Internal Assessment Test 2

Solution

Sub:	**Operating System - OS**					**Sub Code:**	**BCS303**	**Branch:**	**CSE**
Date:	**06/01/2026**	**Duration:**	**90 mins**	**Max Marks:**	**50**	**Sem / Sec:**	**3 A,B,C**		

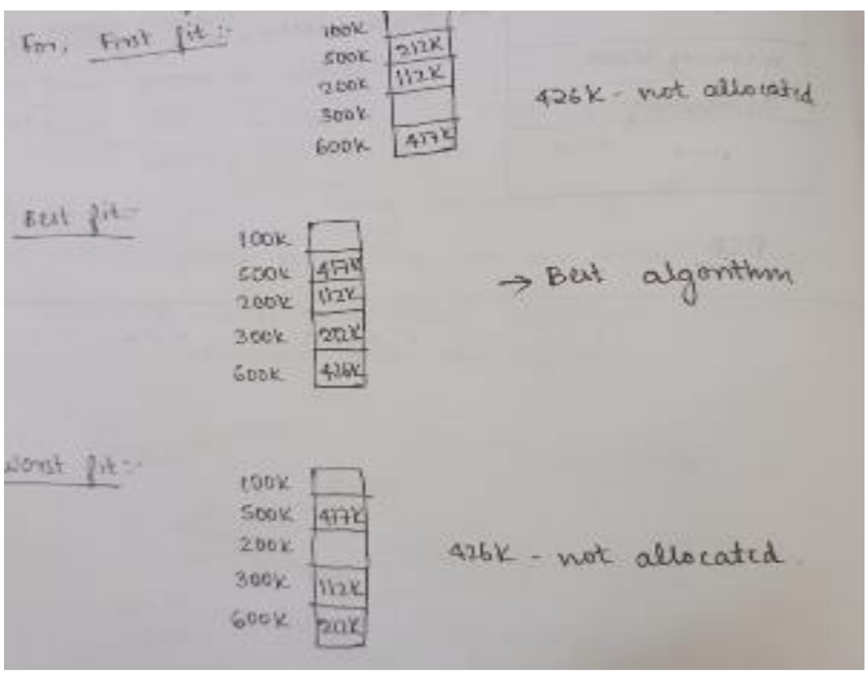
1. A **Given the memory partitions of 100K,500K,200K,300K and 600K. Apply first fit, best fit and worst fit algorithms to place 212K, 417K, 112K and 426K.**

Solution

Given:

Memory Partitions:
100K, 500K, 200K, 300K, 600K

Process Sizes:
212K, 417K, 112K, 426K



1. **First Fit Algorithm -** Allocate first block that is large enough

   Allocated: 212K, 417K, 112K

   Not allocated: 426K

2. **Best Fit Algorithm -** Allocate smallest block that fits

   Allocated: All processes allocated successfully

3. **Worst Fit Algorithm -** Allocate first block that is large enough

   Allocated: 212K, 417K, 112K

   Not allocated: 426K

1.B

Explain Internal and External fragmentation with examples.

Solution:

Internal Fragmentation

Occurs **inside allocated memory**
Memory allocated > memory required
Example: Process needs 90K, block allocated is 100K → 10K wasted

External Fragmentation

Occurs between allocated memory blocks
Free memory exists but not contiguous
Example:

Free blocks: 50K + 40K + 60K

Request: 120K → Cannot allocate

2.A

Explain Access Matrix Protection system of OS

Solution:

- Protection system that defines **which process can access which resource**
- Represented as a **matrix**

Structure:

Rows → Subjects (Users / Processes)
Columns → Objects (Files, Devices)
Entries → Access rights (Read, Write, Execute)

Example:

	File1	File2
P1	R,W	R
P2	R	—

Advantages:

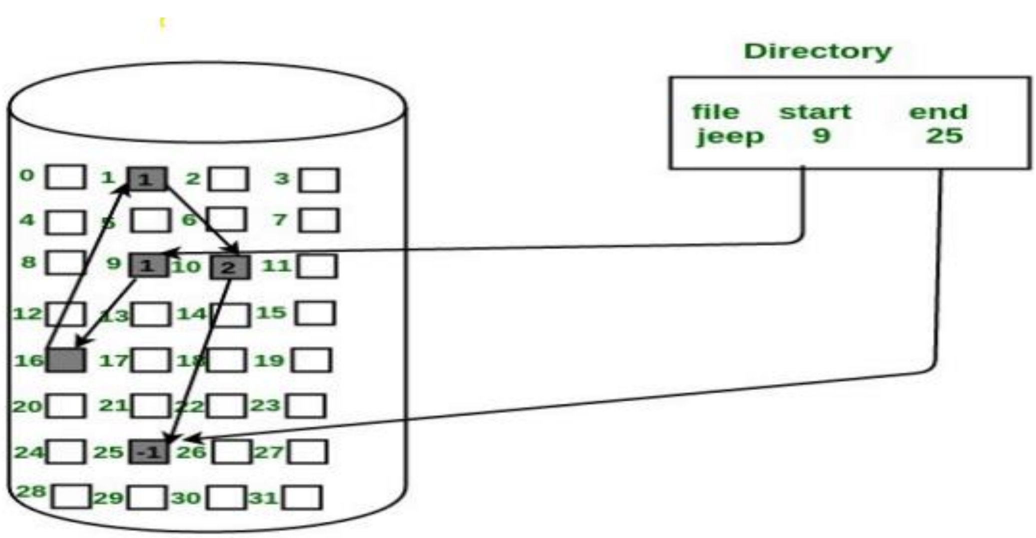
- Flexible
- Fine-grained protection

2. B

With a suitable example, explain any one File Allocation method.

Solution:

Linked File Allocation



In Linked List Allocation, each file is a linked list of disk blocks that need not be contiguous. The disk blocks can be scattered anywhere on the disk. It is easy to

	allocate the files because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consists of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation. We can avoid the external fragmentation. In linked allocation, each file is a linked list of disk blocks. The directory contains a pointer to the first and optionally the last block of the file. For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27. Each block contains a pointer to the next block and the last block contains a NIL pointer. The value -1 may be used for NIL to differentiate it from block 0.
3 a	**What is thrashing? How can it be controlled? Explain any one method.** Thrashing is the state of a process where there is high paging activity. A process that is spending more time paging than executing is said to be thrashing. **Cause of Thrashing** When memory is filled up and processes starts spending lots of time waiting for their pages to page in, then CPU utilization decreases(Processes are not executed as they are waiting for some pages), causing the scheduler to add in even more processes and increase the degree of multiprogramming even more. Thrashing has occurred, and system throughput plunges. No work is getting done, because the processes are spending all their time paging. In the graph given below , CPU utilization is plotted against the degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.

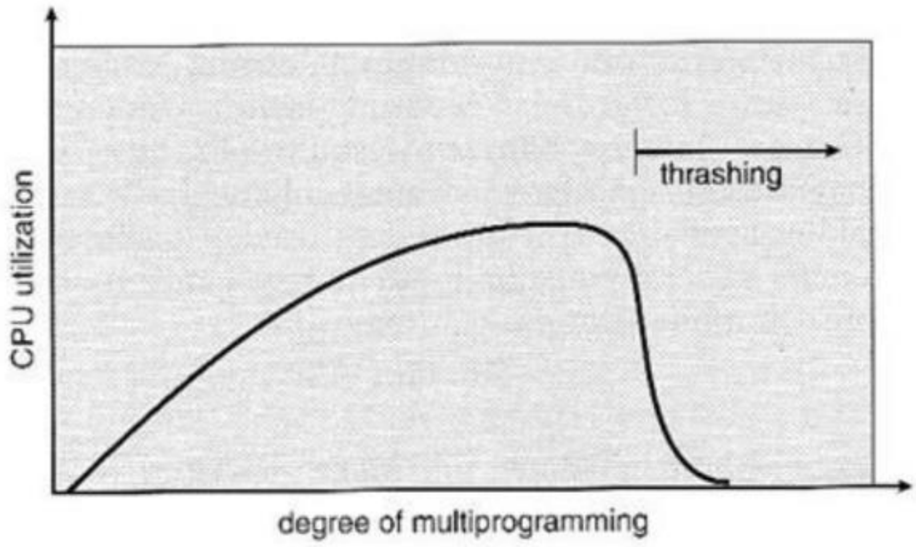


Figure 9.18 Thrashing.

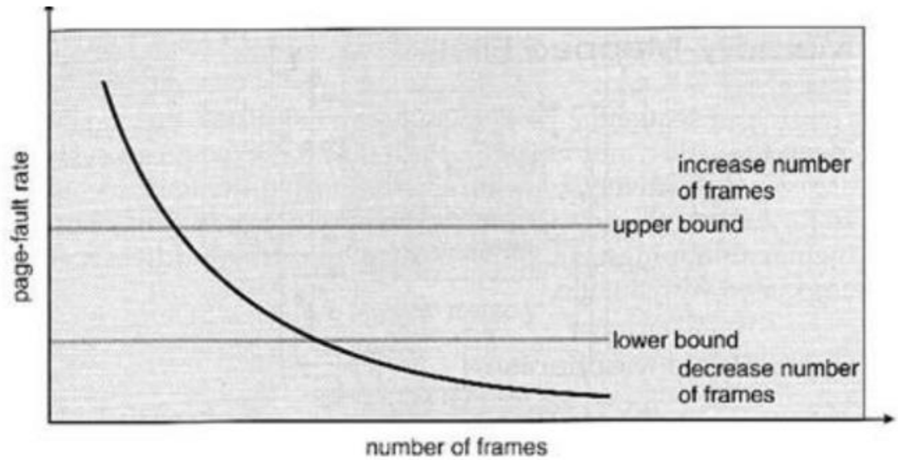
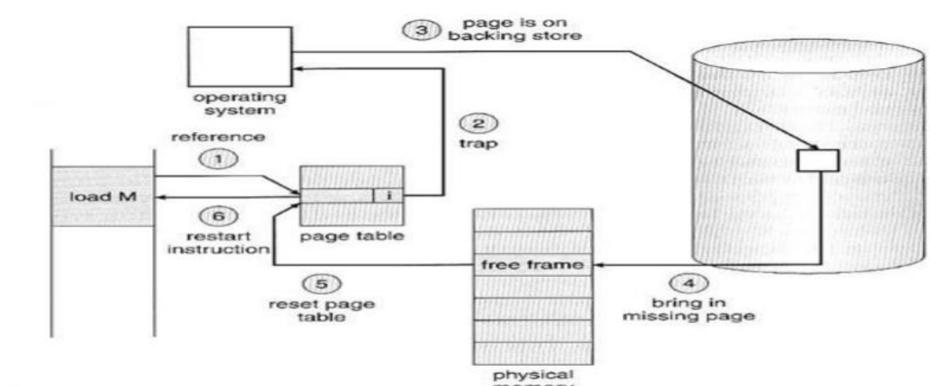
Solution:-

- Local page replacement policies can prevent thrashing process from taking pages away from other processes, but it still tends to clog up the I/O queue.
- Working-Set Model
- Page-Fault Frequency

Page-Fault Frequency

When page- fault rate is too high, the process needs more frames and when it is too low, the process may have too many frames.

The upper and lower bounds can be established on the page-fault rate. If the actual page-fault rate exceeds the upper limit, allocate the process another frame or suspend the process. If the page-fault rate falls below the lower limit, remove a frame from the process. Thus, we can directly measure and control the page-fault rate to prevent thrashing.

	 **Figure 9.21** Page-fault frequency.
3b	**What is demand paging? Explain with a neat diagram step to handle page fault.** The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them. (on demand.) This is termed as lazy swapper, although a pager is a more accurate term.  **Figure 9.6** Steps in handling a page fault. **On the other hand, if a page is needed that was not originally loaded up, then**

	a page fault trap is generated, which must be handled in a series of steps: 1. The memory address requested is first checked, to make sure it was a valid memory request. 2. If the reference is to an invalid page, the process is terminated. Otherwise, if the page is not present in memory, it must be paged in. 3. A free frame is located, possibly from a free-frame list. 4. A disk operation is scheduled to bring in the necessary page from disk. 5. After the page is loaded to memory, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference. 6. The instruction that caused the page fault must now be restarted from the beginning.
4 a	**What is the Critical section problem? What are the requirements for the solution to critical section problems? Explain the solution using Semaphores.** Consider a system consisting of n processes {P0, P1, ..., Pn−1}. Each process has a segment of code, called a critical section, in which the process may be changing common variables,updating a table, writing a file, and so on. when one process is executing in its critical section, no other process is allowed to execute in its critical section. The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical

section may be followed by an exit section. The remaining code is the remainder section. The general structure of a typical process Pi is shown in Figure

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (true);
```

General structure of a typical process P_i.

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion. If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, **and this selection cannot be postponed indefinitely.**

3. Bounded waiting. There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a **request to enter its critical section and before that request is granted.**

The hardware-based solutions to the critical-section problem are complicated as well as generally inaccessible to application programmers. So operating-systems designers build software tools to solve the critical-section problem, and this synchronization tool called as Semaphore**.**

Semaphore S is an integer variable

Two standard operations modify S: wait() and signal()

Originally called P() and V()

Can only be accessed via two indivisible (atomic) operations

Must guarantee that no two processes can execute wait () and signal () on the same semaphore at the same time.

```
•  wait (S) {
      while S <= 0
                 ; // no-op
        S--;
   }
•  signal (S) {
     S++;
   }
```

(b) Explain Segmentation in detail.

- Logical address consists of a two tuple:
    - <segment-number, offset>,
- Segment table – maps two-dimensional physical addresses; each table entry has:
- base – contains the starting physical address where the segments reside in

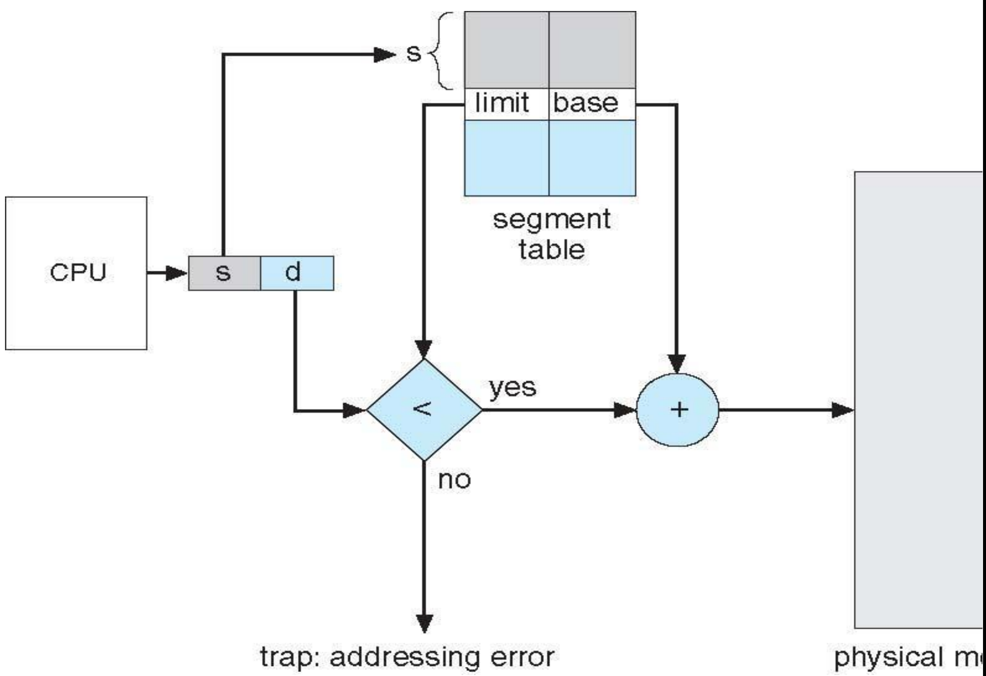
memory

- limit – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;

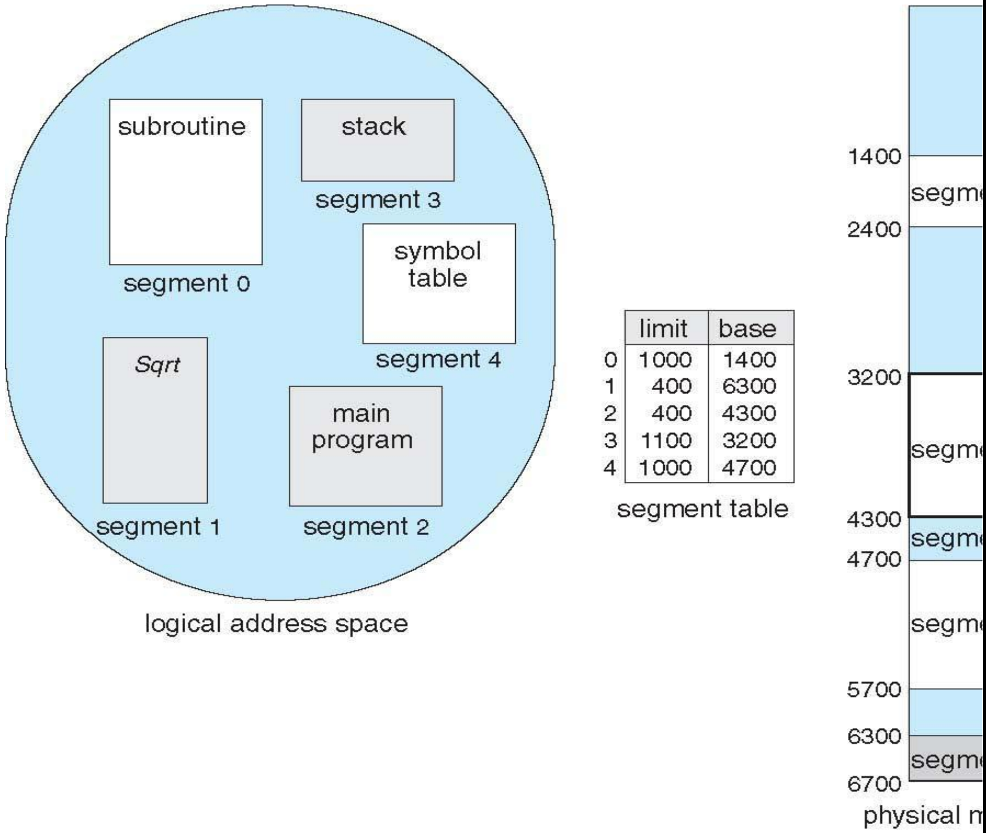
segment number s is legal if s < STLR

- Protection With each entry in segment table associate: validation bit = 0 ⇒illegal segment read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

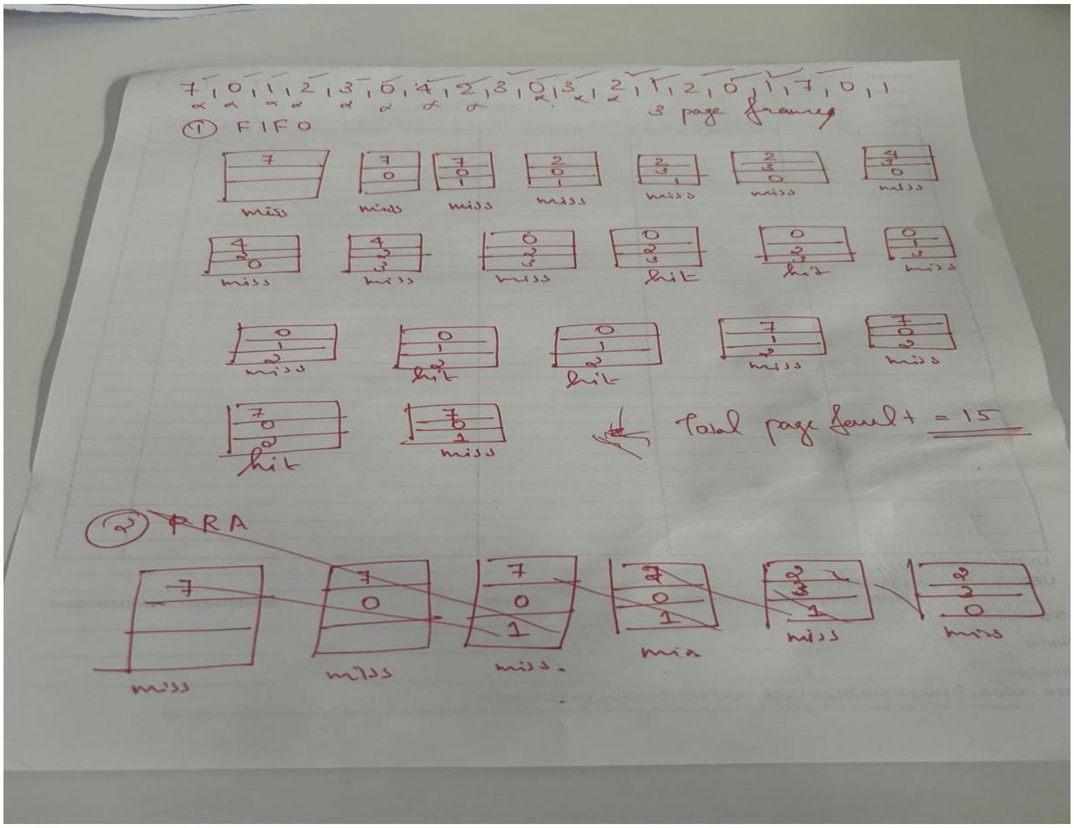
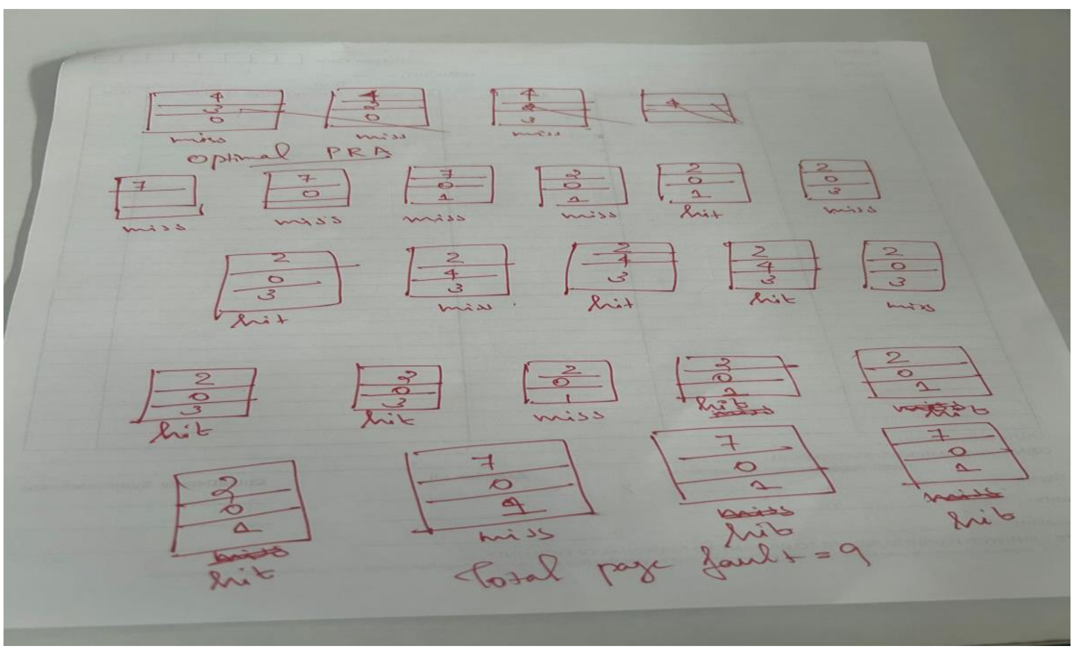
- A segmentation example is shown in the following diagram



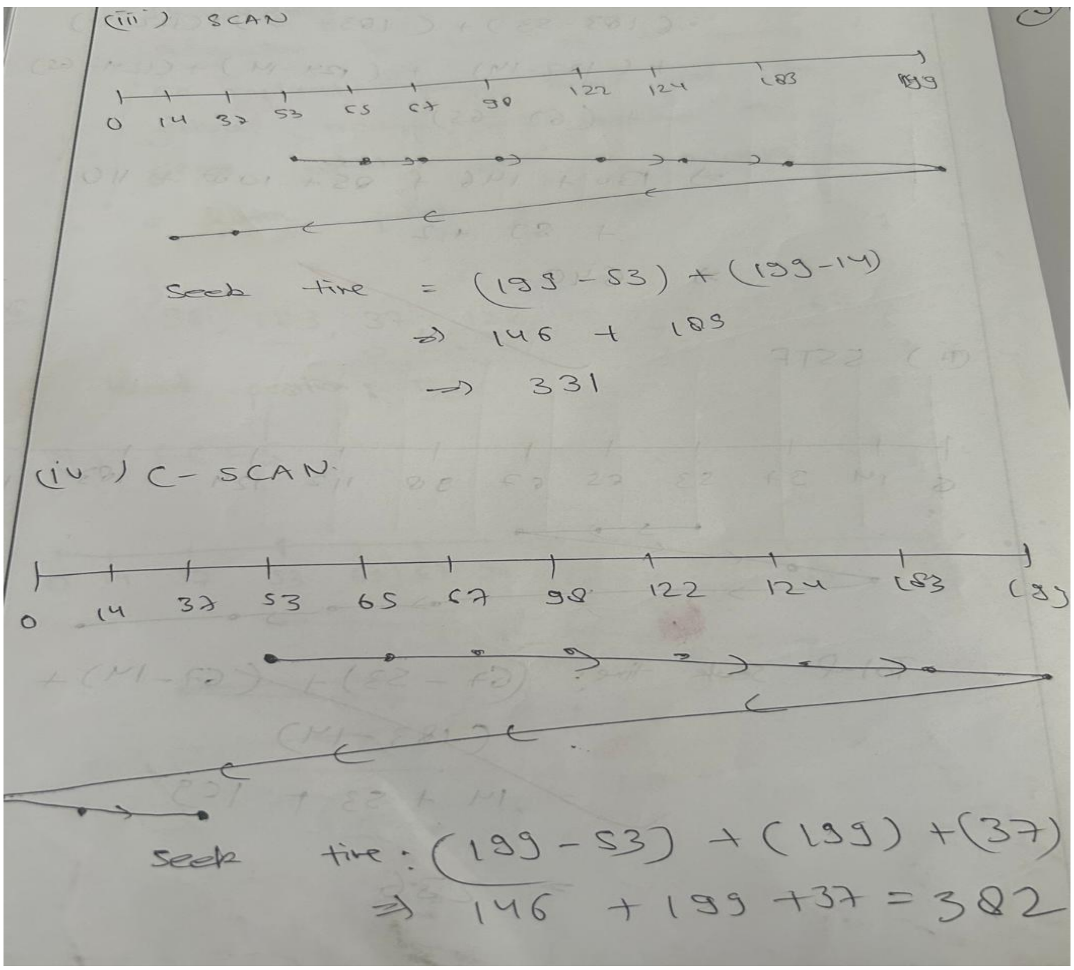
-

<table>
<tr>
<td></td>
<td>


●
</td>
</tr>
<tr>
<td>5</td>
<td>Consider the following Sequence
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
How many Page faults occur with 3 Page frames
(1) FIFO
(2) Optimal Page Replacement Algorithm
Which Algorithm is efficient and why?</td>
</tr>
</table>

	  6.Consider the input sequence

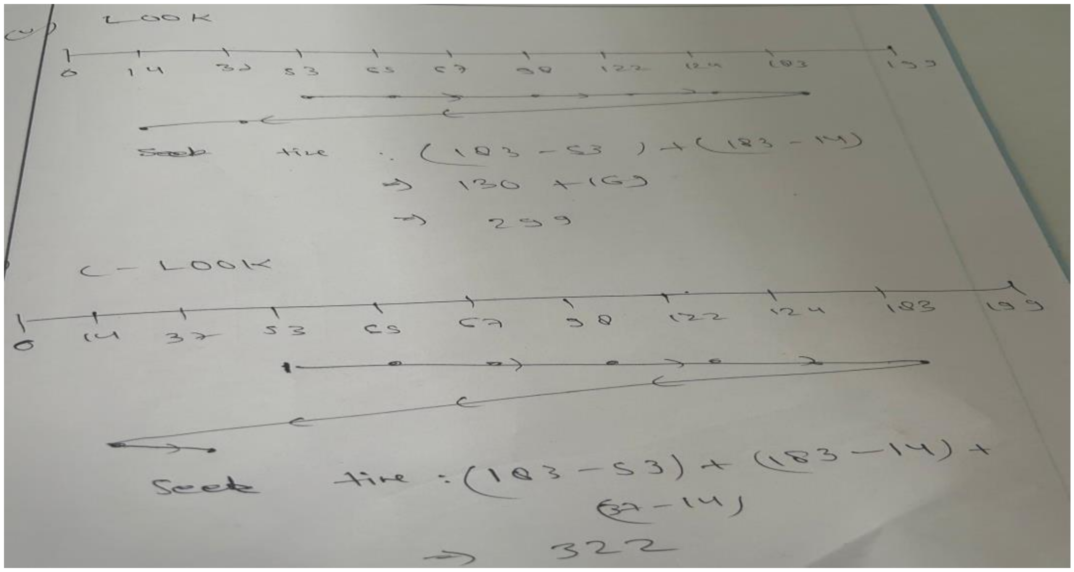
98,183,37,122,14,124,65,67
Head Position at 53
Apply Disk Scheduling Algorithm and find seek time
(1) FCFC Scheduling
(2) SSTF Scheduling
(3) SCAN Scheduling
(4)
C
-SCAN Scheduling
(5) LOOK Scheduling
(6)
C
-LOOK Scheduling

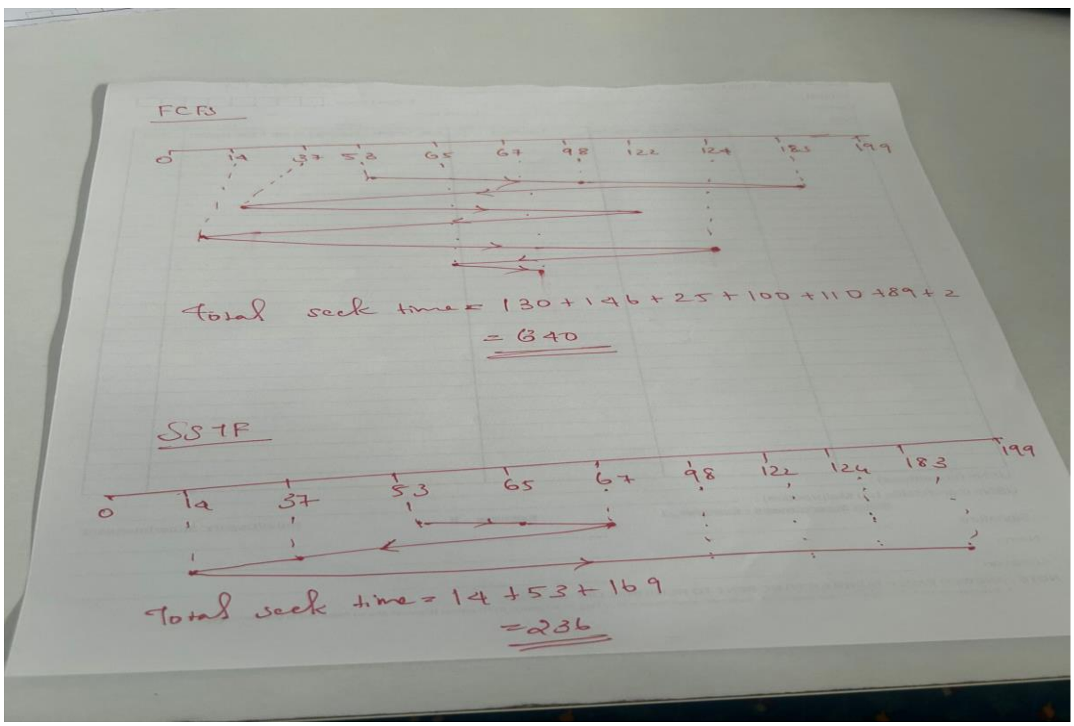

(iii) SCAN

Seek time = (199 − 53) + (199 − 14)
⇒ 146 + 185
⇒ 331

(iv) C-SCAN

Seek time : (199 − 53) + (199) + (37)
⇒ 146 + 199 + 37 = 382


LOOK
0 14 37 53 65 67 98 122 124 183 199
Seek time : (183 − 53) + (183 − 14)
→ 130 + 169
→ 299
C-LOOK
0 14 37 53 65 67 98 122 124 183 199
Seek time : (183 − 53) + (183 − 14) + (37 − 14)
→ 322


FCFS
0 14 37 53 65 67 98 122 124 183 199
Total seek time = 130 + 146 + 25 + 100 + 110 + 89 + 2
= 640
SSTF
0 14 37 53 65 67 98 122 124 183 199
Total seek time = 14 + 53 + 169
= 236