COMPUTER SCIENCE AND ENGINEERING

CSE2005 OPERATING SYSTEMS

SCHEDULING ALGORITHMS

SUBMITTED TO

Dr. Renuka Devi

SCOPE

VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI

SUBMITTED BY

P. Maruthi Sai Saketh (18BCE1024)

S. Lokesh (18BCE1082)

K. G. R. Abhiram (18BCE1051)

INTRODUCTION

The **Scheduling Process** is a fundamental function of an operating system. The goal is to share computer resources among a number of processes. Almost all the computer resources are scheduled before use. The main use of scheduling algorithms is to maximize the use of CPU.

The following are a few terms used in scheduling algorithms. Better understanding of these terms will help in developing scheduling algorithms.

THROUGHPUT

This is the number of processes that are completed execution per unit time. In a better scheduling algorithm, the number of jobs processed per unit time should be **maximized.**

TURNAROUND TIME

This is the interval between the time of submission of a process and the time of completion of that process The turnaround time should be **minimized** in a better scheduling algorithm.

WAITING TIME

The sum of the time periods a process spends waiting in the ready queue. Generally the goal of a better scheduling algorithm is to **minimize** the waiting time.

BURST OUT TIME (service time)

It is the time taken for the execution of a process after entering the CPU, Usually to complete the process within the burst out time.

OPTIMIZATION CRITERIA

The Optimization Criteria to use all the resources efficiently is to match the following.

- Maximum CPU Utilization;
- Maximum Throughput;
- Minimum Turnaround Time;
- Minimum Waiting Time; and
- Minimum Response Time.

SCHEDULING ALGORITHMS

There are many scheduling algorithms that were developed over time to meet one or more criteria. The following are a few of them.

FIRST COME FIRST SERVED (FCFS)

It is the simplest algorithm when a process becomes ready, then it joins the ready queue and when the currently running process completes, then the process which comes next into the ready queue will be executed

ROUND ROBIN

The main role of round robin is to reduce the waiting time of the shortest process in the ready queue. The operating system assigns short time slices to each process and if the time slice allocated for a process is not enough for it to run to completion, the process is placed at the back of the ready queue while another process from the ready queue is dispatched

SHORTEST JOB NEXT

The algorithm prioritizes a process with the shortest execution time to run during scheduling.

PRIORITY SCHEDULING

Every process will be assigned with some priority, and the process with the highest priority is allowed to run first. Priority scheduling can be either preemptive or non-preemptive.

FITTEST JOB FIRST DYNAMIC ROUND ROBIN (FJFDRR)

In this algorithm, a factor f is calculated for every process. The process which has the smallest f value will be scheduled first. The factor f is calculated on the basis of given user priority (UP), system priority (SP) which is assigned as lowest burst time has highest priority, user weight (UW which is chosen as 60%) and system weight (SW which is chosen as 40%).

The factor f is calculated as

$$f = UP * UW + SP * BW$$

The dynamic time quantum is calculated as

$$TQ = median\ (remaining\ burst\ time\ of\ all\ the\ processes)$$

This gives better results in comparison to priority based static round robin scheduling PBSSR.

HIGHEST RESPONSE RATIO NEXT (HRRN)

It was proposed to minimize the average value of the normalized turnaround time over all processes. For every process response ratio , R is calculated:

$$R = (w + s)/s$$

where w is the waiting time of the process and s is the expected burst time. Whenever the current process completes, the process with the highest response ratio will be allocated with cpu to run

- **WORKING**
  - The highest response ratio next(HRRN) scheduling algorithm works on the basis of response ratios. Initially the first arrived process will enter in to the CPU then the response ratios for the remaining process will be calculated
  - Response ratio will be calculated based on the waiting time and burst out time **RR=(waiting time + burst out time)/burst out time**
  - Based on that the process with the highest response ratio will be selected and entered in to the CPU for execution
  - In between the waiting time of the remaining process will be changed so again the response ratios are calculated and execution will be done based on the new ratios

- **EXAMPLE**

At time **t=0**, process P1 is considered for execution and runs to completion for 9 time units and terminates.

At time **t=9**, all the other processes (P2, P3, P4 and P5) have arrived. The response ratio, R, is computed for processes P2, P3, P4 and P5 using the waiting time of each process and their corresponding burst time

P2: RR = (7+6)/6 = 2.167 P3: RR= (5+5)/5 = 2

P4: RR = (4+3)/3 = 2.333 P5: RR = (1+7)/7 = 1.143

Process P4 has the highest RR, and as a result runs for 3 time units and terminates.

At time **t=12**, Processes P2, P3 and P5 are available RR, is computed for processes P2, P3 and P5 using the waiting time of each process and their burst out time

P2: R = (10+6)/6 = 2.67 P3: R = (8+5)/5 = 2.6

P5: R = (4+7)/7 = 1.57

Process P2 has the highest response ratio R, runs for 6 time units and terminates.

At time **t=18**, Processes P3 and P5 are available. With the waiting time and their corresponding burst time.

P3: RR = (14+5)/5 = 3.8 P5: RR = (10+7)/7 = 2.43

Process P3 has the highest response ratio, R, runs for 5 time units and terminates.

At time **t=23**, Process P5 is the only process available Process P5 has waited for 15 time units. Process P5 runs to completion for 7 time units and terminates.

Code link: https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/hrrn.cpp

MODIFIED HIGHEST RESPONSE RATIO NEXT (MHRRN)

MHRRN mainly depends on the response ratio and the user priority. it will give equal priorities to both the external and the internal priorities

Hp= 0.5 * RR + 0.5 * E

Where RR= response ratio E=user priority

- **WORKING**
  - Modified highest response ratio is similar to the HRRN but instead of response ratio **hybrid priority (Hp)** is calculated it is based on the priorities
  - That is the response ratio, R is considered as the internal priority of a process while the length of service time (burst out time) of that process is considered to be the external priority, E.
  - Hybrid priority is Hp= 0.5 * RR + 0.5 * E . Here equal priority is given to the both external and internal priorities based on that the remaining process will be executed similar to the HRRN

- **ALGORITHM**

Step 1 : Start

Step 2 : Initialize AWT = 0, ATAT = 0, ART = 0

Step 3 : Processes arrive at the ready queue, RQ

Step 4 : Processes in RQ are sorted and assigned priority according to burst time

Step 5 : Hybrid priority, Hp = 0.5R + 0.5E

Step 6 : Then the process with the highest hp will be executed first and then the remaining Hp will be calculated.

Step 7 : Then the remaining process will be executed based on their Hp values

Step 8 : Then the final process will be executed until its completes

Code link: https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/mhrnn.cpp

MODIFIED PREEMPTIVE HIGHEST RESPONSE RATIO NEXT [MPHRRN]

PMHRRN mainly depends on the response ratio and the user priority. It is same as the MHRRN but it is preemptive it will give equal priorities to both the external and the internal priorities

Hp= 0.5 * RR + 0.5 * E

Where RR= response ratio E=user priority

- **WORKING**
  - Modified preemptive highest response ratio next scheduling algorithm(PMHRRN) it is similar to the MHRRN but **it is preemptive**
  - First arrived process will be executed first and then hybrid priority(Hp) will be calculated based on the internal and external priorities
  - As it is preemptive the highest hp will be given CPU first in between any new process arrives in the ready queue then the **Hp** will be calculated for the two process and then the highest hp will given CPU for execution in between if any new process arrived in the ready queue similarly the execution will be stopped and the hp will be calculated again.
  - Finally the left process will be run until it is completed.

- **ALGORITHM**

Step 1 : Start

Step 2 : Initialize the average waiting time (AWT) = 0, average turnaround time (ATAT) = 0, average response time(ART) = 0

Step 3 : Processes arrive at the ready queue, RQ

Step 4 : Processes in RQ are sorted and assigned priority according to burst time

Step 5 : Hybrid priority, Hp = 0.5R + 0.5E

Step 6 : Are there multiple processes with the highest Hp?

If yes, did processes with the same highest Hp arrive at the same time?

If yes,

Pi = any process with highest Hp,

Else

Pi = process with earliest arrival time among the processes

Else, Pi = process with highest hybrid priority, Hp

Step 7 : Pi executes a burst time

Step 8 : Is Remaining Burst Time [Pi] = 0?

If yes,

process Pi leaves RQ, calculate WT, R, and TAT of Pi , go to 9

Else go to 4

Step 9 : Is RQ = null?

If yes,

calculate AWT, ART, ATAT of all processes, go to 10

Else go to 4.

Step 10 : STOP

Among these, MPHRRN is the best scheduling algorithm because it has the shortest average weighting time

- **EXAMPLE**

Consider 5 different processes as shown below in the table.

process	arrival time	Burst time	priority
P1	0	9	1
P2	2	6	3
P3	4	5	4
P4	5	3	5
P5	8	7	2

At time **t=0,** process P1 is available and runs for 2 time units. because it is the only process available for that time period.

At time **t=2**, processes P1 and P2 are available. Their priorities are determined and their hybrid priority, Hp, is computed. The longer the burst time of the process, the lower the value of the priority of the process.

P1: RR = (0+7) /7 = 1; Hp = 0.5(1) + 0.5(1) = 1

P2: RR = (0+6)/6 = 1; Hp = 0.5(1) + 0.5(2) = 1.5

Process P2 has the high Hp.Therefore p1 preempted and P2 runs for 2 time units. Note that process P2 runs for 2 time units .

At time **t=4**, processes P1, P2 and P3 are available and their hybrid priority, Hp, is computed. The burst time of each process is also updated.

P1: RR= (2+7)/7 = 1.285; Hp= 0.5(1.285) + 0.5(1) = 1.143

P2: RR= (0+4)/4 = 1; Hp= 0.5(1) + 0.5(3) = 2

P3: RR= (0+5)/5 = 1; Hp= 0.5(1) + 0.5(2) = 1.5

P2 has the highest hybrid priority, Hp and so runs for 1 time unit.

At time **t=5**, processes P1, P2, P3 and P4 are available and their hybrid priority, Hp, is computed.

P1: RR = (3+7)/7 = 1.43; Hp= 0.5(1.43) + 0.5(1) = 1.2

P2: RR= (0+3)/3 = 1; Hp= 0.5(1) + 0.5(3) = 2

P3: RR = (1+5)/5 = 1.2; Hp= 0.5(1.2) + 0.5(2) = 1.6

P4: RR = (0+3)/3 = 1; Hp= 0.5(1) + 0.5(3) = 2

Processes P2 and P4 have the highest Hp but process P2 runs for 3 time units because it arrived earlier than P4. It then leaves the queue.

At time **t=8**, processes P1, P3, P4 and P5 are available and their Hp, is computed.

P1: RR = (6+7)/7 = 1.86; Hp= 0.5(1.86) + 0.5(1) = 1.423

P3: RR = (4+5)/5 = 1.8; Hp= 0.5(1.8) + 0.5(2) = 1.9

P4: RR = (1+3)/3 = 1.333; Hp= 0.5(1.333) + 0.5(3) = 2.167

P5: RR = (0+7)/7 = 1; Hp= 0.5(1) + 0.5(1) = 1

Process P4 has the highest hybrid priority, Hp, and runs for 3 time units and leaves the queue. Note that process P4 runs for 3 time units because after the execution of a burst time, when the hybrid priority is recomputed, it turns out to be the process with the highest value. At time, t = 11, processes P1, P3 and P5 are available and their Hp, is computed.

P1: R = (9+7)/7 = 2.29; Hp= 0.5(2.29) + 0.5(1) = 1.643

P3: R = (7+5)/5 = 2.4; Hp= 0.5(2.4) + 0.5(2) = 2.2

P5: R = (3+7)/7 = 1.429; Hp= 0.5(1.429) + 0.5(1) = 1.214

Process P3 has the highest hybrid priority, Hp, runs for 5 time units and leaves the queue.

At time **t=16**, processes P1 and P5 are available and their Hp, is computed

P1: R = (14+7)/7 = 3; Hp= 0.5(3) + 0.5(1) = 2

P5: R = (8+7)/7 = 2.143; Hp= 0.5(2.143) + 0.5(1) = 1.57

Process P1 has the highest hybrid priority, Hp, runs for 7 time units and leaves the queue. At time t=23, only process P5 is available. P5 runs for 7 time units and leaves the queue

COMPARING PMHRRN WITH HRRN AND MHRRN

By comparing the above outputs of different algorithms we came to the conclusion that

Average Turnaround time(ATAT) should be very less for the best scheduling algorithms TAT of HRRN(14.6) is greater than ATAT of MHRRN(14.4) and it is greater than PMHRRN(13.8)

Waiting time(WT) should be very less for the best scheduling algorithms WT of HRRN(8.6) is greater than WT of MHRRN(8.4) and it is greater than PMHRRN(7.8)

Average response time(ART) should be very less for the best scheduling algorithms RT of HRRN(2.59) is greater than RT of MHRRN(2.51) and it is greater than PMHRRN(2.3)

THE IMPROVED ROUND ROBIN

- **WORKING:**
  - The improved Round Robin (IRR) CPU scheduling algorithm works similar to Round Robin (RR) with a small improvement. IRR picks the first process from the ready queue and allocates the CPU to it for a time interval of up to 1 time quantum.
  - After completion of the process's time quantum, it checks the remaining CPU burst time of the currently running process. If the remaining CPU burst time of the currently running process is less than 1 time quantum, the CPU again allocated to the currently running process for remaining CPU burst time.
  - Otherwise, if the remaining CPU burst time of the currently running process is longer than 1 time quantum, the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.
  - By this algorithm, it is proved that improved round robin has better performance than round robin according to waiting time, turnaround time and number of context switches.
- **EXAMPLE:**

If there are 4 processes p1, p2, p3, p4 with arrival time 0 for all the processes and burst time 10, 15, 25, 35. Let the time quantum be 10.

Initially ready queue will be p1(10), p2(15), p3(25), p4(35)

1) p1 will be executed completely

Ready queue p2(15), p3(25), p4(35)

2) p2 will execute for 1 time quantum and the remaining burst time will be 5. 5<time quantum so it will execute again and complete it's total execution.

Ready queue p3(25), p4(35)

3) p3 will execute for 1 time quantum and the remaining burst time will be 15. 15>time quantum so it is added to tail end of ready queue

Ready queue p4(35), p3(15)

4) p4 will execute for 1 time quantum and the remaining burst time will be 25. 25>time quantum so it is added to tail end of ready queue

Ready queue p3(15), p4(25)

5) p3 will execute for 1 time quantum and the remaining burst time will be 5. 5<time quantum so it will execute again and complete its total execution.

Ready queue p4(25)

6) One process left so it will execute until it finishes its total execution.

Gantt Chart:

0 P1 10	10 P2 25	25 P3 35	35 P4 45	45 P3 60	60 P4 85

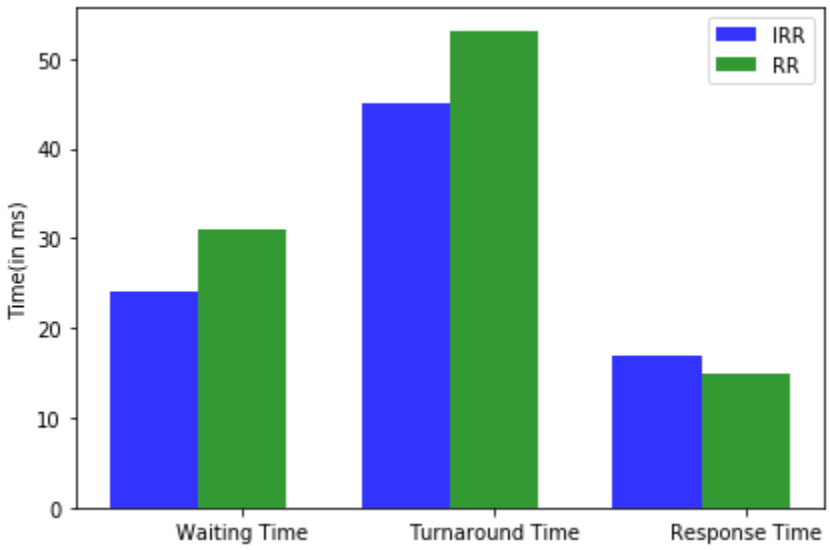
Improved Round robin:

	P1	P2	P3	P4	Average
Waiting Time	0	10	35	50	23.75
Turnaround Time	10	25	60	85	45
Response Time	0	10	25	35	17.5

Round robin:

	P1	P2	P3	P4	Average
Waiting Time	0	30	45	50	31.25
Turnaround Time	10	45	70	85	52.5
Response Time	0	10	20	30	15

We can observe that improved round robin has less average waiting and average turnaround time but has more average response time.



❖ **ALGORITHM:**

Step 1 : Start

Step 2 : Read number of Processes

Step 3 : Read arrival time, burst time for all the processes

Step 4: Read quantum time and find total sum of burst times

Step 5 : Sort the processes according to their arrival time

Step 6 : Make a ready queue for the processes

Step 7 : Repeat Steps 8,9,10,11 until queue becomes empty

Step 8 : Enque the first process of the ready queue and allocate it to CPU for time interval upto 1 time quantum

Step 9 : If the remaining burst time of the currently running process is less than 1 time quantum then goto Step 10…else goto Step 11

Step 10 : Allocate CPU again to currently running process for remaining burst time. After completion of execution goto Step 7. This process finished it's total execution.

Step 11 : Remove the currently running process from the ready queue and put it to tail end of the queue

Step 12 : END

Code link: https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/irr.c

SMART JOB FIRST DYNAMIC ROUND ROBIN (SJFDRR)

It is a round robin based algorithm with not a fixed time quantum but it calculates time quantum for every cycle. It is improved version of Fittest Job First dynamic round robin algorithm. It has both user priority and system priority.

- **WORKING**
  - The idea of this approach is to make the CPU Scheduler arrange the process in ascending order on the burst time and assign the system priority and calculate a smart priority factor 'SPF' for each process.
  - The process having the smallest 'SPF' value will be scheduled first. The time quantum is calculated dynamically. Based on the analysis, we show that the new proposed algorithm (SJFDRR) solves the fixed time quantum problem and increases the performance of Round Robin.
  - In this algorithm every process has two types of priority one is user priority which is given by the user itself (PRU) and second is the system priority which is defined by scheduling system in such a way that lowest burst time has highest system priority (PRS).

- The two important factors are also taken for calculating smart priority factor (SPF) which is user priority ratio (UPR) and system priority ratio (SPR). The user priority has more importance so the user priority ratio is given 55% weight and system priority ratio is given 45% weight. Then Smart Priority Factor 'SPF' is calculated as

$$\boldsymbol{SPF = PRU * UPR + PRS * SPR}$$

- The Smart Time Quantum (STQ) is calculated as follows:

$$\boldsymbol{Smart\ Time\ Quantum\ (STQ) = (B + M)/2}$$

where, M = median of the set of processes in ready queue

B = Maximum burst time of the processes in ready queue

- This algorithm works only when arrival time is same for all the processes.
- This algorithm proves that,

It has minimal average waiting and turnaround time and context switches than "Fittest job first dynamic round robin (FJFDRR) scheduling algorithm"

❖ **ALGORITHM:**

Step 1 : Start

Step 2 : Read number of processes

Step 3 : Read process id, burst time and user priority(PRU)

Step 4 : Set arrival time 0 to all the processes

Step 5 : Find system priority(SPR) for each process by sorting the burst times, lowest gets the highest priority

Step 6 : Find special priority value(SPF) for each process with the formula

$$SPF = 0.55 * PRU + 0.45 * PRS$$

Step 7 : Sort the process according to SPF in the ready queue

Step 8 : Repeat Steps 9, 10, 11 until ready queue becomes empty

Step 9 : Compute time quantum using maximum burst time and median of the processes in the ready queue

Step 10 : If burst time is less than or equal to time quantum

then execute it completely. Else goto Step 11

Step 11 : Execute for 1 time quantum and add it to the tail end of the queue.

- **EXAMPLE:**

If there are 4 processes p1, p2, p3, p4 with arrival time is 0 for all the processes and burst times are 10, 15, 25, 35 respectively.

PRU (User priority):

p1=3, p2=4, p3=2, p4=1

PRS (System priority):

p1=4, p2=3, p2=2, p1=1

Smart Priority factors:

SPF=0.55*PRU + 0.45*PRS

P1=3.45, p2=3.55, p3=2, p4=1

So the order of execution in the ready queue is p2, p1, p3, p4

1) Ready queue: p2(15), p1(10), p3(25), p4(35)

Time quantum = (median + maximum burst time)/2 = 27

Every process executes for 27ms

p2, p1, p3 are completely executed

2) Ready queue: p4(8)

Time quantum = 8

P4 executes for 8ms

Gantt Chart:

0	p2	15	15	p1	25	25	p3	50	50	p4	77	77	p4	85

From p2 to 1st p4 the time quantum is 27 and for the 2nd p4 time quantum is 8.

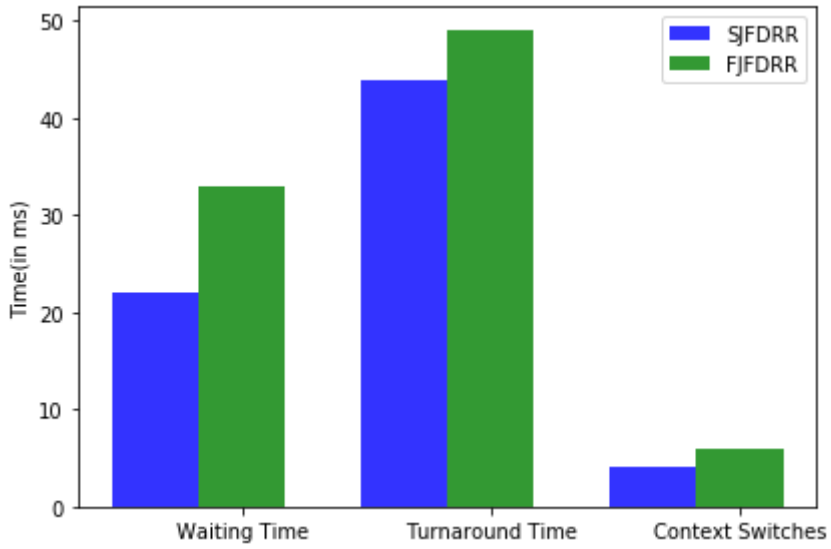
Waiting time:

	P1	P2	P3	P4	Average
Waiting Time	15	0	25	50	22.5
Turnaround Time	25	15	50	85	43.75

Context Switches = 4

	P1	P2	P3	P4	Average
Waiting Time	15	0	45	70	32.5
Turnaround Time	25	15	70	85	48.75

Context Switches = 6



We can observe that SJFDRR has less average waiting time, average turnaround time and context switches than FJFDRR.

Code link(SJFDRR): https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/sjfdrr.c

Code link(FJFDRR): https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/fjfdrr.c

LOTTERY TICKET SCHEDULING:

Lottery ticket algorithm is a type of process scheduling, somewhat different from other scheduling methods. Processes are scheduled in a random manner. Lottery scheduling can be preemptive or non-preemptive.

- **WORKING**
  - In this scheduling every process has some tickets and the scheduler picks a random ticket and the process having that ticket is the winner and it is executed for a time slice and then another ticket is picked by the scheduler. These tickets represent the share of processes.
  - A process having a higher number of tickets gives it more chance to get chosen for execution. If there are 'n' number of processes and there are 'm' number of tickets to be distributed, then the 'm' number of tickets are distributed to processes according to their priorities.
  - The scheduler selects the given number of tickets which are allocated randomly and the processes which have more no of allocated tickets are given first preference.

- **EXAMPLE**

If there are 3 processes to be executed and there are around 100 tickets we assign some tickets for each of the processes. For example, process A has been allocated with 30 tickets (say 1 to 30) and process B (say 31 to 45) with 15 tickets and process C (say 46 to 100) with 55 tickets.

Scheduler is assigned to pick some random ticket, since process A is having more number of tickets it has a high chance of being chosen and getting executed. So this works on a probabilistic method to solve starvation among the clients, who depend on one another process.

❖ WAYS TO MANIPULATE TICKETS

- **Ticket Currency:**

  Scheduler allocates certain number of tickets to different users in a chosen currency and users can give tickets to their processes in a different currency. E.g. Two users A and B are given 100 and 200 tickets respectively. User A runs two processes and gives 50 tickets to each process in A's own currency. User B runs 1 process and gives all 200 tickets to that process in B's currency. Now at the time of scheduling, tickets of each process are converted into global currency i.e A's process will have 50 tickets each and B's process will have 200 tickets and scheduling is done on this basis.

- **Transfer Tickets:**

  A process can pass its tickets to another process.

- **Ticket inflation:**

  With this technique a process can temporarily raise or lower the number of tickets it owns. Ticket inflation is an alternative to explicit ticket transfers in which a client can escalate its resource rights by creating more lottery tickets. In general, such inflation should disallow, since it violates desirable modularity and load insulation properties.

  For example, a single client could easily monopolize a resource by creating a large number of lottery tickets. However, ticket inflation can be very useful among mutually trusting clients; inflation and deflation can be used to adjust resource allocations without explicit communication.

❖ ALGORITHM

STEP 1 : Start

STEP 2 : Read number of processes

STEP 3 : Read burst time and priority for each process

STEP 4 : Read time quantum

STEP 5 : Sort the processes according to priority

STEP 6 : Calculate total burst time

STEP 7 : Formula/Algorithm for assigning number of lottery tickets to each process

For simplicity, in our code we took number of tickets is equal to priority of the process

STEP 8 : While time is less than total time repeat Steps 9, 10, 11, 12

STEP 9 : Get a random ticket in range of tickets assigned to processes

STEP 10 : Find out which process is having that ticket

STEP 11 : Execute that process for 1 time quantum

STEP 12 : Increment time

STEP 13 : End.

Code link:https://github.com/Maruthi-Saketh-2001/Algorithms-codes/blob/master/random_ticket.cpp

COMPARISON OF SCHEDULING ALGORITHMS:

We assume that we have five processes P_1 to P_5 as shown in table I. We compare the results of the discussed algorithms over a set of data provided in Table 1 and Table 2

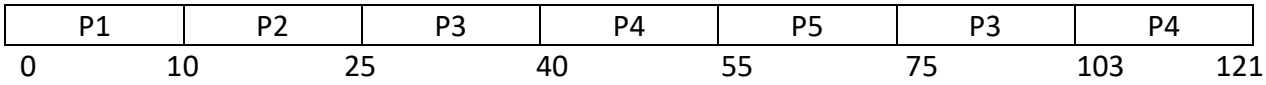
Table 1

Process	Arrival Time	Burst Time	Priority
P1	0	10	5
P2	1	15	4
P3	3	43	1
P4	5	33	2
P5	10	20	3

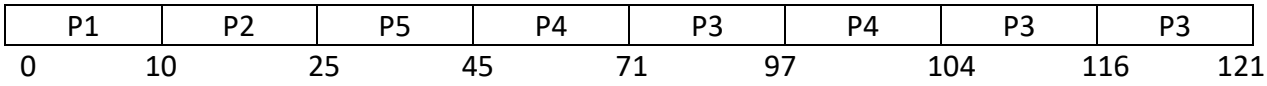
Let the Time quantum be 15

Gantt Charts:

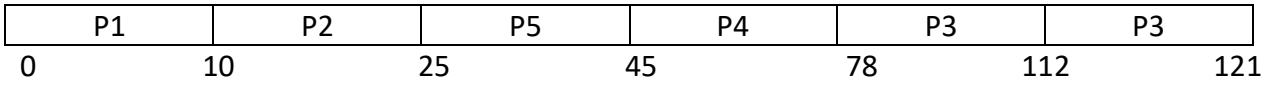
Improved Round Robin:



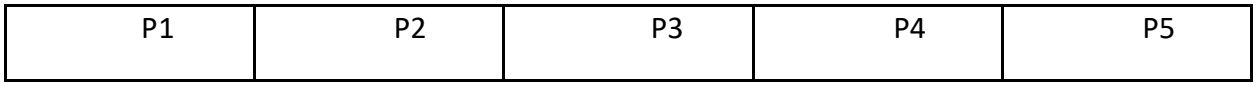
Fittest Job First Dynamic Round Robin (FJFDRR):



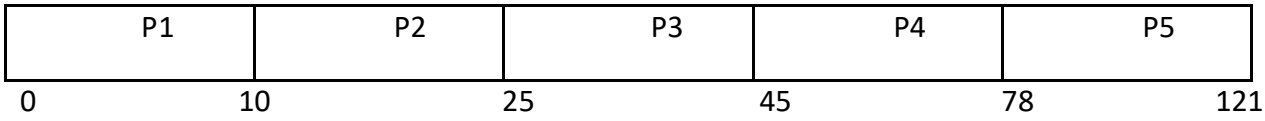
Smart Job First Dynamic Round Robin (SJFDRR):



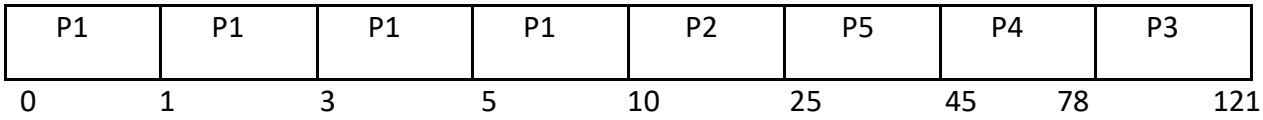
HRRN



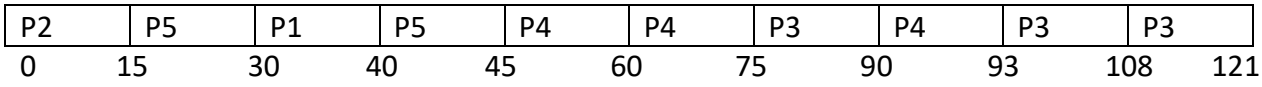
MHRRN



PMHRRN

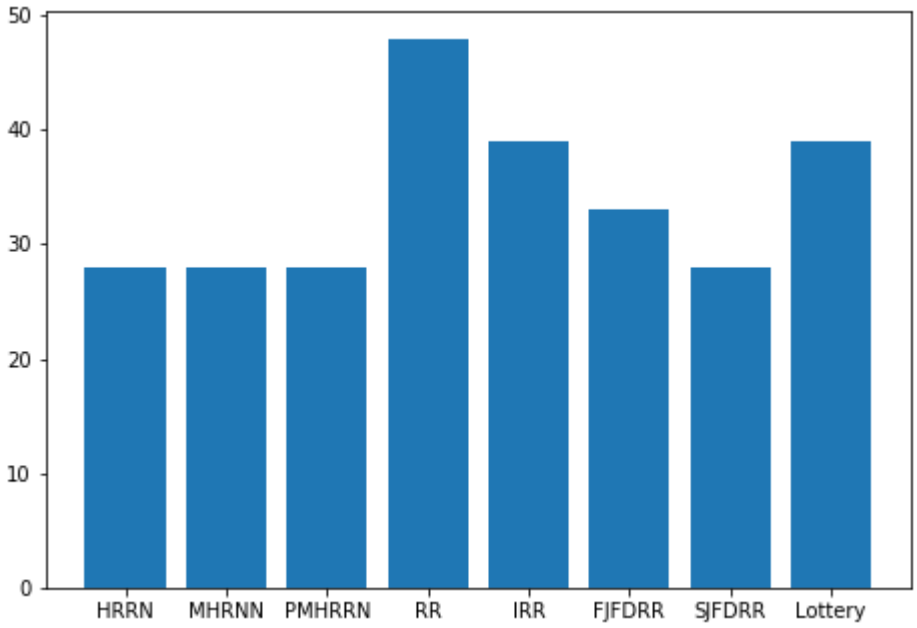


Lottery based algorithm:



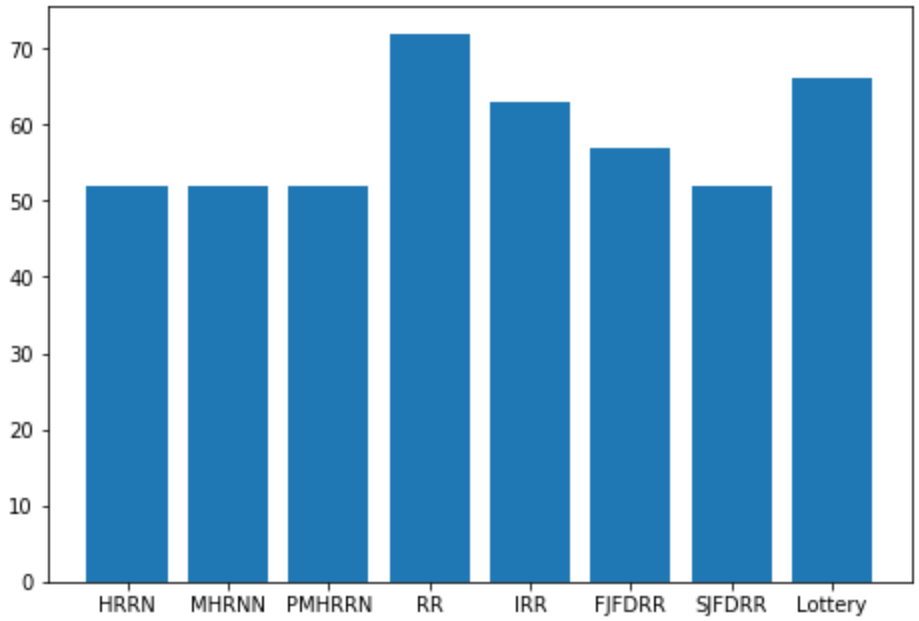
Waiting Time (in ms):

Algorithm	P1	P2	P3	P4	P5	**Average**
FCFS	0	10	25	68	101	**40**
SJF	0	9	75	40	15	**27.8**
RR	0	9	72	83	75	**47.8**
HRRN	0	9	15	40	75	**27.8**
MHRRN	0	9	15	40	75	**27.8**
PMHRNN	0	9	75	40	15	**27.8**
IRR	0	9	57	83	45	**38.8**
FJFDRR	0	9	75	66	15	**33**
SJFDRR	0	9	75	40	15	**27.8**
Random Ticket	30	0	78	60	25	**38.6**



Turnaround Time (in ms):

Algorithm	P1	P2	P3	P4	P5	**Average**
FCFS	10	25	68	101	121	**65**
SJF	10	24	118	73	35	**52**
RR	10	24	115	116	95	**72**
HRRN	10	24	35	73	118	**52**
MHRRN	10	24	35	73	118	**52**
PMHRNN	10	24	118	73	35	**52**
IRR	10	24	100	116	65	**63**
FJFDRR	10	24	118	99	35	**57.2**
SJFDRR	10	24	118	73	35	**52**
Random Ticket	40	15	121	93	30	**65.8**



Response Time(in ms):

Algorithm	P1	P2	P3	P4	P5	**Average**
FCFS	0	10	25	68	101	**40.8**
SJF	0	9	75	40	15	**27.8**
RR	0	9	22	35	45	**22.2**
HRRN	0	9	15	40	75	**27.8**
MHRRN	0	9	15	40	75	**27.8**
PMHRNN	0	9	75	40	15	**27.8**
IRR	0	9	22	35	45	**22.2**
FJFDRR	0	9	68	40	15	**26.4**
SJFDRR	0	9	75	40	15	**27.8**
Random Ticket	30	0	75	45	15	**33**

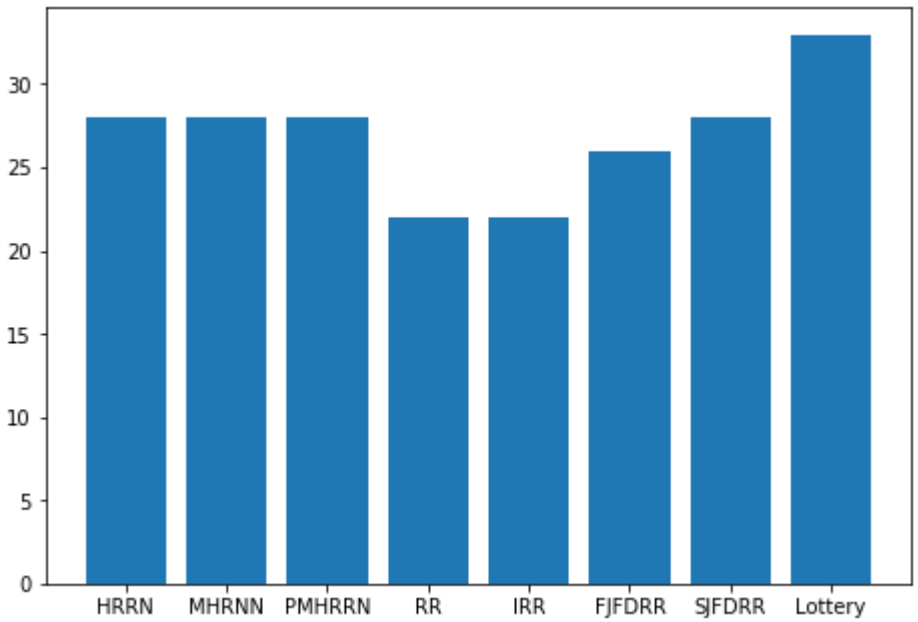


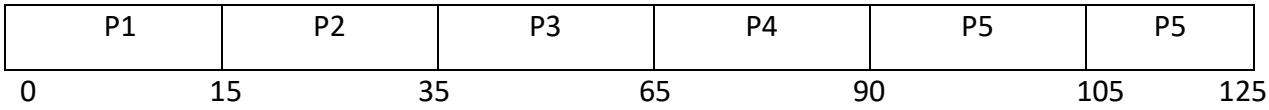
Table 2

Process	Arrival Time	Burst Time	Priority
P1	0	15	5
P2	3	20	4
P3	8	30	3
P4	10	25	2
P5	12	35	1

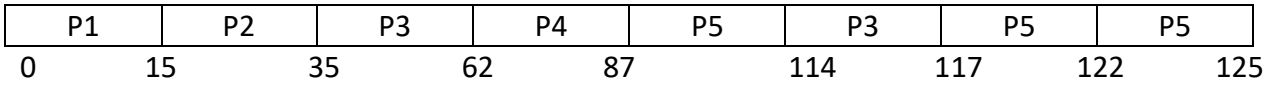
Let the time quantum be 15

Gantt Charts:

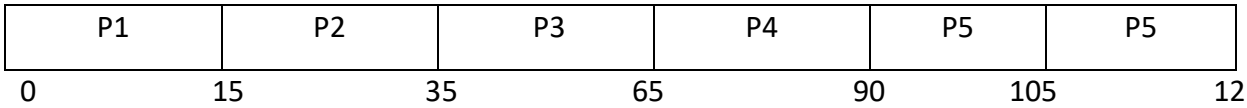
Improved Round Robin:



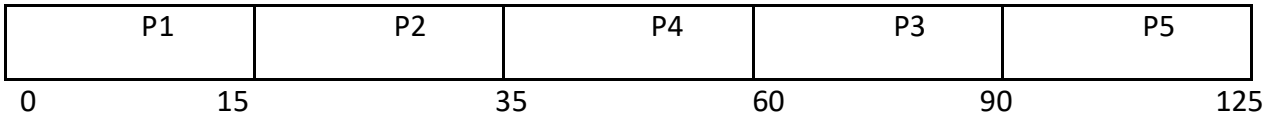
Fittest Job First Dynamic Round Robin (FJFDRR):



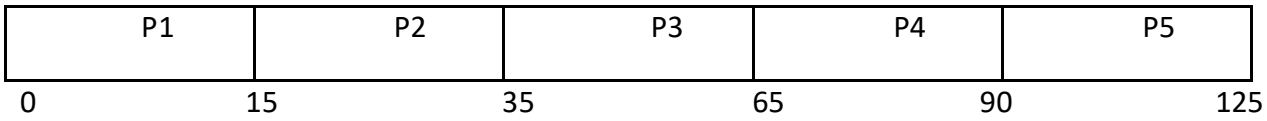
Smart Job First Dynamic Round Robin (SJFDRR):



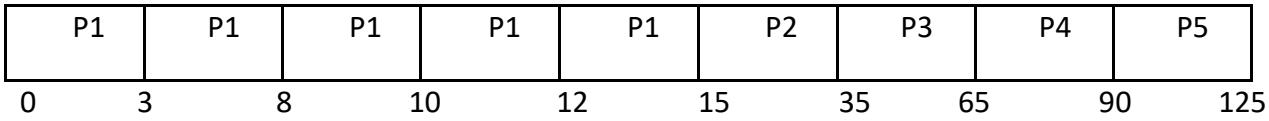
HRRN



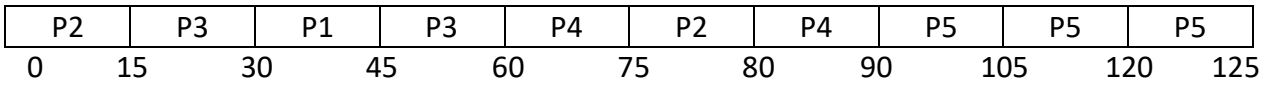
MHRRN



PMHRRN

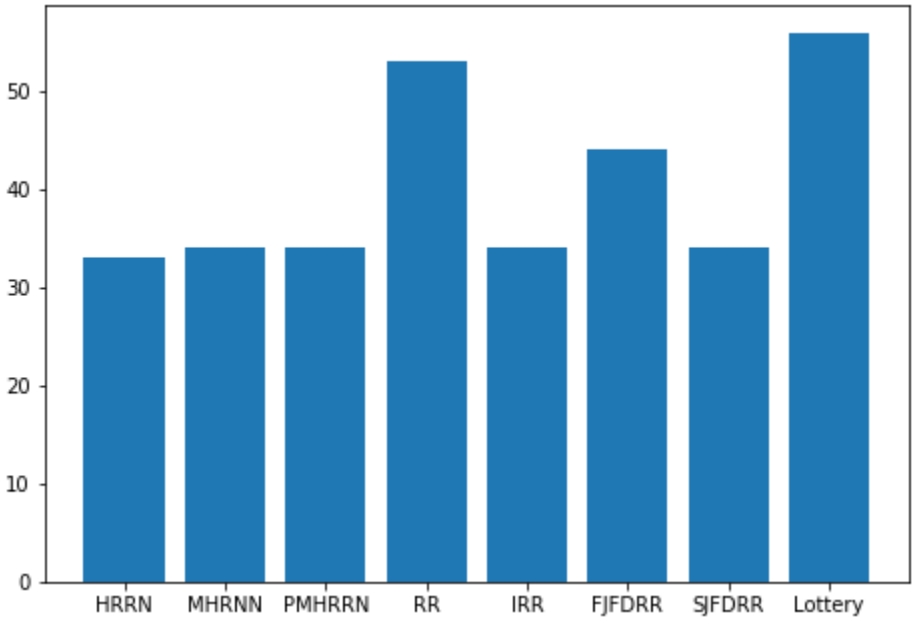


Lottery based algorithm:



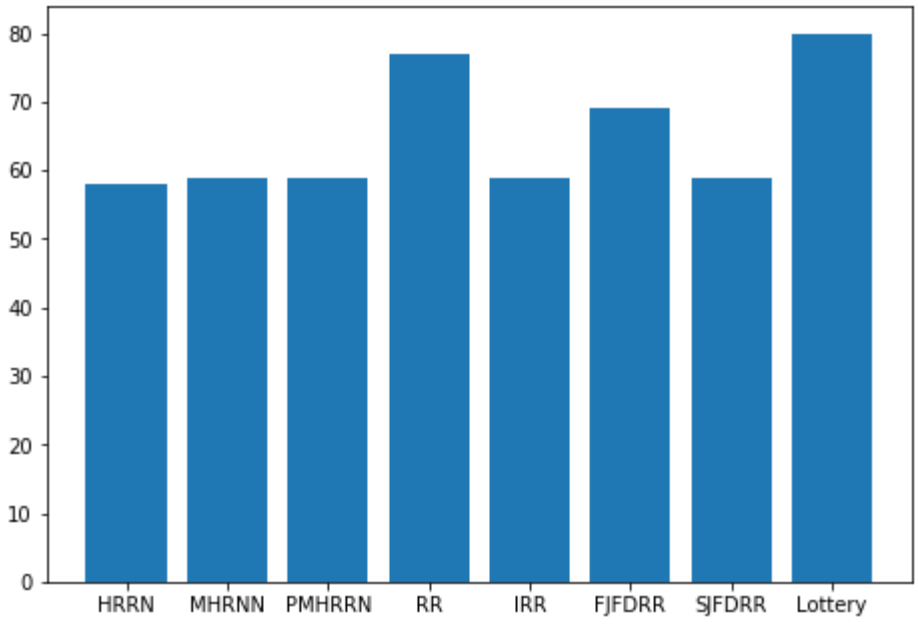
Waiting Time:

Algorithm	P1(in ms)	P2(in ms)	P3(in ms)	P4(in ms)	P5(in ms)	**Average (in ms)**
FCFS	0	15	35	65	90	**41**
SJF	0	12	52	25	78	**33.4**
RR	0	57	57	70	78	**52.4**
HRRN	0	12	25	52	78	**33.4**
MHRRN	0	12	27	55	78	**34.4**
PMHRNN	0	12	27	55	78	**34.4**
IRR	0	12	27	55	78	**34.4**
FJFDRR	0	12	79	52	78	**44.2**
SJFDRR	0	12	27	55	78	**34.4**
Random Ticket	30	65	30	65	90	**56**



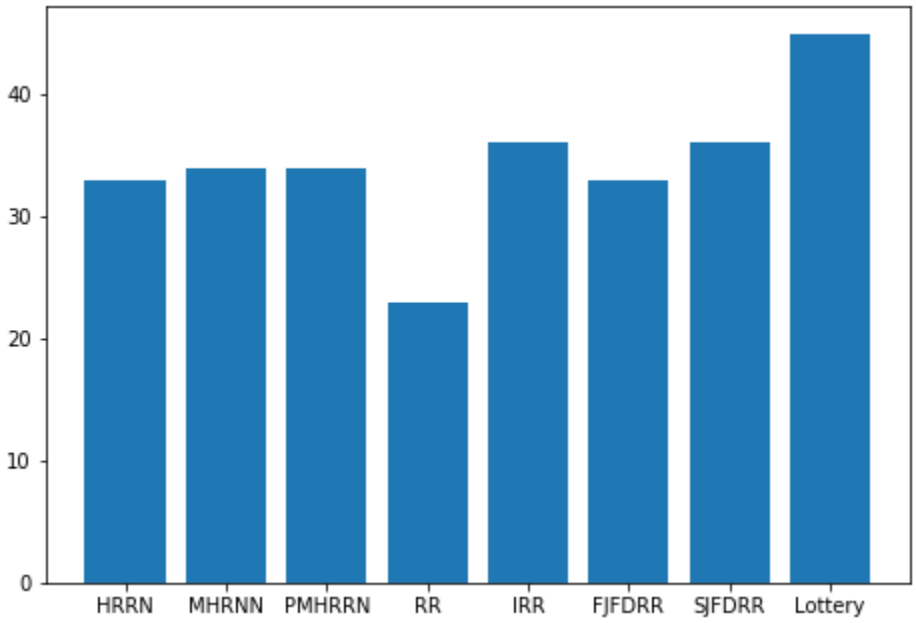
Turnaround Time (in ms):

Algorithm	P1	P2	P3	P4	P5	**Average**
FCFS	15	35	65	90	125	**66**
SJF	15	32	82	50	113	**58.4**
RR	15	77	87	95	113	**77.4**
HRRN	15	32	50	82	113	**58.4**
MHRRN	15	32	57	80	113	**59.4**
PMHRNN	15	32	57	80	113	**59.4**
IRR	15	32	57	80	113	**59.4**
FJFDRR	15	32	109	77	113	**69.2**
SJFDRR	15	32	57	80	113	**59.4**
Random Ticket	45	80	60	90	125	**80**



Response Time (in ms):

Algorithm	P1	P2	P3	P4	P5	**Average**
FCFS	0	15	35	65	80	**39**
SJF	0	12	52	25	78	**33.4**
RR	0	12	22	35	48	**23.4**
HRRN	0	12	25	52	78	**33.4**
MHRRN	0	12	27	55	78	**34.4**
PMHRNN	0	12	27	55	78	**34.4**
IRR	0	12	27	55	88	**36.4**
FJFDRR	0	12	27	52	75	**33.2**
SJFDRR	0	12	27	52	88	**35.8**
Random Ticket	30	0	45	60	90	**45**



Advantages and Disadvantages (According to above Output):

1) **Improved Round Robin(IRR):**
   - Round robin based algorithm, works on time quantum.
   - Produces minimal waiting time and turnaround time than round robin algorithm.
   - Has less number of context switches than round robin algorithm.
   - Response time is more than normal round robin algorithm. Since in this algorithm if remaining burst time is less than quantum time then again it gets CPU execution, so the starting point time/ response time increases if there are any that type of burst times are present prior to it.
   - No user priority or system priority option in this algorithm.
   - In some cases, IRR produces better waiting time, turnaround time and response time than FJFDRR and SJFDRR and in some cases FJFDRR and SJFDRR produces better results. It depends on the user priority (for SJFDRR and FJFDRR) and time quantum (for IRR), if the user gives more priority to less burst time process then SJFDRR and FJFDRR may give better results.
   - Response ratio algorithms are equal or better than IRR in all waiting time, turnaround time and response time.
   - Since random ticket algorithm is also a round robin based and executes randomly, IRR produces better results than random ticket algorithm.
2) **Smart Job First Dynamic Round Robin (SJFDRR):**
   - Round robin based algorithm.
   - Calculates time quantum dynamically.
   - No need to trail and fix a time quantum which produces better results (min waiting and turnaround time).
   - Priority based algorithm, has both system priority and user priority. User priority has given more weightage in this algorithm (0.55 for user and 0.45 for system). System priority is calculated based on burst times, more the burst time less priority. If user gives more priority to less burst time process then the output is better than giving more priority to large burst time process. Since if large process executes first then waiting time for the remaining process increases. If minimal waiting time and turnaround time is agenda then it is better to give more weightage to system priority than user priority.
   - It is same as FJFDRR except dynamic time quantum. Here (median + maximum burst time) is used whereas in FJFDRR only median is used. So here time quantum value is increased in every cycle, so there is a high chance of completing in one cycle which reduces waiting and turnaround time but the response time is increased.
   - It even has less number of context switches than FJFDRR algorithm.

- Since in each cycle all the arrived processes need to be execute, there is no starvation problem. Even if a high priority process enters it needs to wait until next cycle.
- It may produce better or equal waiting and turnaround than IRR. Response time is depend on input values. However it depends on user priority of SJFDRR and fixed time quantum of IRR.
- This algorithm almost equalizes waiting, turnaround time and response time of response ratio algorithms in waiting and turnaround times.
- In most of the cases, comparing to random ticket algorithm this algorithm produces better results for all waiting, turnaround and response time.

3) Lottery Ticket Method:

- This technique can be used to approximate other scheduling algorithms, such as the shortest job next and fair share scheduling.
- The time for starvation reduces.
- Useful for clients who depend on each other processes
- It ensures that it has non-zero probability of being selected at each operation.
- It is more useful for priority scheduling when compared to other scheduling algorithms because more priority processes can be given more tickets, which in turn gives more chance of being operated first.
- If there are equal priority processes and if the time quantum is less, then processes should considerably wait for less time compared to other processes because all the processes will have equal chance of being selected according to the tickets allotted after time quantum of the preceding process.
- The scheduler may not fairly assign priority to a specific process to be executed because of its randomness while choosing a ticket.
- Sometimes the scheduler takes a lot of time to execute all the processes.
- If there is equal priority between processes and if the time quantum is high then processes should wait for a long time until the selected process is finished processing.

COMPARING LOTTERY TICKET ALGORITHM WITH OTHER SCHEDULERS:

- **FCFS:**
  - **FCFS** acts as both preemptive and non preemptive similar to the lottery ticket algorithm.
  - **FCFS** executes the processes in the exact order in which each process enters the CPU.
  - Unlike **FCFS**, the lottery ticket algorithm works as per time quantum and distributed tickets.
- **SJF:**
  - **SJF** works as the closest to completion to be done first and the newly entered process also has the equal opportunity as the older ones.
  - Lottery ticket algorithm can be used to approximate the SJF; it does not give time to complete the whole process in a go, but makes a process work on a given time quantum.
- **Priority:**
  - The scheduler executes processes based on user priority.
  - Lottery ticket algorithm also executes processes by assigning more tickets to a process based on users priority.

4) Highest Response Ratio Next(HRRN)

- From the above output of example one we can observe that HRRN has the minimal waiting time, turnaround time compared with the other algorithms
- And the response time is very less for round robin as it is preemptive we every the process will starts execution quickly compared to the other algorithms
- From the second example we can say that some time its even better then the MHRRN and PMHRRN as the user priorities are not in the proper order
- As we are considering the burst time and the waiting time we can get better output compared with the algorithms that are using only burst time and only waiting time.
- Starvation of the small process will be reduced
- It depends both on the response time and the burst time more the burst time less the response ratio it will be executed at last
- More the waiting time response ratio will be more then there is more chance of executing first.

5) Modified Highest Response Ratio Next(MHRRN)

- From the above outputs we can say that its almost same for the HRRN and the MHRRN for some inputs
- As it is similar to the HRRN here we are considering the user priority also as for calculating Hybrid priority
- In the second input we can see that HRRN is better then the MHRRN as because it is considering the user priority also.
- But we can see that it has the minimum waiting time, turnaround time and the response time so its very much useful if user needs some criteria like doing the some process first
- So the process that needs fast by the user will be executed first based on his priority
- There will be chance of starvation if we give high priority is given for short process then there will be more average waiting time
- We can change the priorities ratio based on the needs of the user. During this cases we get better outputs in HRRN then in MHRRN
- Sometimes the average waiting and turnaround time will be more then the basic algorithms like FCFS SJF if we cannot give the priorities properly

6) Preemptive Highest Response Ratio Next(PMHRRN)

- From the study of above outputs we can say that both the MHRRN and PMHRRN are similar but it is preemptive
- For bulk amount of process we can get better response time and the waiting time compared to the other algorithms
- Since its preemptive every process will enter the ready queue and starts execution quickly because for every arrival time of the new process we preempt the execution of the process and calculate the Hybrid priority and then execute according to it
- So that we can get better response time in PMHRRN
- **PMHRRN** mainly depends on hybrid priority that is both the internal and external user priority.
- Compared to the MHRRN we can reduce the starvation time as it is preemptive the process will be preempted if there is a new process arrived at that time.
- Waiting time will be reduced with it

- There is chance of getting more average waiting time if the priorities is not given properly like giving more priority to the short process

Referenced research papers:

1) https://pdfs.semanticscholar.org/a873/7a10e48d3fdb38d9e89759a8d8974948071d.pdf (IRR)
2) https://www.researchgate.net/publication/320384212_Implementation_of_Smart_Job_First_Dynamic_Round_Robin_SJFDRR_Scheduling_Algorithm_with_Smart_Time_Quantum_in_Multi-core_Processing_System (SJFDRR)
3) https://www.researchgate.net/publication/331658405_Fittest_Job_First_Dynamic_Round_Robin_FJFDRR_scheduling_algorithm_using_dual_queue_and_arrival_time_factor_a_comparison (FJFDRR)
4) https://link.springer.com/chapter/10.1007%2F978-3-642-22191-0_60 (HRRN)
5) http://article.sapub.org/10.5923.j.computer.20150503.02.html#Sec2.6 (PMHRRN)
6) https://www.usenix.org/publications/library/proceedings/osdi/full_papers/waldspurger.pdf (Lottery Scheduling)
7) https://pdfs.semanticscholar.org/f272/36e8b34042a8ef899832541fb681c01a2ebc.pdf (Lottery Scheduling)
8) http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-lottery.pdf (Lottery Scheduling)