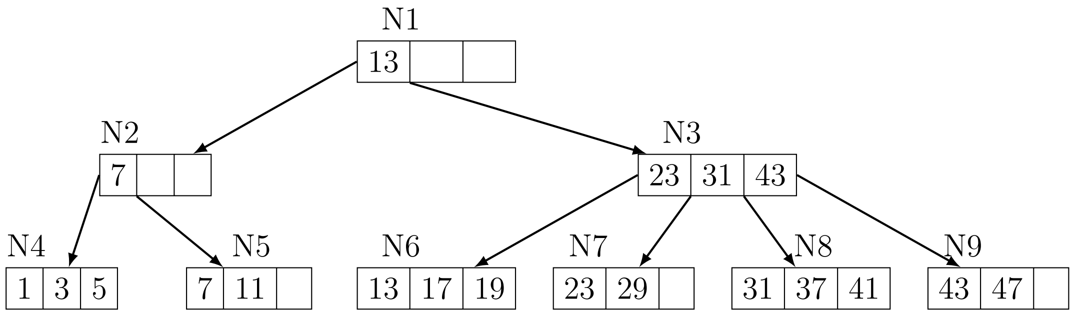
3. Yes. Each entry in the index would contain the value of K2 (20 bytes) and a pointer to the corresponding record (8 bytes). The entire data file contains 20 million records, so the index file would be 28 million bytes. Each block can contain $\lfloor 8,192/28 \rfloor = 292$ index records. The entire index would therefore be $\lceil 28,000,000/292 \rceil = 95,891$ blocks.

4. No. You can't build a sparse index on an unsorted file, and since R is only sorted on K1 a sparse index on K2 would not work (unless R happens to also be sorted over K2, or if you sort the file over K2 first).

5. Yes. You can have a second-level sparse index on the index from (1). Each entry in the second-level index would contain a pointer to a block in the dense first-level index, and the value of the first key K1 in that block. Because the dense first-level index would contain 68,494 blocks, the second-level index would have to contain 68,494 entries, each 28 bytes long. This second-level index would take up $\lceil 68,494/292 \rceil = 235$ blocks. Combined with the size of the first-level index it would be $68,494 + 235 = 68,729$ blocks.

6. Yes. You can have a second-level sparse index on the index from (2). Each entry in the second-level index would contain a pointer to a block in the sparse first-level index, and the value of the first key K1 in that block. Because the sparse first-level index would contain 172 blocks, the second-level index would have to contain 172 entries. This is small enough to fit in a single block. Combined with the size of the first level index, the total size would be $172 + 1 = 173$ blocks.

Problem 3



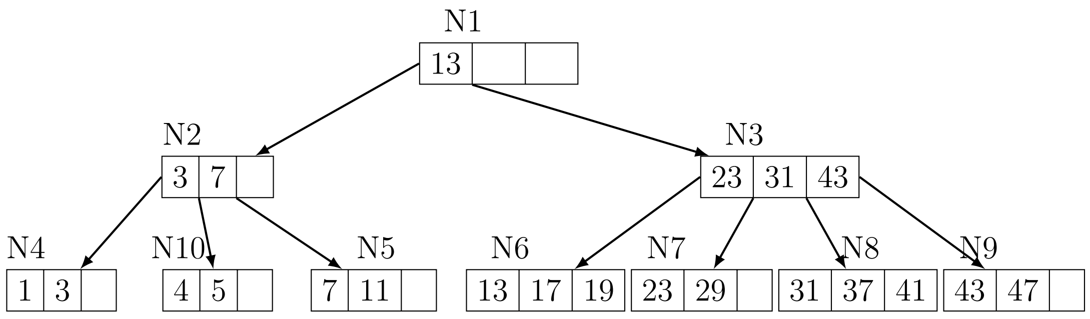
1.

First the root note, N1, would be accessed, then the program would follow the pointer to the right of the key 13 because $35 > 13$ to N3. From there, it would follow the pointer between 31 and 43 to N8. At N8 it would realize that the search key 35 is not in the data file, because the leaf layer is dense.

2.

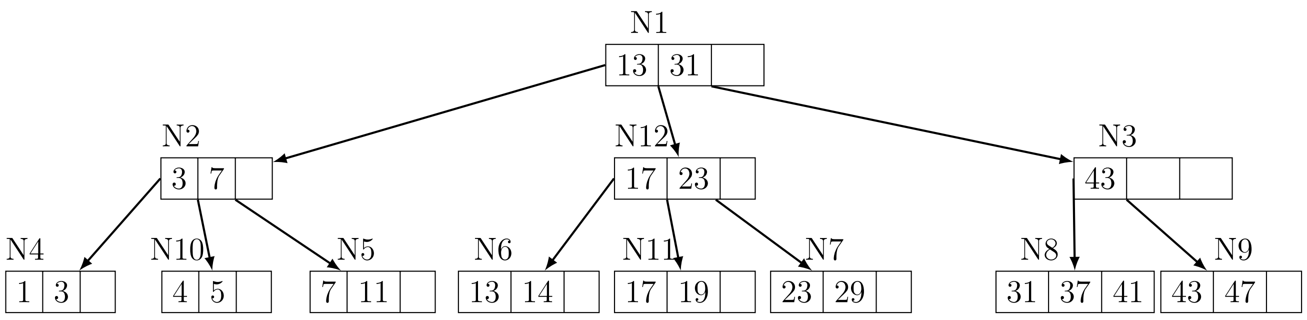
First the root node, N1, would be accessed, then the program would follow the pointer to the left of the key 13 because $9 < 35$ to N2. From there, it would follow the pointer to the right of key 7 to N5. At N5, it would retrive the key 11, then follow the leaf pointer to N6. Here, it would retrieve keys 13, 17, and 19, then follow the leaf pointer to N7. At this point it would stop, since $23 > 21$.

3.

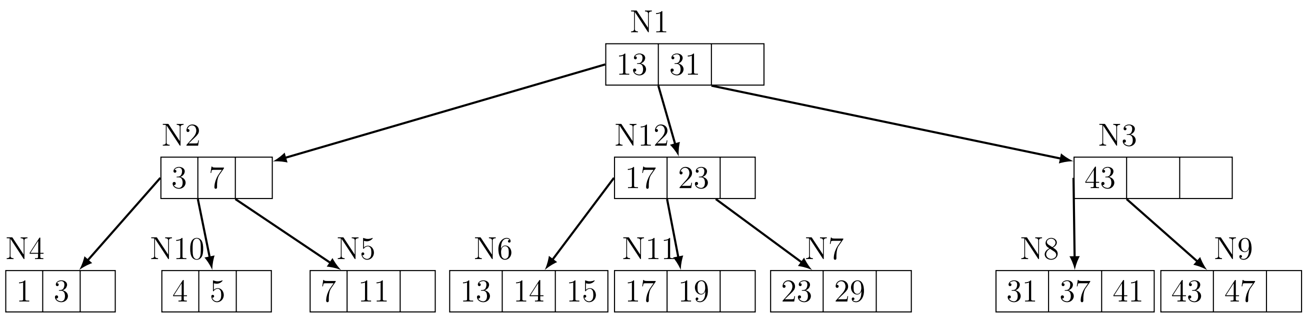


4.

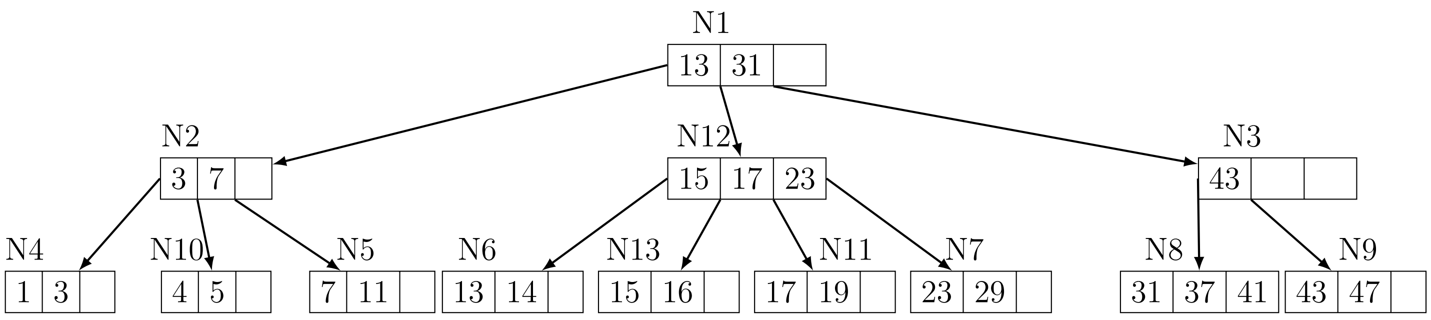
After insertion of key 14:



After insertion of key 15:



After insertion of key 16:

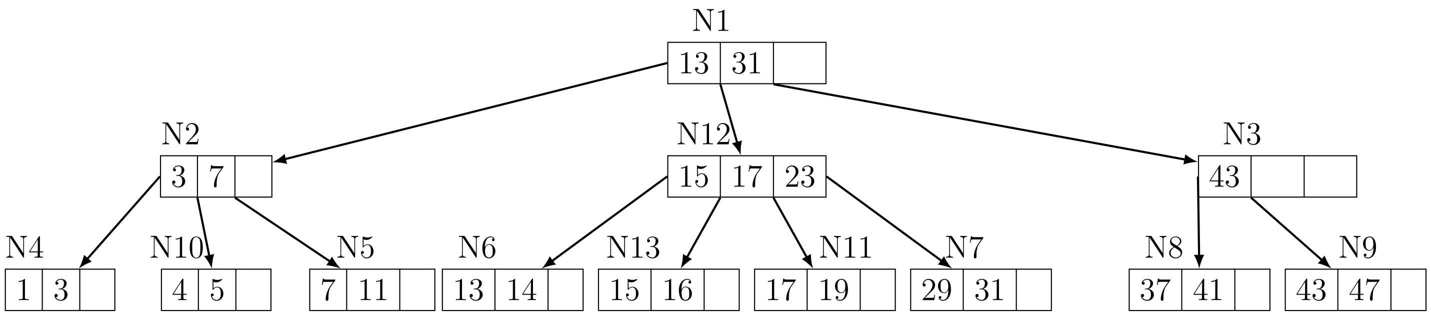


5.

Assuming we're still using the modified tree from the previous question, first the root, N1, would be accessed, then the program would follow the pointer to the left of 13 because $6 < 13$ to N2. From there, it would follow the pointer between 3 and 7 to N10. Finding no keys in the range, it would follow the leaf pointer right to N5, grabbing keys 7 and 11 from it. It would then continue right to N13, where it would grab 13 and then exit.

6.

Assuming we're still using the same tree from the previous questions, this is the result of deleting key 23:


N1
13 31
N2
3 7
N12
15 17 23
N3
43
N4
1 3
N10
4 5
N5
7 11
N6
13 14
N13
15 16
N11
17 19
N7
29 31
N8
37 41
N9
43 47