CS3223 Tutorial 1

Huang Wentao

huangwentao@u.nus.edu

Q1

Consider the Megatron 747 disk with the following characteristics:

a. There are four platters providing eight surfaces, each surface has 2^{13} = 8192 tracks, each track has 2^8 = 256 sectors, and each sector has 2^9 = 512 bytes.
b. The disk rotates at 3840 rpm.
c. To move the head assembly between cylinders takes 1 ms to start and stop, plus 1 ms for every 500 cylinders traveled. Thus, the heads move one track in 1.002 ms, and move from the innermost to the outermost track, a distance of 8192 tracks, in about 17.4 ms.
d. 10% of the space (on top of the usable space) are used for gaps, i.e., between 2 sectors, there is a gap that is not used to store data.

Answer the following questions.

1) What is the capacity of the disk?
2) What is the minimum and maximum time it takes to read a 4096-byte block (8 consecutive sectors) from the disk?
3) Suppose that we know that the last I/O request accessed cylinder 1000. What is the expected (average) block access time for the next I/O request on the disk?

Q1

1)

capacity = 8 * 2^13 * 2^8 * 2^9 = 2^33 = 8 GB

Q1

2)

to transfer 8 sectors, we need to read 8 sectors + 7 gaps
rotational delay t = 60/3840 = 1/64 = 15.6ms
read 1 sector + 1 gap, s = t/number of sector = 0.061 ms
read 1 sector w/o gap = 0.9*s = 0.055 ms
transfer time tb = 7*0.061 + 0.055 = 0.482 ms
average time to fetch a block = seek time + rotational delay + transfer time

min transfer occurs when seek time = rotational delay = 0,
min = 0.482

max occurs when head move from innermost to outermost track + full rotational delay
max = 17.4 + 15.6 + 0.48 = 33.5 ms

Q1

3)

the next I/O request could be from any of the cylinders 1, 2, ... 8192 with equal likelihood.

the no. of cylinders travelled in these cases would be 999, 998,
... 7192. Hence the average no. of cylinders travelled would be
(999+998+...+1+0+1+...+7192)/8192 = 3218.45

seek time = start time + no. of 500 tracks*1 = 1+3218.5/500 = 7.44 ms
average rotational delay = (60/3840) /2 = 7.81ms
transfer time = 0.482 ms

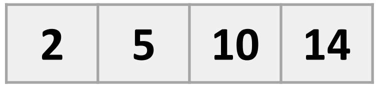
expected block access time = 7.44 + 7.81 + 0.482 = 15.73 ms

Q2

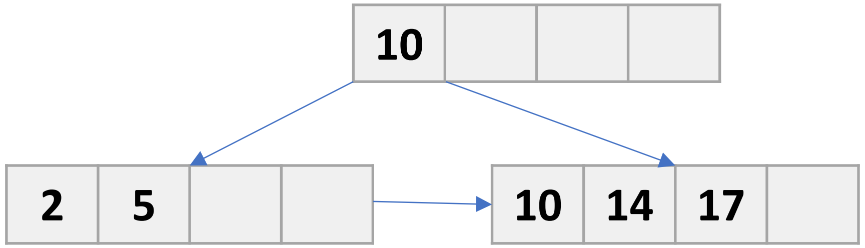
Consider a B+-tree of order 2 for this problem, that is initially empty. Show what the tree looks like at this point after each of the following actions:

1) You insert the key 10 (and a pointer to this record) into the tree.
2) Next you insert keys 2, 5, 14, and 17 (in this order) into the tree.
3) Next you insert keys 1, 3, 4, 6, 7, 8, 9, 11, 12, 13, 15 and 16 (in this order).
4) Delete keys 1, 2 and 3.

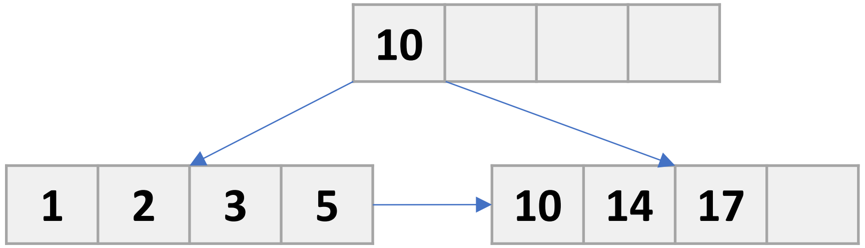
Q2



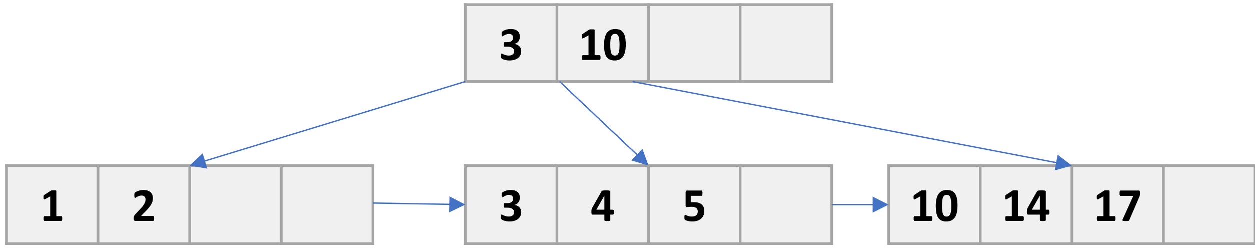
(1)



(2)

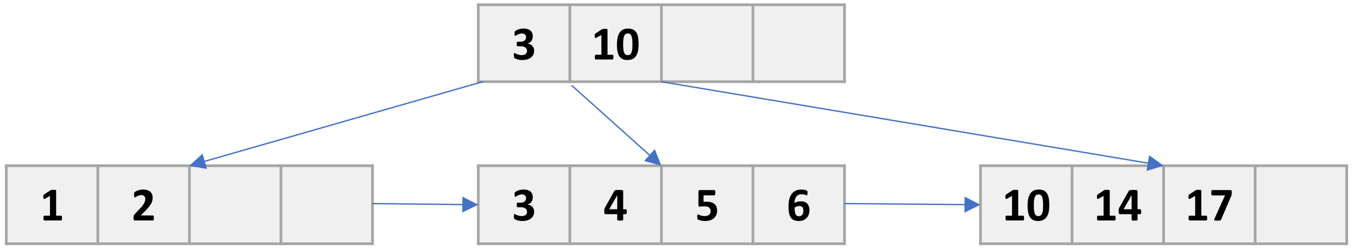


(3)

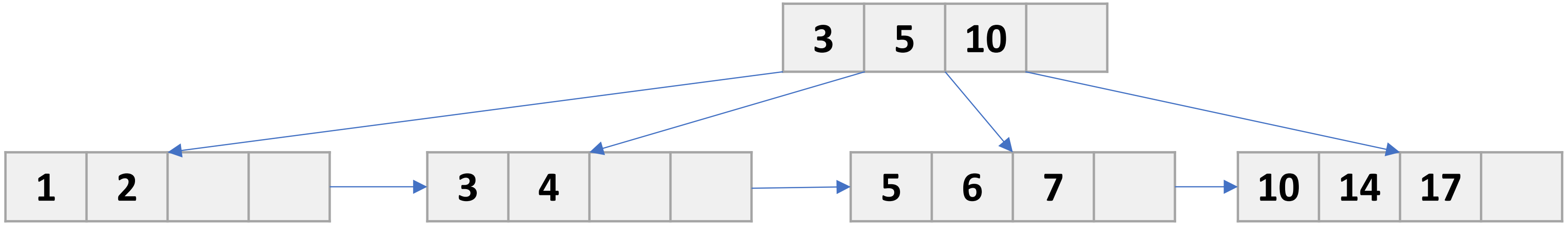


(4)

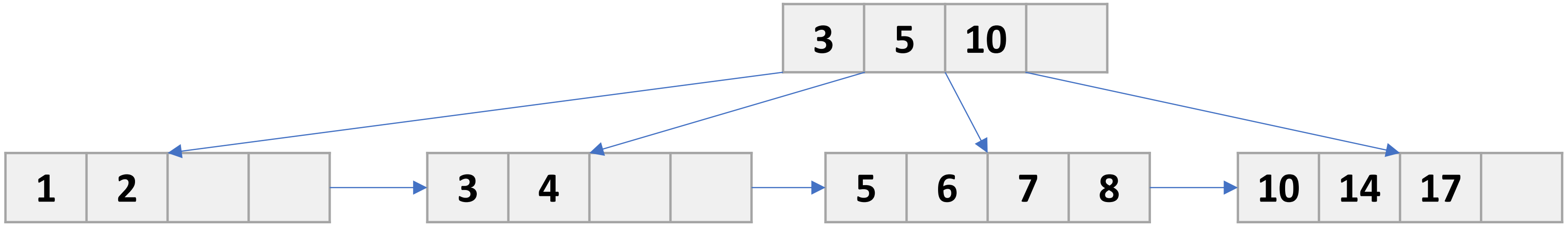
Q2


3 10
1 2
3 4 5 6
10 14 17

(5)

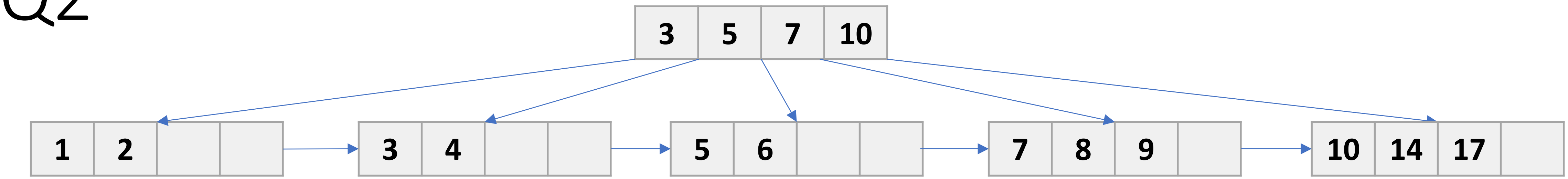

3 5 10
1 2
3 4
5 6 7
10 14 17

(6)

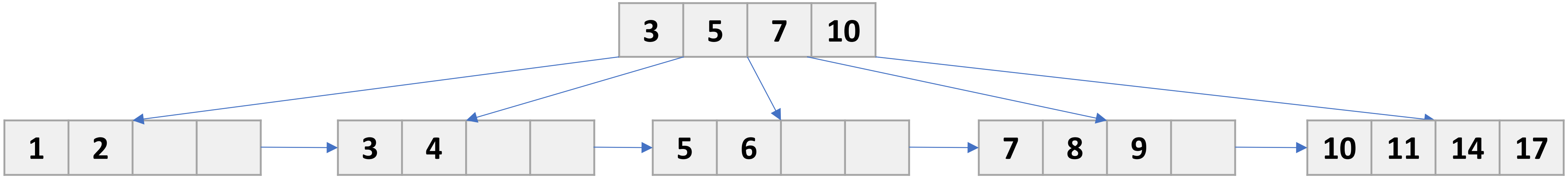

3 5 10
1 2
3 4
5 6 7 8
10 14 17

(7)

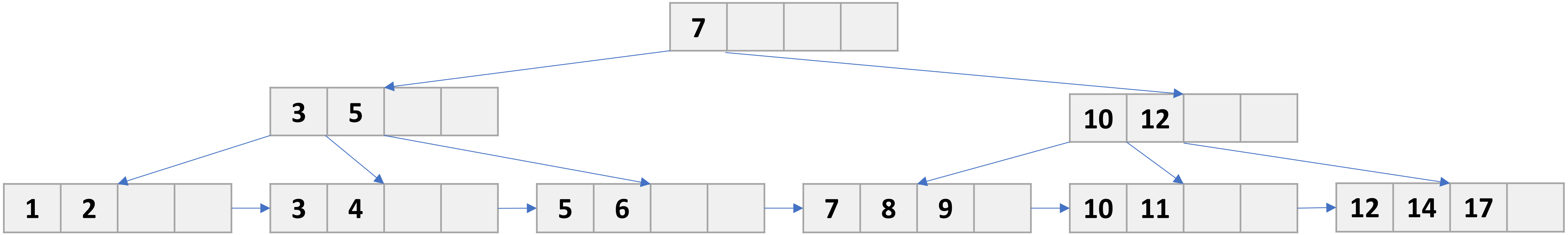
Q2


3 5 7 10
1 2
3 4
5 6
7 8 9
10 14 17

(8)

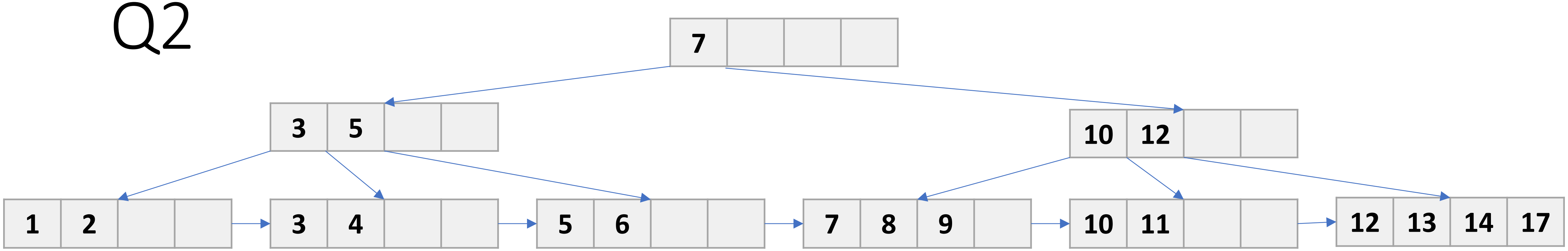

3 5 7 10
1 2
3 4
5 6
7 8 9
10 11 14 17

(9)

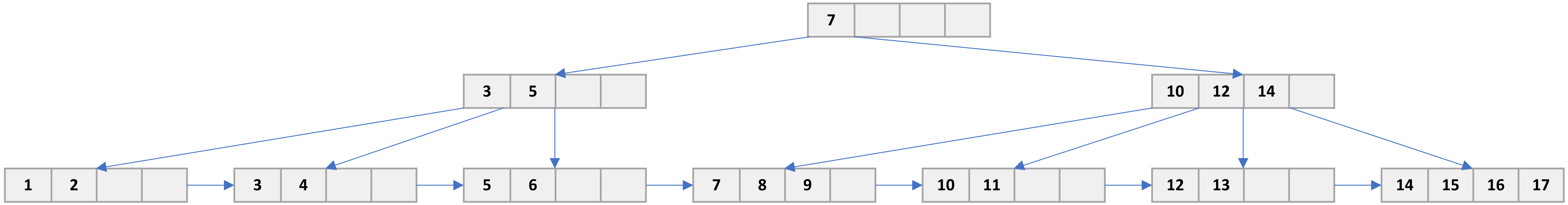

7
3 5
10 12
1 2
3 4
5 6
7 8 9
10 11
12 14 17

(10)

Q2

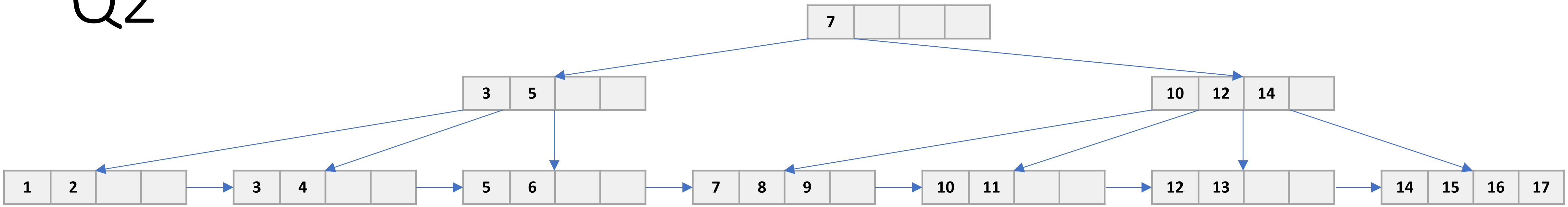

7
3
5
10
12
1
2
3
4
5
6
7
8
9
10
11
12
13
14
17

(11)

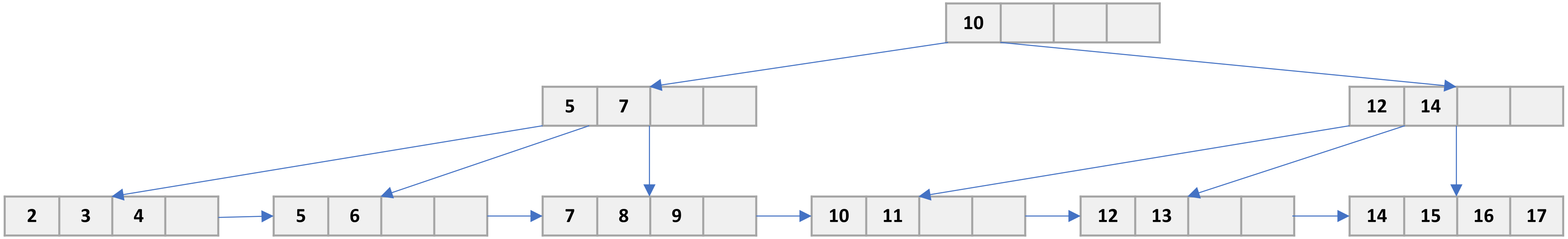

7
3
5
10
12
14
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

(12)

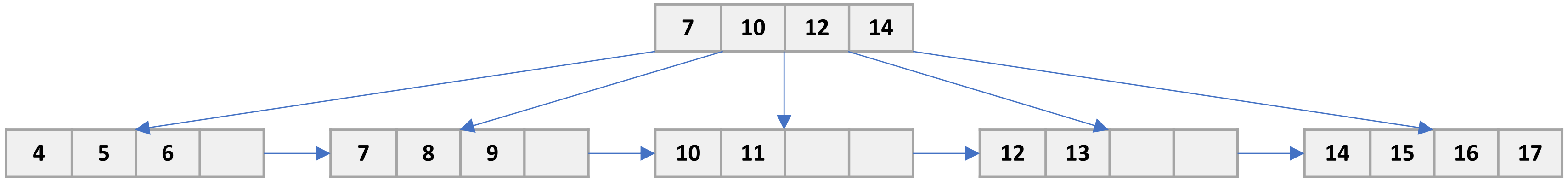
Q2


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

(13)


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

(14)


7 10 12 14
4 5 6
7 8 9
10 11
12 13
14 15 16 17

(15)

Q3

For some applications, the values of keys only increase. For example, consider a file of employees where ID numbers are never reused as employees leave the company, and a new number, higher than any used before, is assigned to each new employee. Suppose you would like to index the file on this key using a B+-tree. What is the effect on the B -tree as records are inserted, and how would you modify the B+-tree insertion algorithm to improve performance?

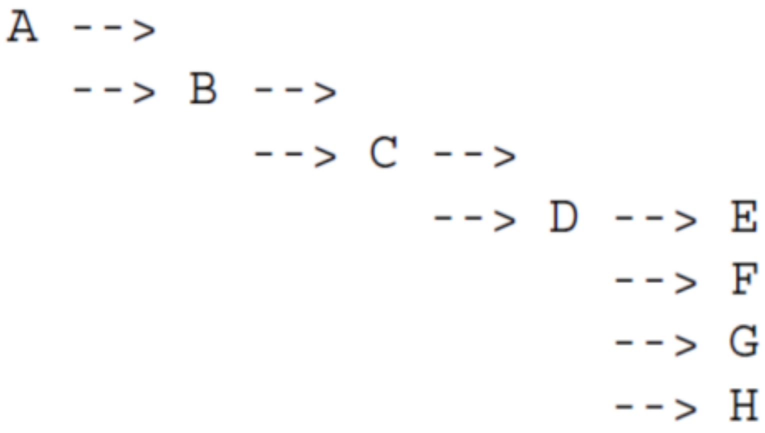
Q3

1) Current Issues:
    a. Space utilization is 50%
    b. Tree will be taller than desired — poor search performance

2) Modification:
    a. When splitting a node, keep the left node full while the right node contains only one record.
    b. Set a pointer at the end of the last node so that to save time for insertion.

Q4

To manage the buffer pool more efficiently, a database buffer manager might distinguish between different types of disk pages that may have different access patterns. In particular, it has been pointed out that index pages (e.g., B+-tree access) often have a different pattern of access compared to data pages. Consider the example index shown in the figure below:



```
A -->
  --> B -->
        --> C -->
              --> D --> E
                    --> F
                    --> G
                    --> H
```

Q4

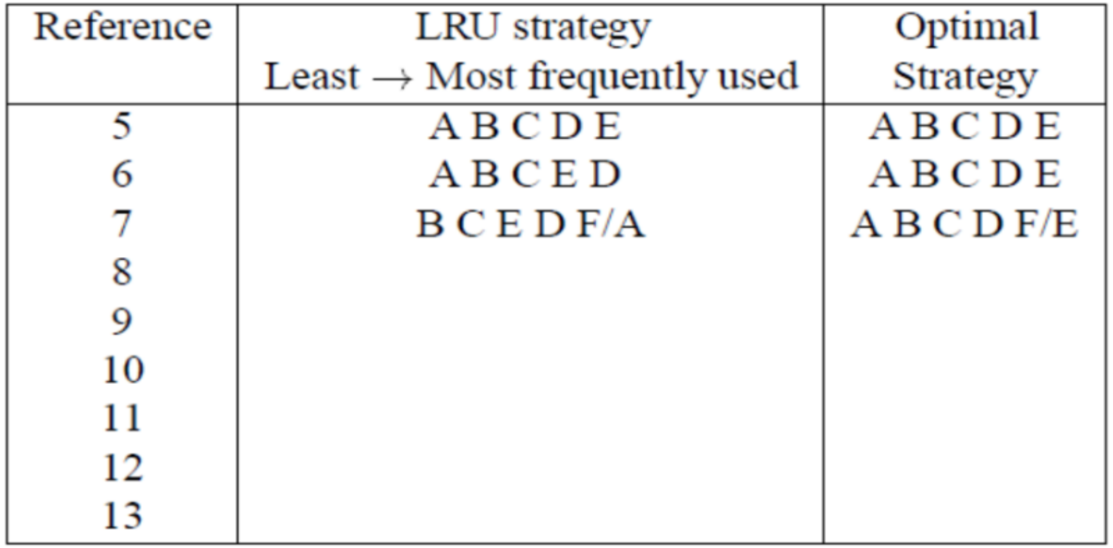
Node A is the root node of a B+-tree index, B, C, and D are index nodes, and E, F, G, H leaf nodes that contain the clustered data records. Consider the following logical page reference string corresponding to an index traversal

$$A_1, B_2, C_3, D_4, E_5, D_6, F_7, D_8, G_9, D_{10}, H_{11}, D_{12}, C_{13}, \ldots$$

The subscript in each page number denotes the reference sequence number.

Assume that the allocated buffer has 5 pages. Contrast the use of an LRU replacement policy and an Optimal replacement policy (i.e., one which victimizes the page with the longest expected time until its next reference.)

Q4



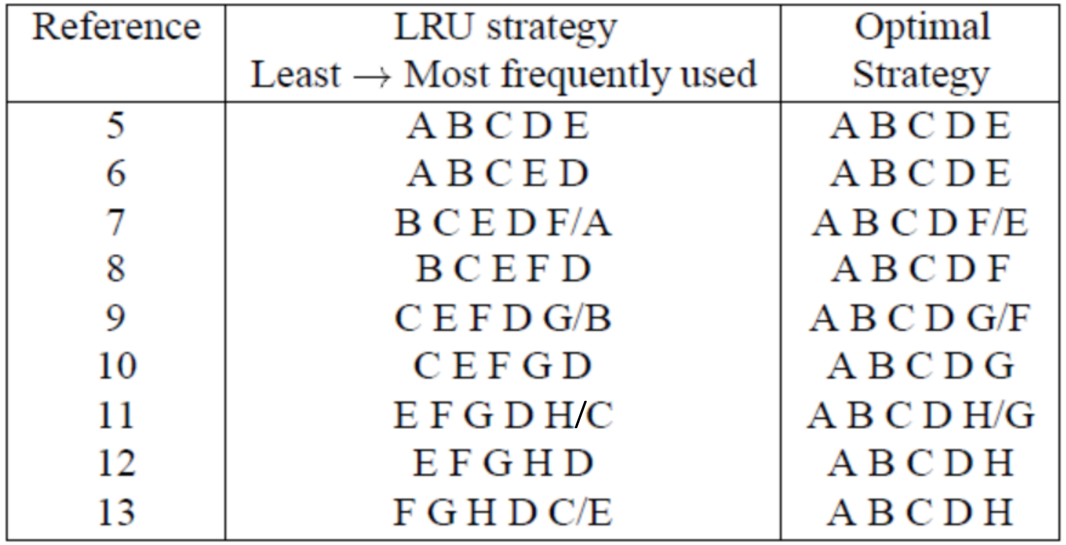
Reference	LRU strategy	Optimal
	Least → Most frequently used	Strategy
5	A B C D E	A B C D E
6	A B C E D	A B C D E
7	B C E D F/A	A B C D F/E
8		
9		
10		
11		
12		
13		

(A buffer entry I/J means that a page replacement has occured with page I replacing page J.)

Complete the above table. Why is LRU not an effective replacement algorithm here. Can you suggest a better strategy than LRU?

Q4

A_1, B_2, C_3, D_4, E_5, D_6, F_7, D_8, G_9, D_{10}, H_{11}, D_{12}, C_{13}, …



Reference	LRU strategy Least → Most frequently used	Optimal Strategy
5	A B C D E	A B C D E
6	A B C E D	A B C D E
7	B C E D F/A	A B C D F/E
8	B C E F D	A B C D F
9	C E F D G/B	A B C D G/F
10	C E F G D	A B C D G
11	E F G D H/C	A B C D H/G
12	E F G H D	A B C D H
13	F G H D C/E	A B C D H

- The LRU is clearly suboptimal because it chooses to replace useful pages like B and C which are needed later. Examine for instance line 13 in the above table: under the LRU, only one (C) out of five buffer pages in memory is useful. Under the ideal situation, we have pages A, B, and C which are much more likely to be referenced in the future.
- A more optimal strategy here is to choose pages for replacement based on the corresponding level of the page in the B+-tree.

Extra Question 1

Suppose we use double buffering to read blocks from a hard disk to improve the IO performance. That is, the disk controller has two buffers: one to store the data from a disk block, and the other for an application process to consume the data of another block. Assume each disk block needs time R to read, and the application needs time P to process one block. What is the total time for the application to read in 10 disk blocks and consume the data?

Extra 1

The time is always: P + R + 9*max(P,R).

If P ≥ R, it takes 10 * P + R, since we need to spend 10 * P to consume the data, plus R to read the first IO. The time of reading the other blocks is overlapping with the time to consume the data.

If P ≤ R, it takes 10 * R + P, since we need to spend 10 * R to consume the data, plus P to process the last block. The time of processing the other 9 blocks is overlapping with the time to read the data.

Extra Qusetion 2

Consider a hard disk with the following specifications.

1) 3.5in in diameter
2) 10 platters, and 2 surfaces each platter
3) Usable capacity: 10GB
4) Number of cylinders: 256
5) 1 block = 4 KB
6) No overhead between blocks (gaps).

Suppose the inner 1.5-inch diameter is not used for data. Consider the following two options.

Option A: Different tracks have the same number of sectors.

Option B: The number of sectors in the innermost track is 50% of the average number of sectors per track, and the number of sectors in the outermost track is 200% of the average number of sectors per track.

Calculate the densities of the innermost track and outermost track under these two options.

Extra 2

Option A:
Average capacity of a track = 10GB/(10*2*256) = 2MB
Density of the innermost track = capacity of a track / (pi * inner diameter) = 2MB / (3.1416 * 1.5in) = 0.424MB/inch.
Density of the outermost track = capacity of a track / (pi * outer diameter) = 2MB / (3.1416 * 3.5in) = 0.182MB/inch.

Option B:
Average capacity of a track = 10GB/(10*2*256) = 2MB
Density of the innermost track = 2MB * 0.5 / (pi * inner diameter) = 1MB / (3.1416 * 1.5in) = 0.212MB/inch.
Density of the outermost track = 2MB * 2 / (pi * outer diameter) = 4MB / (3.1416 * 3.5in) = 0.364MB/inch.