OS

File ~~Memory~~ management

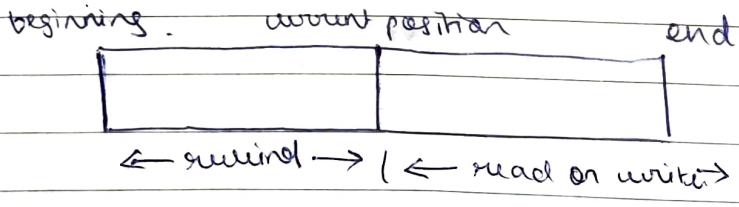
9.(1) File access method

When file is used, information is read and accessed into computer memory and there are several ways to access these information of the file.

There are three ways to access file into computer system:

⇒ **Sequential Access**

In this type of file, a fixed length is used for records. All records are of the same length, consisting of the same number of fixed length fields, in a particular order. Usually the first field in each record is referred to as the key field. Access requires the sequential search of the file for a key match.



⇒ ~~Re~~ Direct Access

For direct access, the file is viewed as a numbered sequence of blocks or records.

There is no concept of sequential ordering here.

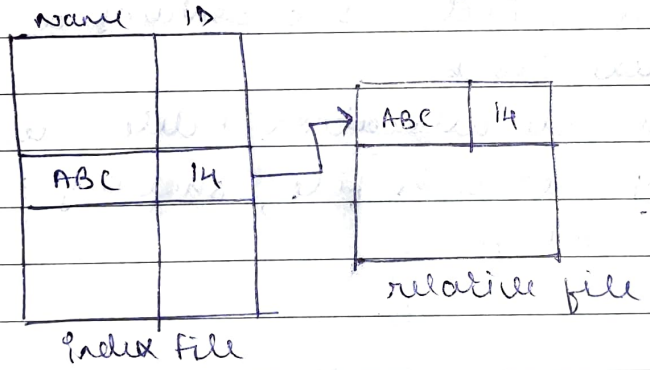
Thus, we may read block 14, then read block 53, and then write block 7.

⇒ Index Sequential method

It can be built on top of a direct access method. In this method, we construct an index for the file.

The index contains the pointer to the various blocks.

To find the record in the file, we first search the index and then by the help of pointer we access the file directly.

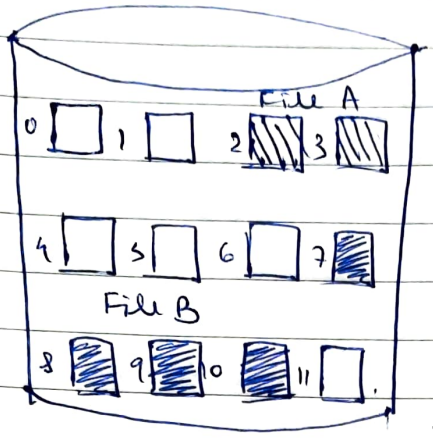


Q2) File allocation method

1) Contiguous Allocation

With contiguous allocation, a single contiguous set of blocks is allocated to a file at the time of file allocation.

The file allocation table needs just a single entry for each file showing the starting block & the length of the file.



File Allocation table

File name	start Block	length
File A	2	2
File B	7	4

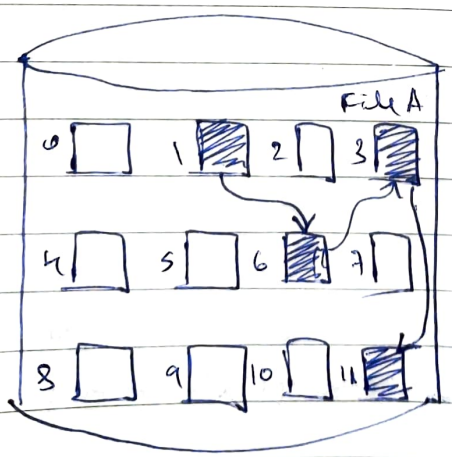
Here we can have sequential & direct access

DISADV : ⇒ external fragmentation will occur, making it difficult to find contiguous blocks of space.

2) Chained Allocation

Here, allocation is on an individual ~~bas~~ block basis. Each block contains a pointer to the next block.

The file allocation table needs just a single entry for each file, showing the starting block & the end block.



File allocation table

File name	start	end
File A	1	11

~~DISADV~~ ⇒

Here, we can have sequential access.

DISADV : ⇒ access time is more
⇒ there is no ~~this~~ accomodation of the principle of locality.
⇒ one block - one address → more space each block

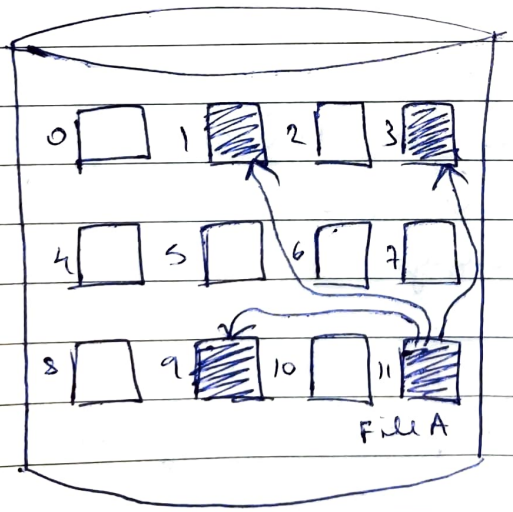
3) Indexed Allocation

It overcomes many of the problems of contiguous and chained allocation.

Here, allocation may be on the basis of either fixed size blocks or variable size portions.

Allocation by blocks eliminates external fragmentation, whereas allocation by variable size portions improves locality.

The file allocation table contains a separate one level index for each file; the index has one entry for each portion allocated to the file.



File Allocation table

File name	Index block
File A	11

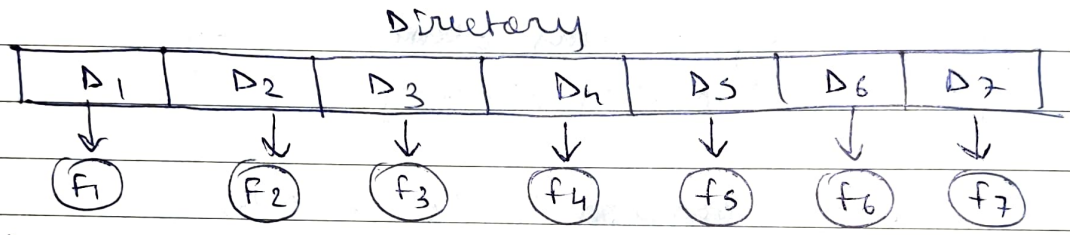
// Here we can have Direct + Index access

Q③ Directory structures

⇒ A directory is a container that is used to contain folders and files.

1) Single level directory

Single level directory is the simplest directory structure. In it all files are contained in the same directory which makes it easy to support and understand.



ADV : ⇒ since its single directory, implementation is easy.

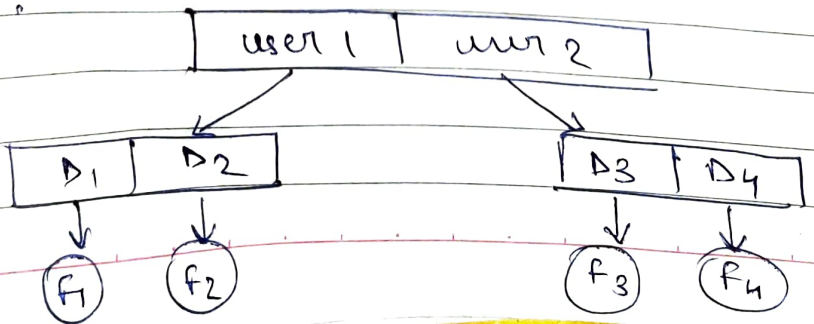
⇒ If files are smaller in size, searching is easy.

DISADV : ⇒ There may be chances of name collision

⇒ Searching will become time taking if the directory is large.

2) Two level directory

In this, to solve the problem of confusion of file names among different user, the solution is to create a separate directory for each user.



ADV : ⇒ Different ~~users can~~ users can have the same directory as well as file name

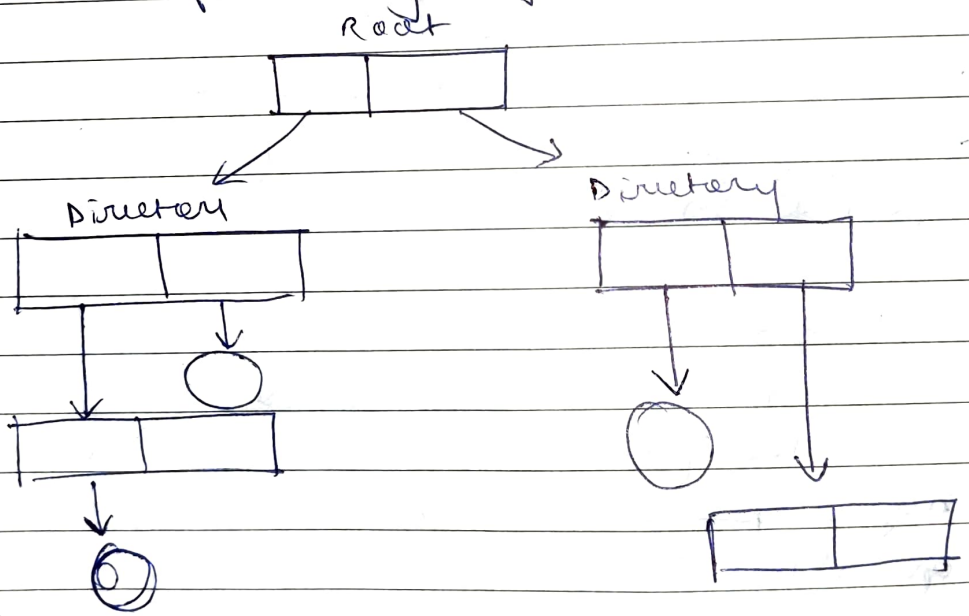
⇒ searching becomes easier due to pathname.

DISADV : ⇒ User is not allowed to share files with other users

⇒ Still, it is not very scalable.

3) Tree structure directory

We have seen two level directory as a tree of height 2. Here we extend the directory structure to a tree of arbitary height.



ADV :

⇒ Very scalable

⇒ Very general

⇒ Searching becomes very easy

DISADV :

⇒ We cannot share files.

⇒ It is inefficient, because accessing a file may go under multiple directories

8④ File operations

1) creating a file

Create operation is used to create a file in the file system.

two steps are needed.

First, space in the file system must be found for file

second, an entry for new file must be made in the directory.

2) writing a file

write operation is used to write the operation into a file.

To write a file, we make a system call specifying both the name of the file and the information to be written.

3) Opening a file

Open operation is used to open a file. once, the file is created, it must be opened before ~~using~~ performing the file processing operations.

4) Reading a file

Read operation is ~~used to~~ used to read the content from a file.

To read from a file, we use a system call that specifies the name of the file & where the next block of the file should be put.

5) Repositioning within a file (File seek)
The directory is searched for the appropriate entry, & the current file position pointer repositioned to a given value.

6) Deleting a file
To delete a file, we search the directory for the named file.
After finding the entry, we release all file space, so that it can be reused by other files.

7) Truncating a file
This is simply deleting the file except deleting the attributes. Rather than deleting file, this function allows all attributes to remain unchanged, except for file length

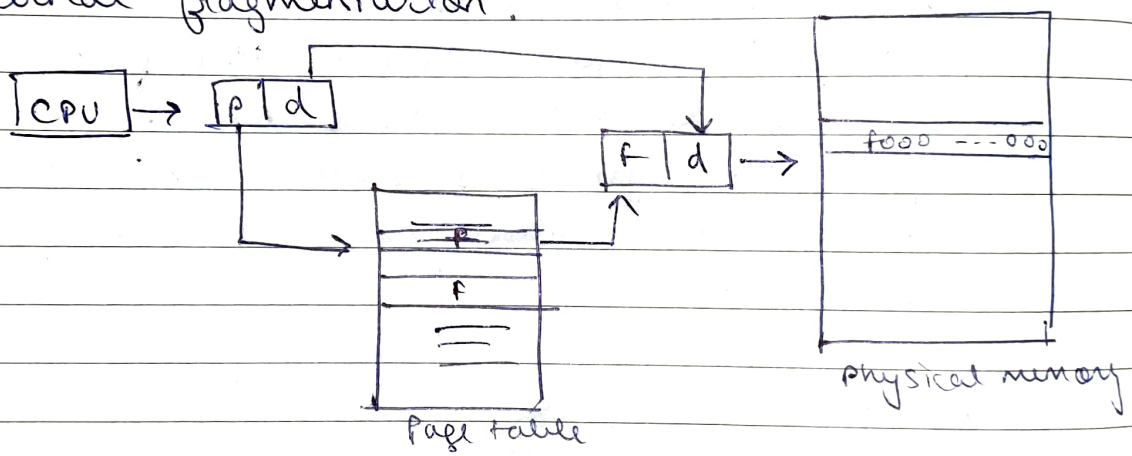
8) Closing a file
When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released.

Q(5) Paging

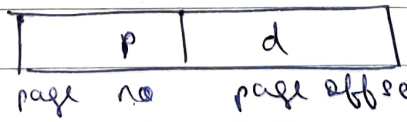
Paging is a memory management scheme that pains the physical address space of a process to be non-contiguous.

In paging method, the main memory is divided into small fixed size blocks of physical memory which is called Frames.

The size of a frame should be kept the same as that of a page to have maximum utilization of the main memory and to avoid external fragmentation.



Every address generated by the CPU is divided into two parts: => page number
=> page offset



When a process arrives in the system to be executed, its size expressed in pages is examined. Each page of the process needs one frame. Thus, if it requires n pages, at least n frames must be available. If n are available, they are allocated.

ADV ⇒ Easy to use memory management
⇒ No need for external fragmentation

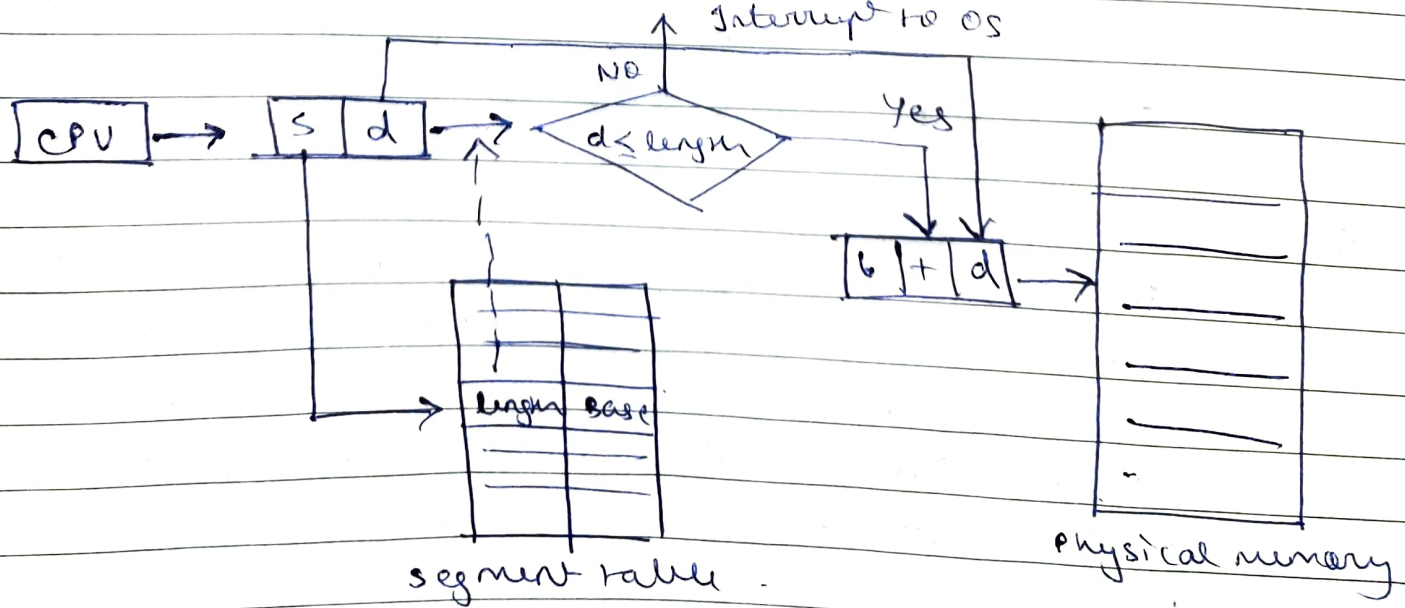
DISADV ⇒ Page tables consume additional memory
⇒ Multi-level paging may lead to memory reference overload
⇒ Internal fragmentation

(6) Segmentation

This method works almost similarly to paging, only difference between two is that segments are of variable length, whereas in paging, pages are of fixed size

Here, the logical address is in two parts: the segment name and its offset
⇒ In segmentation, there is no need to know the length of each segment, as every segment may be of different size.

Therefore, another field known as limit is added to segment table to check the length of segments.



ADV :⇒ Not offers internal fragmentation
⇒ segment tables use lesser memory than paging.

DISADV : ⇒ costly memory management system.

Q(7) Thrashing

In order to handle page fault it is necessary to replace the existing page from memory.

⇒ Since in paging, pages are transferred between main memory and disk, this has an enormous overhead.

Because of this frequent movement of pages, the system throughput reduces.

⇒ This frequent paging activity causing the reduction in system throughput is called Thrashing.

∴ Thrashing is a condition or a situation when the system is spending a major portion of its time ~~search~~ servicing the page faults, but the actual processing done is very negligible.

Q.(8) Demand paging

Deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult, because we cannot say in advance that a process will require a particular page at a particular time.

⇒ Therefore, to overcome this problem, there is a concept called Demand paging.

It suggests keeping all pages of the frames in the secondary memory until they are required.

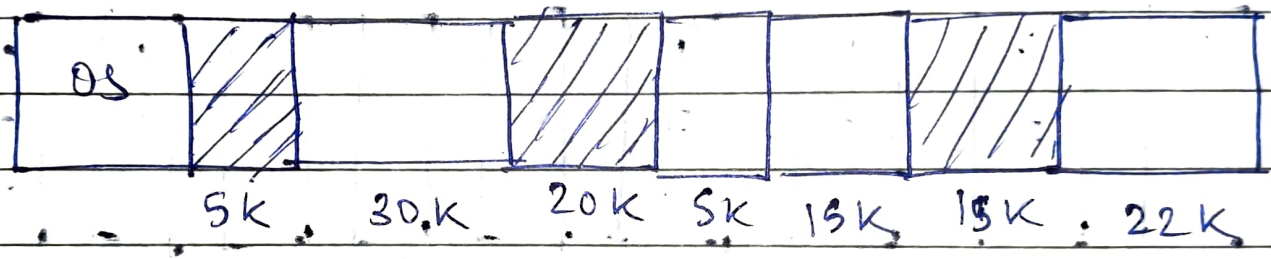
∴ In demand paging, only pages that are required at an instant of the time of execution are loaded.

Demand paging is same as paging system except that an entire process is not swapped in or swapped out. Rather a lazy swapper is used that loads only those pages that are needed.

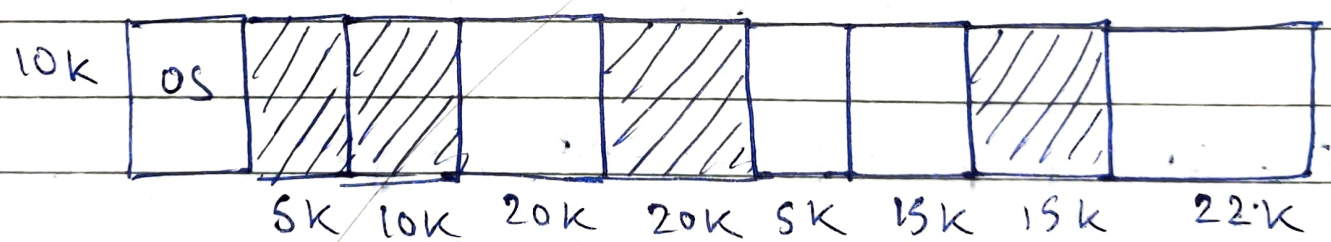
⇒ The swapping operations are hence known as page-in and page out.

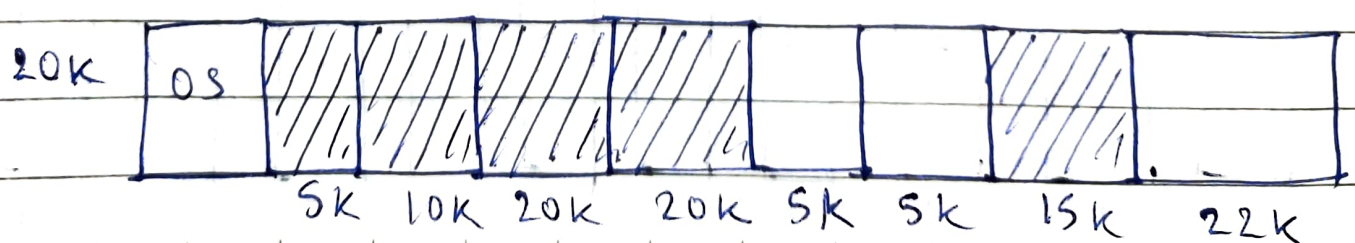
Assume the memory allocation scenario, as the following diagram and allocate memory additional request of 10K and 20K.

Compare the memory allocation using first fit and last fit.

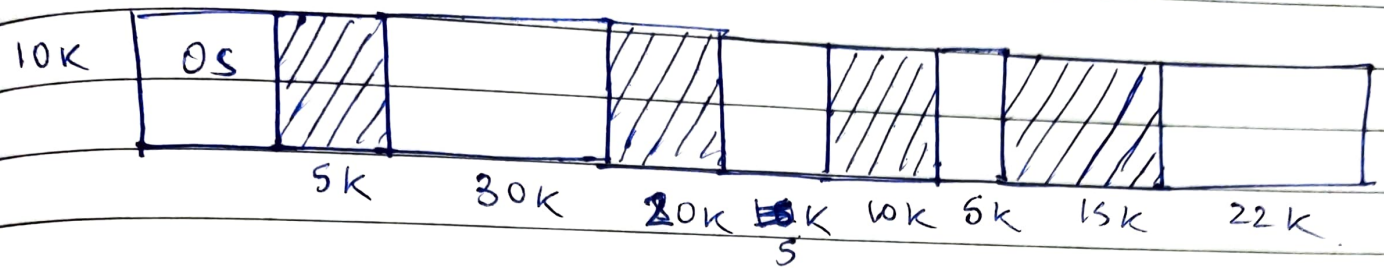


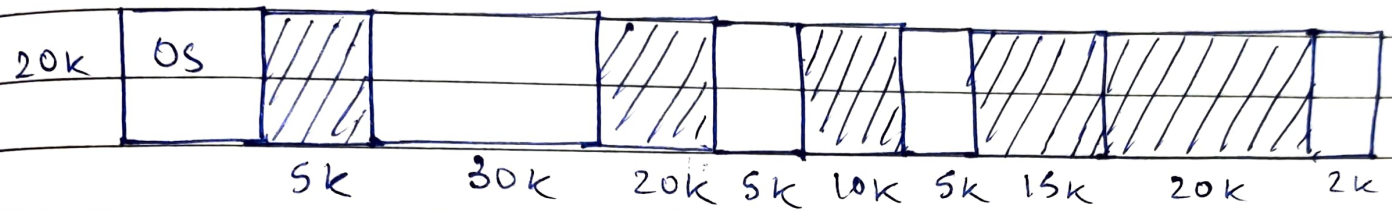
→ First fit



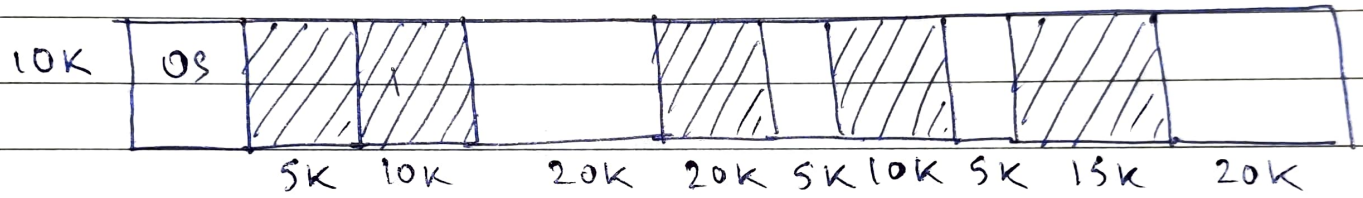


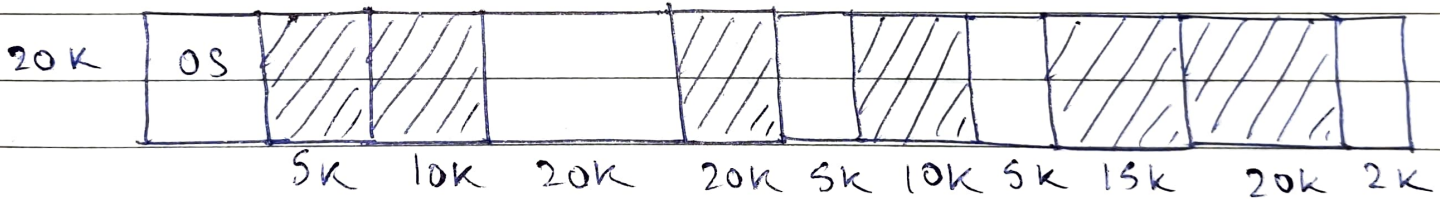
Best fit.


10K
OS
5K
30K
20K
5K
10K
5K
15K
22K


20K
OS
5K
30K
20K
5K
10K
5K
15K
20K
2K

worst fit


10K
OS
5K
10K
20K
20K
5K
10K
5K
15K
20K


20K
OS
5K
10K
20K
20K
5K
10K
5K
15K
20K
2K

Page Replacement Algo

It decides which memory page is to be replaced. The process of replacement is called swap out or write to disk.

Page replacement is done when the requested page is not found in the main memory (page fault)

TYPES

① FIFO algo.

If the req. is not available in the memory (page Fault (MISS)).

↳ else (page Hit)

eg

Frame size = 3

Ref string : ~~8 0 2 7~~ 0 3 0 2 4 3 0 3 2 1 3 0 1 5

Misses = x x x x x x x x x x x x x x x

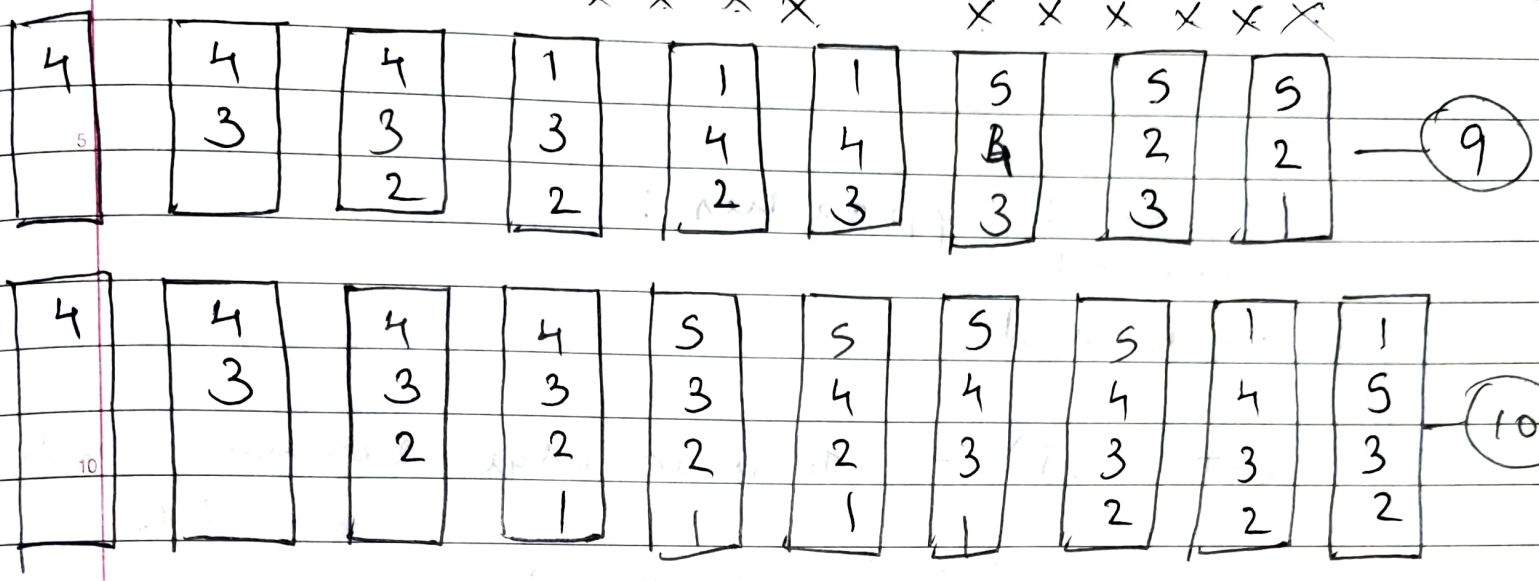
5	5	5	1	1	1	2	2	2
	0	0	0	3	3	3	4	4
		2	2	2	0	0	0	3

0	0	0	3	3	3	→ 15
4	2	2	2	0	0	
3	3	1	1	1	5	

Frame size : 3

Ref string : 4 3 2 1 4 3 5 4 3 2 1 5

Misses : x x x x x x x x x x x x



Frame size → increases → fault also increases

~~DISAD~~ → FIFO.

Belady's Anamaly

It reflects the fact that for some page replacement algo default rate may increase as the number of allocated frames increases.

OS

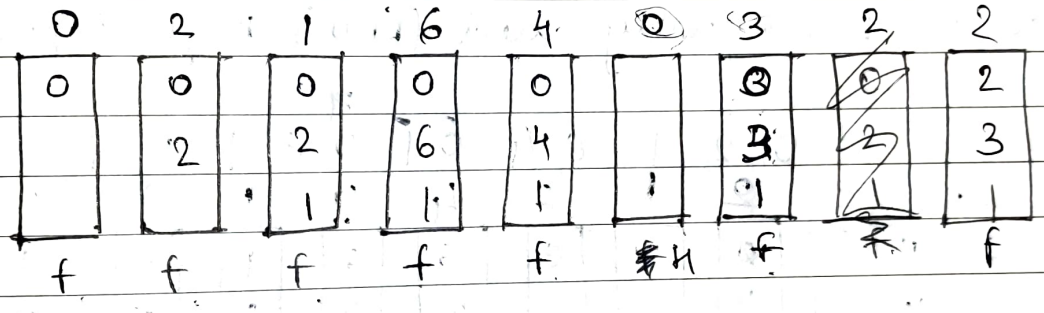
Types

2) Optimal Page Replacement Algo

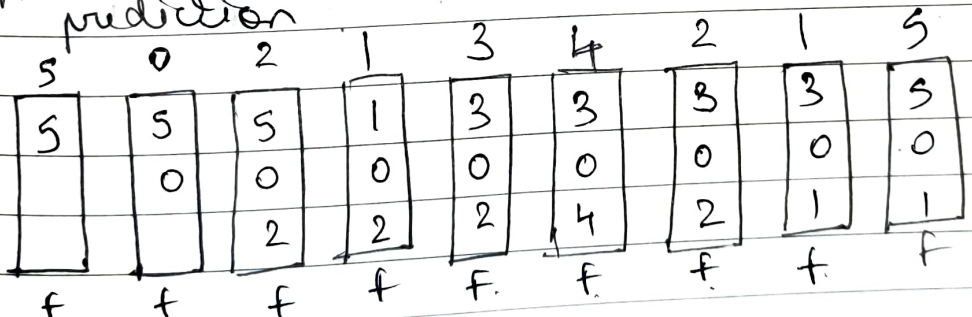
Ref string : 0 2 1 6 4 0 1 0 3 1 2 1
Frame size : 3
H H H H H

→ This algorithm replaces that pages which will not be used for the longest period of time in future

Ref str : 5 0 2 1 0 3 0 2 4 3 0 3 2 1 3 0 1 5
Frame size : 3
H H H H H H H H

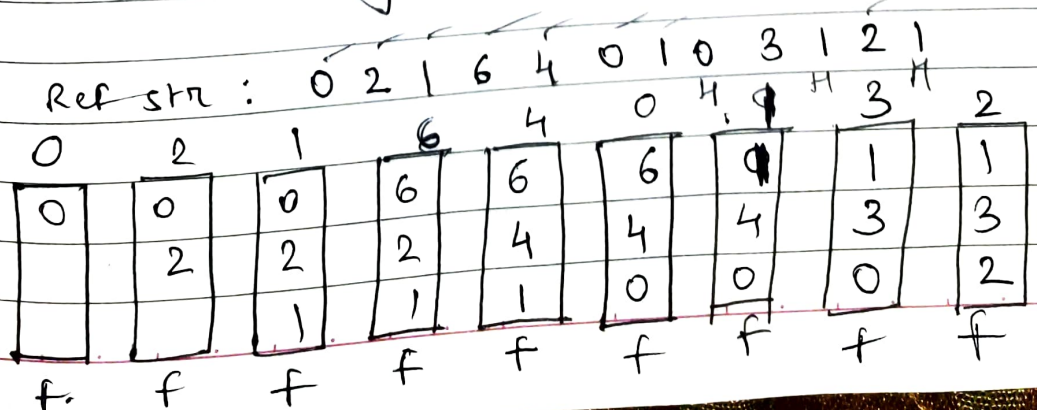


Practical implementation not possible → future prediction



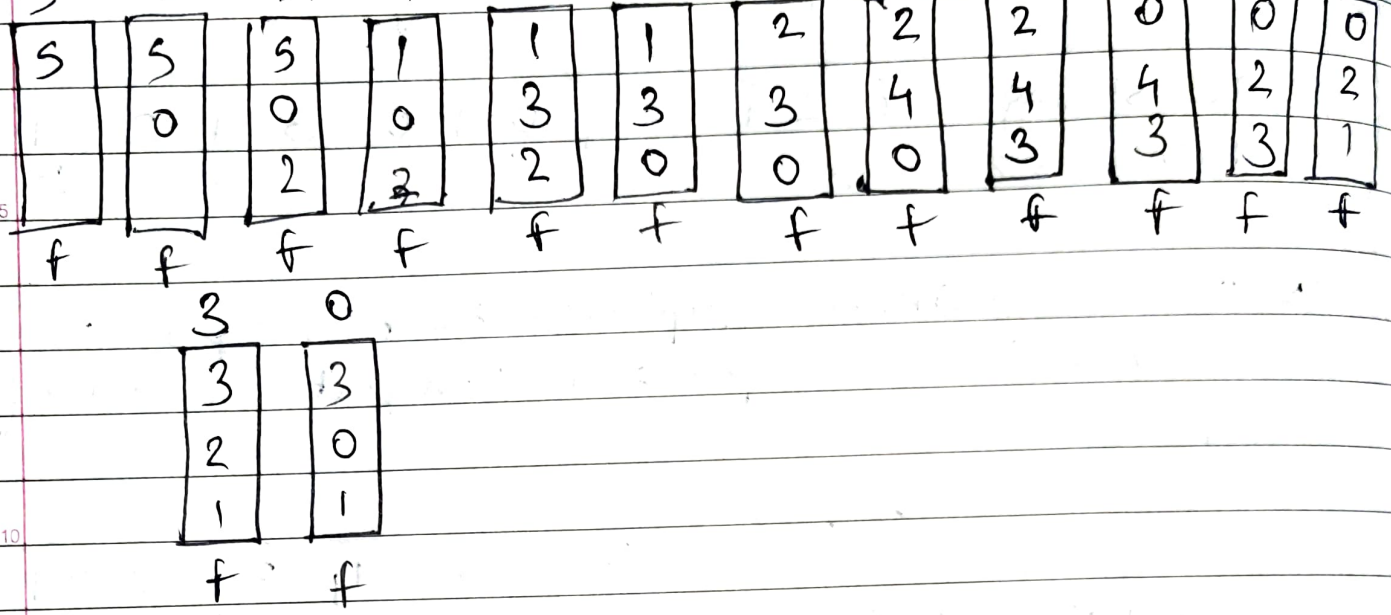
3) Least Recently used (LRU) Page Replacement algo

Ref str : 0 2 1 6 4 0 1 0 3 1 2 1
H H H H



Ref st : 5 0 2 1 0 3 0 2 4 3 0 3 2 1 3 0 1 5

H H H H H H



4> Least Frequently used (LFU) Page Replacement alg

H H H H

0 2 1 6 4 0 1 0 3 1 2 1

Count	0	2	1	6	4	0	1	3	2
0 - 11	0	2	0	6	6	6	1	1	1
1 - 111		2	2	2	4	4	4	3	2
2 - 11			1	1	1	0	0	0	0
3 - 1	f	f	f	f	f	f	f	f	f
4 - 1									
6 - 1									

A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO), Least Recently used (LRU) and optional page replacement policy.

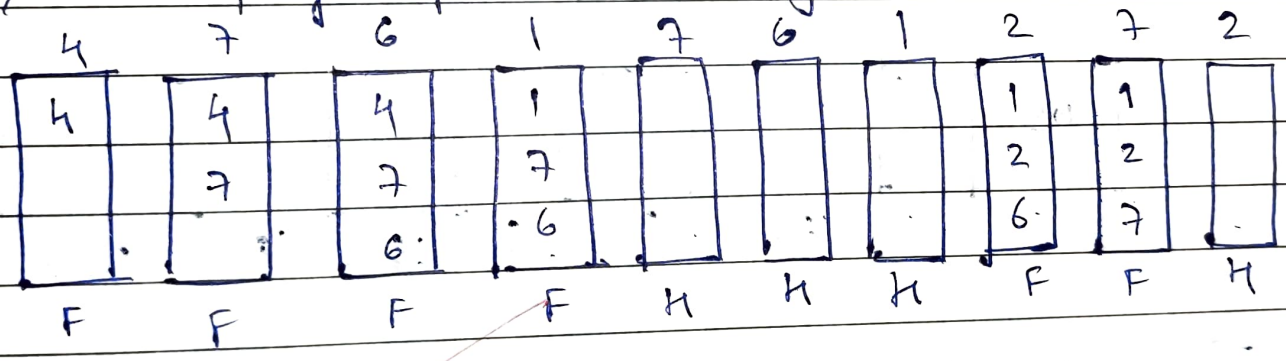
Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given. Also calculate hit ratio and miss ratio.

String given -

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

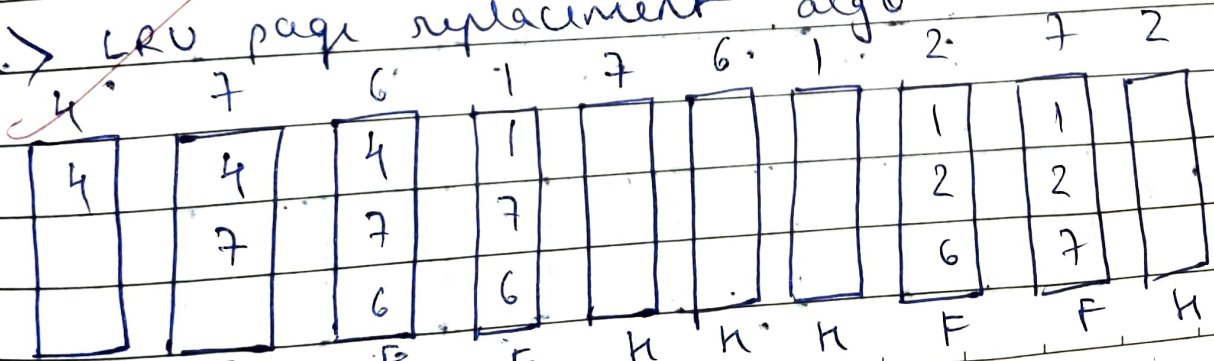
Frame size = 3

1) FIFO page replacement algo



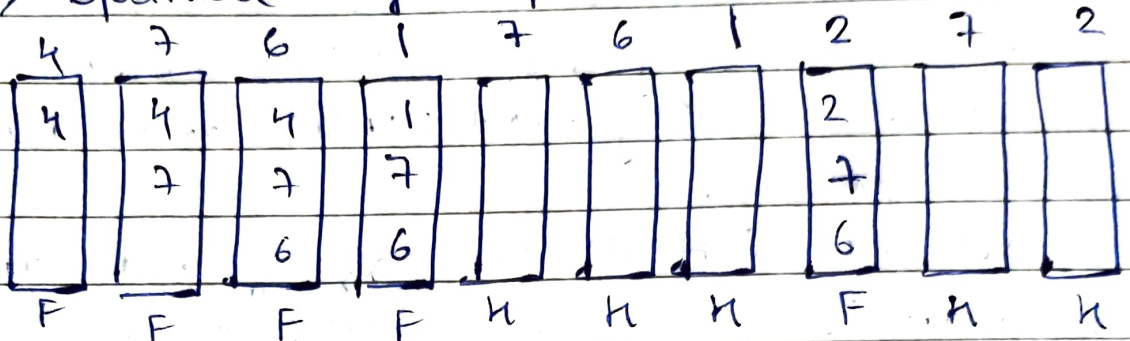
Hit ratio = 4/10 Miss ratio = 6/10

2) LRU page replacement algo



Hit ratio = 4/10 · Miss ratio = 6/10

3) Optimal Page Replacement



Miss ratio = 5/10 Hit ratio = 5/10

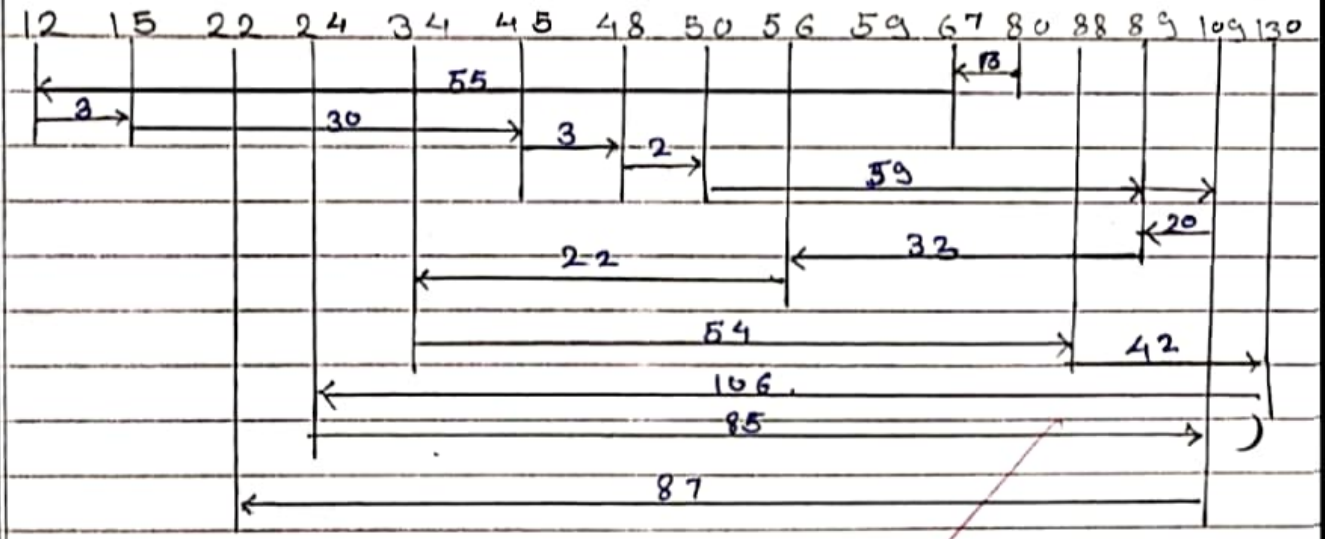
Q.1. Consider a disk queue with I/O requests on the following cylinders in their arriving order:

67, 12, 15, 45, 48, 50, 109, 89, 56, 59, 34, 88, 130, 24, 109, 22

The disk head is assumed to be at cylinder 80. The disk consists of total 150 cylinders. Calculate and show with diagram the disk head movement using FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK scheduling algorithm.

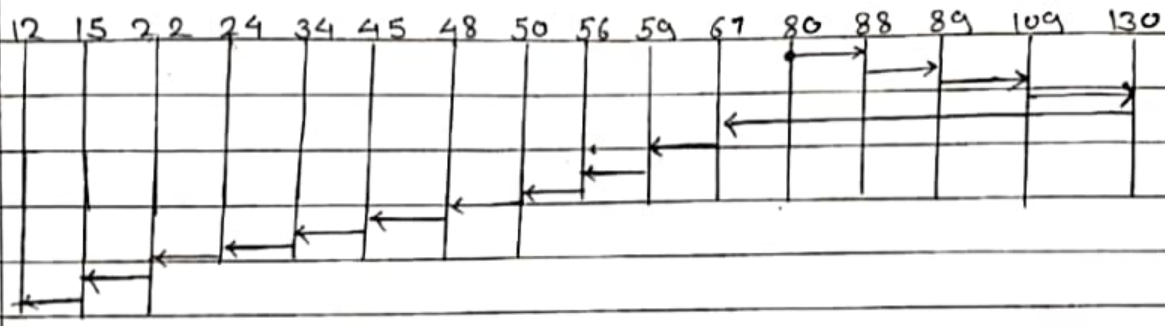
→

1) FCFS



Total head moment = 614.

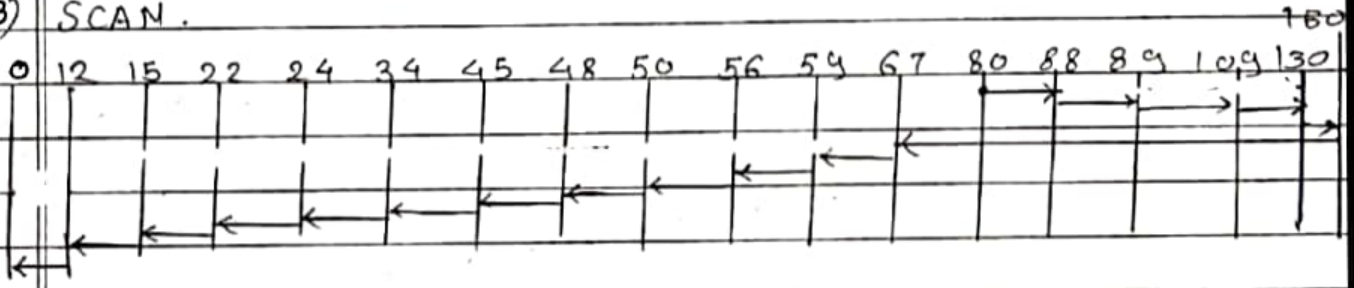
2) SSTF.



Total head moment = (130 - 80) + (130 - 12)

= 168.

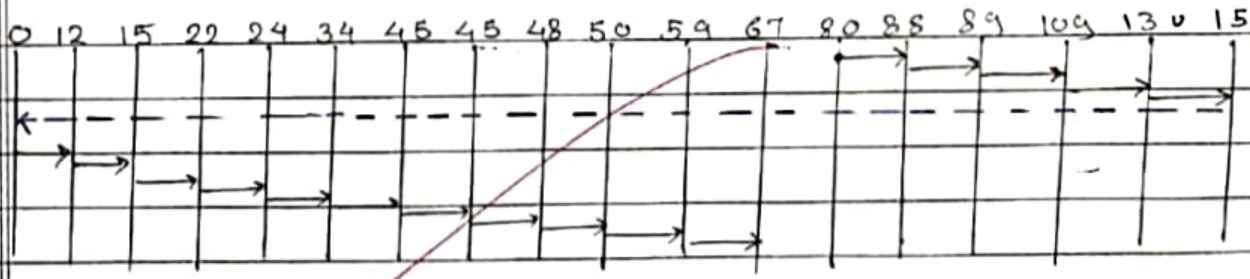
3) SCAN.



Total head moment = (150 - 80) + (150 - 0)

= 220.

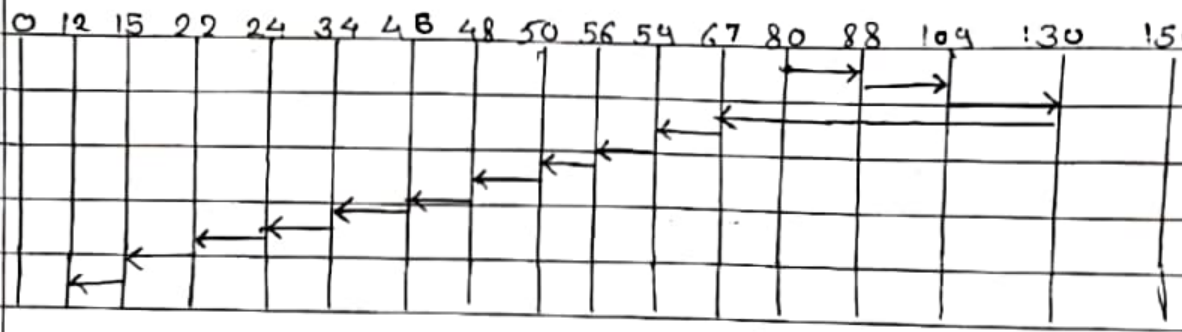
4) C-SCAN.



Total head moment = (80 - 150) + (67 - 0)

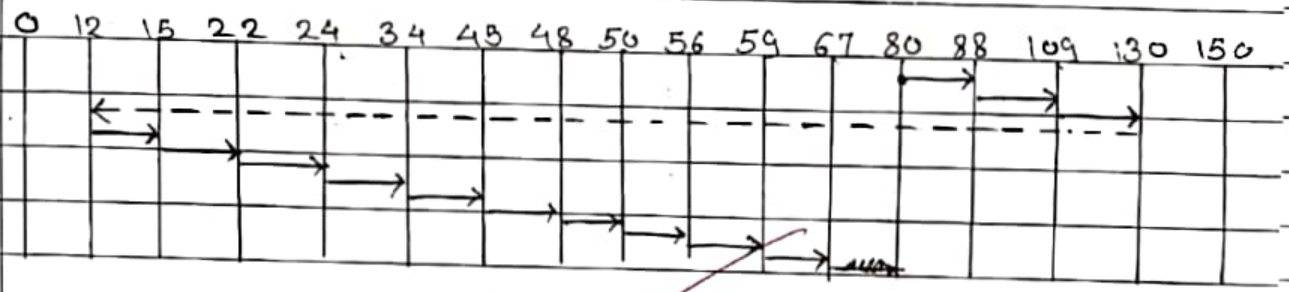
= 137.

5) Look.



Total head moment = (130 - 80) + (130 - 12)
= 168.

6) C-Look.



Total head moment = (130 - 80) + (12 - 67)
= 105.