Q3 (a)Selecting correct option 01 mark

The occurrence of following scenes may cause deadlock-
Explanation of ME=02 marks
Explanation of Deadlock=02 marks

Scene-01:

- Process P arrives.
- It executes wait operation on semaphore S1 successfully.
- Process P gets preempted.

Scene-02:

- Process Q gets scheduled.
- It executes wait operation on semaphore S2 successfully.
- Process Q gets preempted.

Scene-03:

- Process P gets scheduled again.
- It executes wait operation on semaphore S2 unsuccessfully and gets blocked.
- Process P gets preempted.

Scene-04:

- Process Q gets scheduled again.
- It executes wait operation on semaphore S1 unsuccessfully and gets blocked.

Now, Both the processes are blocked and keeps waiting for the signal from the each other.

- The system is in a deadlock state.
- Also, mutual exclusion can be guaranteed.
- Thus, Option (C) is correct.

Q3(b)

One solution of this problem is to use semaphores. The semaphores which will be used here are:

- `m`, a **binary semaphore** which is used to acquire and release the lock.
- `empty`, a **counting semaphore** whose initial value is the number of slots in the buffer, since, initially all slots are empty.
- `full`, a **counting semaphore** whose initial value is `0`.(initialization with meaning=01 mark)

At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

The Producer Operation (02 marks=code and explanation)

The pseudocode of the producer function looks like this:

```
do
{
    // wait until empty > 0 and then decrement 'empty'
    wait(empty);
    // acquire lock
    wait(mutex);

    /* perform the insert operation in a slot */

    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
```

```
while(TRUE)
```

- Looking at the above code for a producer, we can see that a producer first waits until there is atleast one empty slot.
- Then it decrements the **empty** semaphore because, there will now be one less empty slot, since the producer is going to insert data in one of those slots.
- Then, it acquires lock on the buffer, so that the consumer cannot access the buffer until producer completes its operation.
- After performing the insert operation, the lock is released and the value of **full** is incremented because the producer has just filled a slot in the buffer.

The Consumer Operation (02 marks=code and explanation)

The pseudocode for the consumer function looks like this:

```
do
{
    // wait until full > 0 and then decrement 'full'
    wait(full);
    // acquire the lock
    wait(mutex);

    /* perform the remove operation in a slot */

    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

- The consumer waits until there is atleast one full slot in the buffer.
- Then it decrements the **full** semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.
- After that, the consumer acquires lock on the buffer.
- Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.
- Then, the consumer releases the lock.
- Finally, the **empty** semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.

Q3(b)

Problem satatement=01 mark, Semaphore initialization=01 mark, pseudocode for each philosopher=01 marks, explanation=02 marks

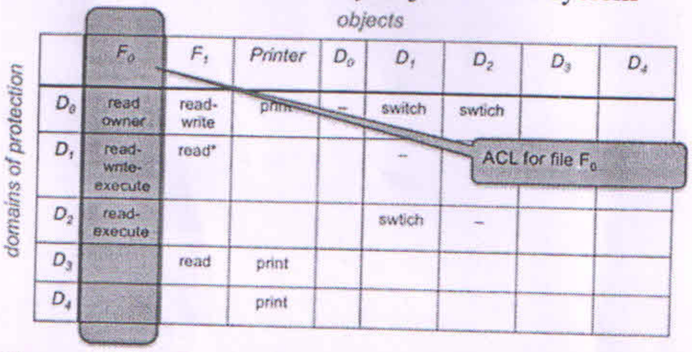
```
process P[i]
while true do
{  THINK;
   PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
   EAT;
   PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
}
```

Q4(b)

Basis for Comparison	Paging	Segmentation
Basic	A page is of fixed block size.	A segment is of variable size.

Basis for Comparison	**Paging**	**Segmentation**
Fragmentation	Paging may lead to internal fragmentation.	Segmentation may lead to external fragmentation.
Address	The user specified address is divided by CPU into a page number and offset.	The user specifies each address by two quantities a segment number and the offset (Segment limit).
Size	The hardware decides the page size.	The segment size is specified by the user.
Table	Paging involves a page table that contains base address of each page.	Segmentation involves the segment table that contains segment number and offset (segment length).

Any 4 points. Each point carry 1 mark
OR
Page fault definition=01 mark
Diagram=01 mark
Explanation=02 marks
In computer science, an **Access Control Matrix** or **Access Matrix** is an abstract, formal security model of protection state in computer systems, that characterizes the rights of each subject with respect to every object in the system



Virus, worms, Trojan, spyware, rootkit, logic bomb, botnets, spam
Ans: List down types=01 mark, working of each type=04 marks (01 mark for each type)

Process	AT	BT	TAT	WT
P_0	0	5	13	8
P_1	1	3	11	8
P_2	2	1	3	2
P_3	3	2	6	4
P_4	4	3	7	4

TQ = 2

Queue ~~P_0 P_1 P_2 P_0 P_3 P_4 P_1 P_0 P_4~~ (01 Mark)

Gantt chart

P_0	P_1	P_2	P_0	P_3	P_4	P_1	P_0	P_4

0 2 4 5 7 9 11 12 13 14 (02 Marks)

Avg TAT = (13+11+3+6+7)/5 = <u>8</u> (01 Mark)

Avg WT = (8+8+2+4+4)/5 = <u>5.2</u> (01 Mark)

<u>OR</u>

<u>Q.2(a)</u>

PNO	AT	Priority	BT	CT	TAT	WT
P_1	0	1	3	10	10	7
P_2	1	5	5	8	7	2
P_3	2	7 (H)	2	4	2	0
P_4	3	2	5	15	12	7
P_5	4	1 (L)	5	20	16	11

Gantt chart:

P_1	P_2	P_3	P_2	P_1	P_4	P_5

0 1 2 4 8 10 15 20 (02 Marks)

P_2 P_3 P_4, P_5

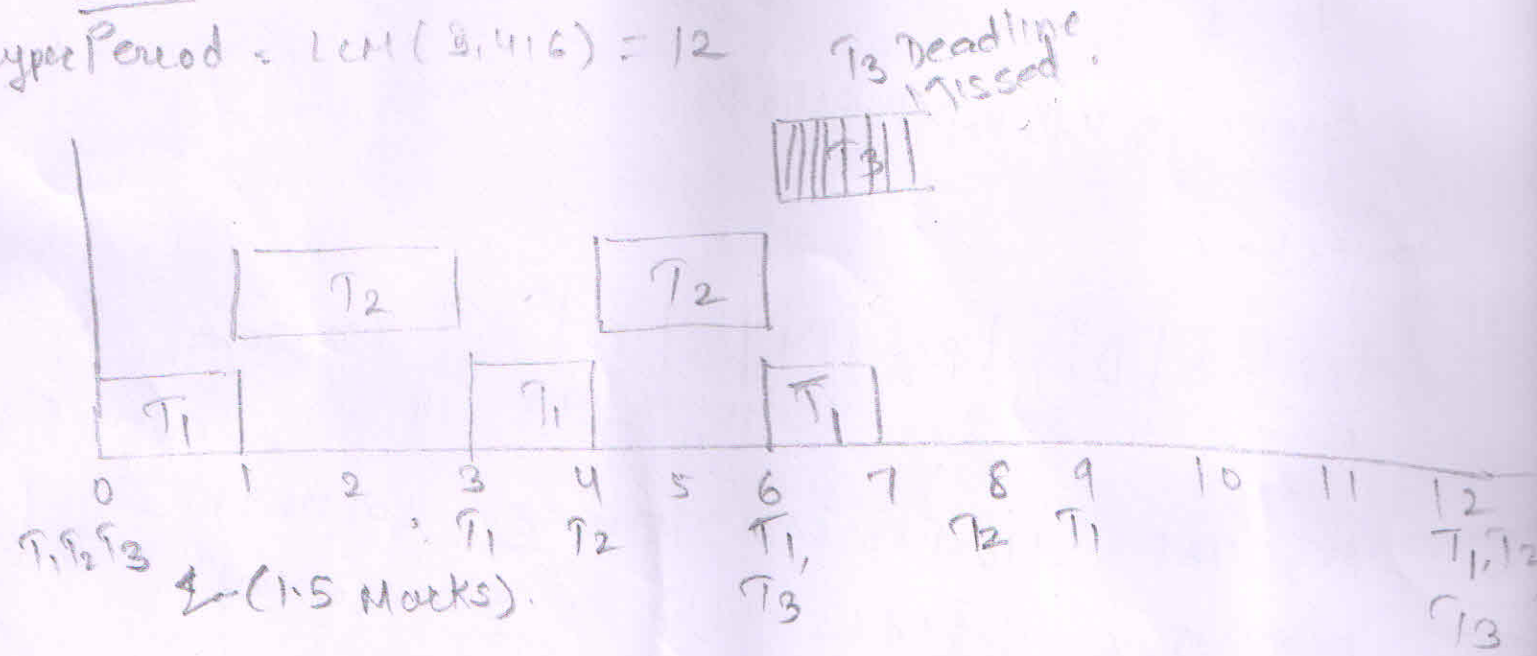
Avg TAT = 9.4 (02 Mark) Avg WT = 5.4 (01 Mark).

Q(2)(b)

Task	ST	P/D	
T_1	1	3	1
T_2	2	4	2
T_3	1	6	3

RMS

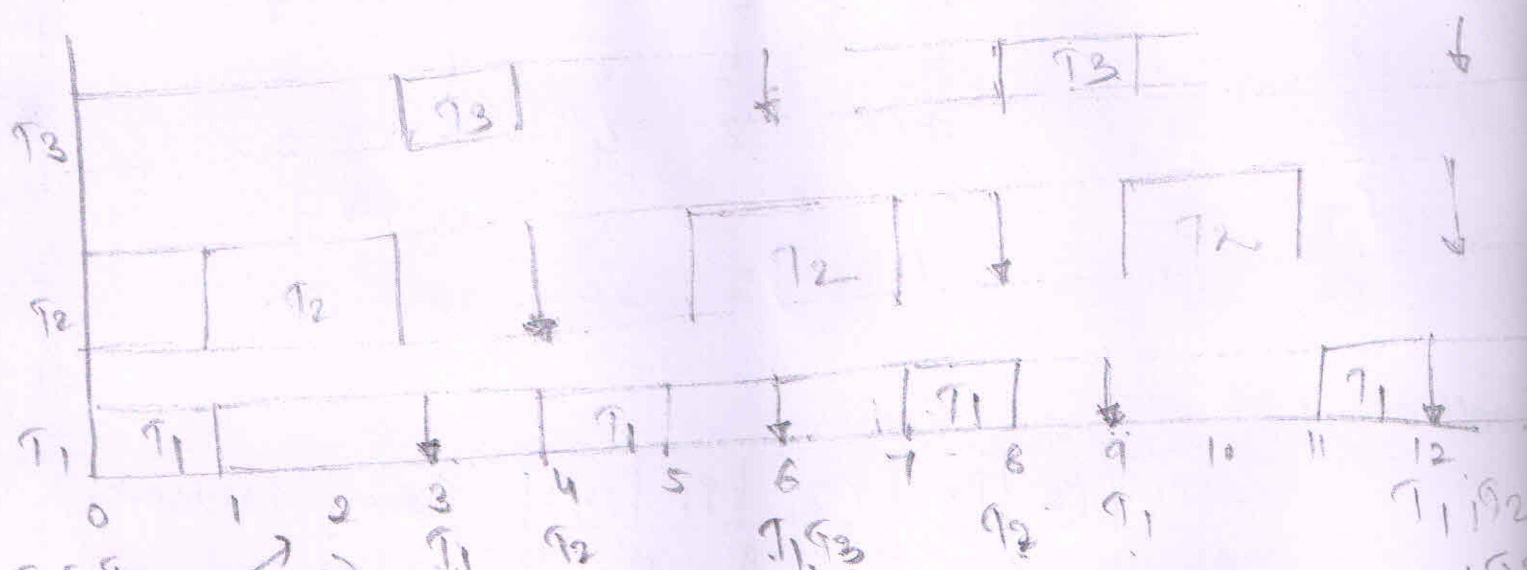
Hyper Period = LCM (3, 4, 6) = 12



(1.5 Marks).

With RMS – Given set of Tasks are Not Schedulable (01 Mark).

EDF



(1.5 Marks) Schedulable using EDF (01 Mark)

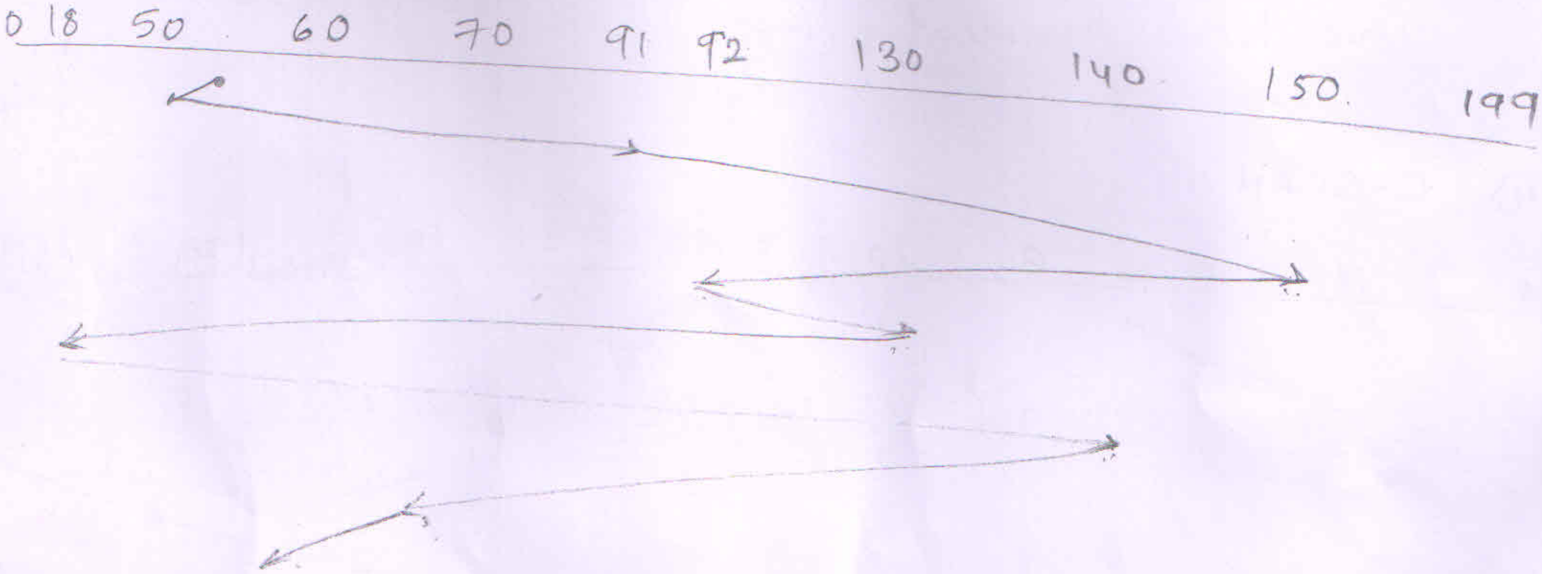
…nders = 0 to 199

…ueue = 50, 91, 150, 92, 130, 18, 140, 70, 60

Current head Position = 53

Previous request = 65

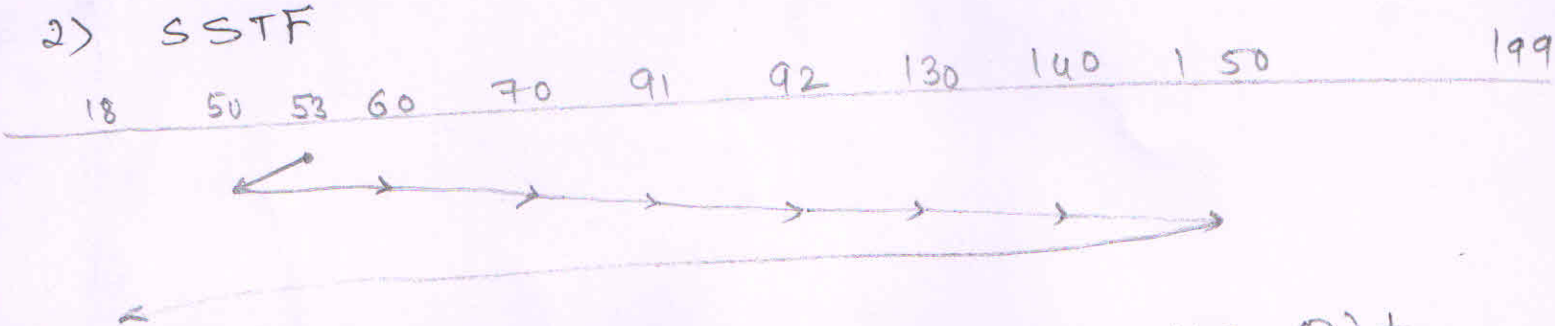
1) FCFS :- (01 Mark for each Algorithm)



Total head Movement = (53 − 50) + (91 − 50) + (150 − 91) + (150 − 92) + (130 − 92) + (130 − 18) + (140 − 18) + (140 − 70) + (70 − 60) = 3 + 41 + 59 + 58 + 38 + 112 + 70 + 1

= 391 + 122 = 513.

2) SSTF

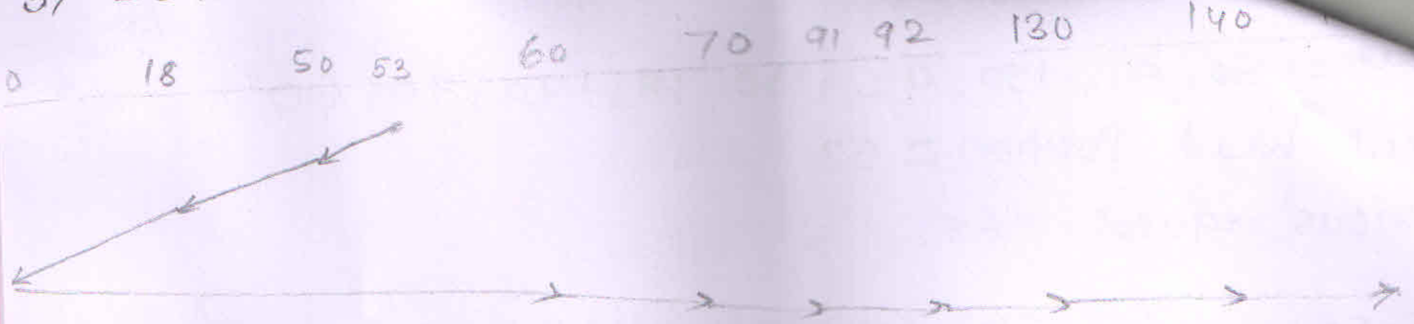


Total head Movement = (53 − 50) + (150 − 50) + (150 − 18)

= 3 + 100 + 132

= 235

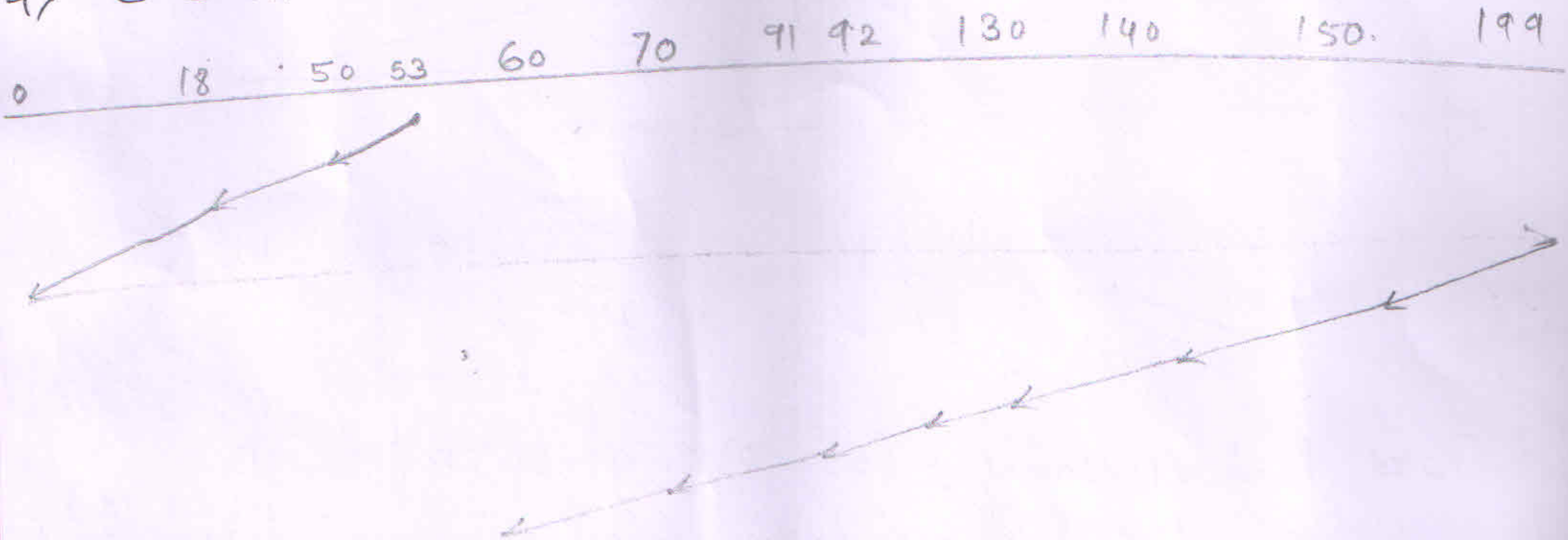
3> SCAN :-



Total head Movement = (53-0) + (150-0) = 53+150

= 203.

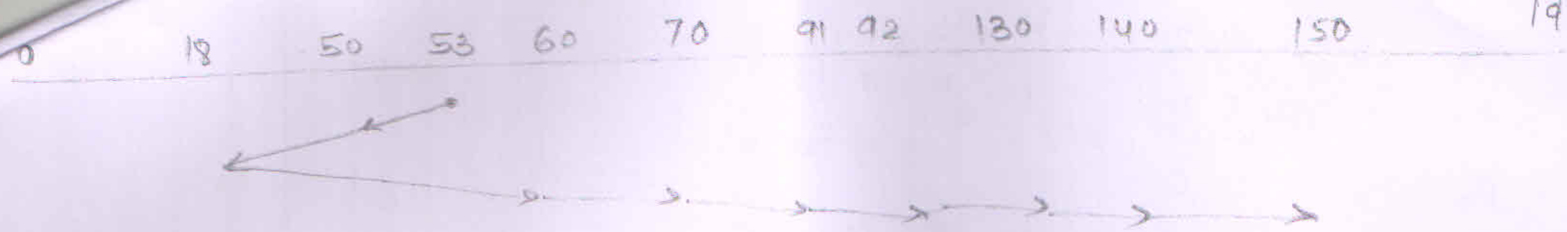
4> C-SCAN :-



Total head Movement = (53-0) + (199-0) + (199-60)

= 53+199+139

= 391

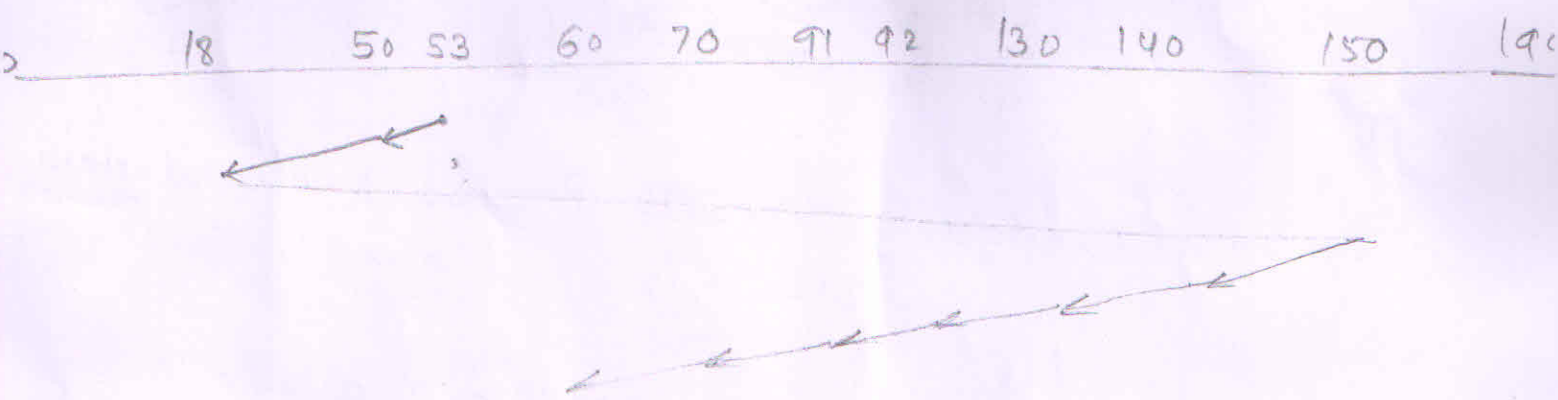


Total head Movement = $(53-18)+(150-18)$

$= 35+132$

$= 167$

6) C-LOOK



Total head Movement = $(53-18)+(150-18)+(150-60)$

$= 35+132+90$

$= 257$

	Max											
	A	B	C	D	A	B	C	D	A	B	C	D
									3	2	1	1
P0	6	0	1	2	4	0	0	1				
P1	1	7	5	0	1	1	0	0				
P2	2	3	5	6	1	2	5	4				
P3	1	6	5	3	0	6	3	3				
P4	1	6	5	6	0	2	1	2				

(01 mark). 1> Total Resources

A = 09
B = 13
C = 10
D = 11
(01 Mark).

2> Need = Max - Allocation

	A	B	C	D
P0	2	0	1	1
P1	0	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

If Need < Available - execute Process & calculate new avail.
Else Do not execute

1> P0 can execute as need < available

New Available = Currently Available + Process's Allocation
= (3,2,1,1) + (4,0,0,1)
= (7,2,1,2)

2> P2 executes = New available = (7,2,1,2) + (1,2,5,4)
= (8,4,6,6)

3> P3 executes : (8,4,6,6) + (0,6,3,3) = (8,10,9,9)

4) P4 executes : (8,10,9,9) + (0,2,1,2) = (8,12,10,11)

5> P1 executes : (8,12,10,11) + (1,1,0,0) = (9,13,10,11)

Yes, Safe sequence = <P0, P2, P3, P4, P1>
(03 Marks).

Memory Partition 100K, 300K, 150K, 650, 450

Process 212K, 315K, 127K, 470K

Fixed size Partitioning $P_0 = 212$, $P_1 = 315$, $P_2 = 127$, $P_3 = 470$

FF

100	300	150	650	450
	P_0	P_2	P_1	

P_4 - Not Allocated

BF

100	300	150	650	450
	P_0	P_2	P_3	P_1

WF

100	300	150	650	450
	P_2		P_0	P_1

P_4 - Not Allocated.

Marks.

(01) Total Internal fragmentation in FF - 88+335+23 = 446

(01) External — 〃 — 〃 — 〃 - = 470.

(01) Total Internal fragmentation in BF = 88+23+135+180 = 426

(01) External - 〃 — 〃 — 〃 — = NIL

(01) Total Internal fragmentation in WF = 438+135+173 = 746

(01) External — 〃 — 〃 — 〃 - = 470K

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

FIFO: (01 Mark)

	4	7	6	1	7	6	1	2	7	2
1	4	4	4	1	1	1	1	1	1	1
2		7	7	7	7	7	7	2	2	2
3			6	6	6	6	6	6	7	7
					*	*	*			*

Total no of Page faults = 10 - 4 = 6

LRU (01 Mark)

	4	7	6	1	7	6	1	2	7	2
1	4	4	4	1	1	1	1	1	1	1
2		7	7	7	7	7	7	2	2	2
3			6	6	6	6	6	6	7	7
					*	*	*			*

Total no of page faults = 10 - 4 = 6.

Optimal (02 Marks).

	4	7	6	1	7	6	1	2	7	2
1	4	4	4	1	1	1	1	1	1	1
2		7	7	7	7	7	7	7	7	7
3			6	6	6	6	6	2	2	2
					*	*	*		*	*

Total No of Page faults = 10 - 5 = 5