ANALYSIS OF MODIFIED ROUND ROBIN SCHEDULING ALGORITHM

FINAL REPORT

Submitted by:

NAME	REG.NO
KOTHA V V S AAKASH	19BCE0186
EDULA VINAY KUMAR REDDY	19BCE0202
MAHANKALI SAI SHARATH CHANDRA	19BCE0316

Prepared for:

OPERATING SYSTEMS (CSE2005)

PROJECT COMPONENT

Submitted To:

PROF. RAHUL RAMAN

Assistant Professor Sr. (Grade 1)

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

TABLE OF CONTENTS:

ABSTRACT:

CPU Scheduling Algorithms is fundamental concept of Operating Systems. A scheduling algorithm solves the problem of deciding which of the outstanding processes is to be allocated to the processor. Round Robin algorithm is one of the prime scheduling among many CPU scheduling algorithms where every process executes for a given time quantum. But logically, every different case has a particular Time quantum at which the execution of Round Robin algorithm will be at its best. Thus, the CPU efficiency and performance are primarily dependent on the value of Time quantum provided. As the value of Time quantum in Traditional Round Robin is independent of the data of processes, the CPU performance decreases. This project proposes a modified Round Robin algorithm which uses the approach of smart Time quantum to make the traditional Round Robin Algorithm more efficient. Our proposed algorithm improves the efficiency of the traditional Round Robin algorithm where the smart Time quantum is generated taking burst times of the processes into consideration. As a consequence, waiting time, turn-around time, and the number of context switches are reduced effectively. Our project also presents a detailed comparative study of different existing modified algorithms with our proposed algorithm.

1. INTRODUCTION:

We know that, in a single-processor system, only one process can run at a time. And any others must wait until the CPU is free and can be rescheduled accordingly. Any process is executed until it must wait, typically for the completion of some I/O request. In a single-processor computer system, the CPU then just sits idle for completion of such I/O requests. All this waiting time is wasted; no useful work is carried out.

With introduction of multiprogramming, we try to use this time productively. Several processes are kept in the memory at one time. When one process has to wait for some I/O request or any other, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues. By switching the CPU among different processes, the operating system can make the computer more productive. The objective of multiprogramming is to have some process running at all times, i.e., to maximize the CPU utilization.

CPU scheduling is the basis of any multiprogramming based operating systems. One of the main reasons for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the application processes and operating system to share the CPU. Another main reason is the that the processes which require to perform I/O operations in the normal course of computation, ordinarily require more time to complete than do the CPU instructions, multiprogramming systems allocate the CPU to another process whenever any process invokes a I/O operation or any other interrupt.

Scheduling refers to the way in which processes are assigned to run on the available CPU's, since there are typically higher number of processes running than there are available CPUs. Scheduling generally refers to deciding of which process should occupy the resource (CPU, disk, etc.). It means that CPU scheduling is a process which allows one process to use the CPU while the execution of another process is kept on hold (in waiting state) due to unavailability of any resource like I/O or any other, thereby making the CPU utilization maximum. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes present in the ready queue to be executed. This selection process is carried out by the short-term scheduler (CPU scheduler). The scheduler selects one of the processes in memory that are ready to execute, and allocates the CPU.

Another component involved in this entire scheduling process is the Dispatcher. The dispatcher is the module that allocates the process selected by the short-term scheduler to the CPU. The functions of Dispatcher involve:

- Context switching
- To restart the process from where it left last time, by remembering the proper locations.

The dispatcher module should be as fast as possible, consider that it is invoked during every process switch. The time taken by the dispatcher module to stop one process and start another process is known as the Dispatch Latency. Low Dispatch Latency implies high utilization of CPU, as the wastage of time in the name of Dispatch Latency is low. Typically, in a drastic case of Dispatch Latency being high, there is no meaning for multiprogramming.

CPU scheduling decisions takes place under the following circumstances:

1. When a process switches from the running state to the waiting state (for any I/O request or any other).
2. When a process switches from the running state to the ready state (when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (after the completion of I/O).
4. When a process terminates.

When Scheduling takes place only under circumstances 1 and 4 shown above, we say the scheduling scheme is a called as a non-preemptive one; otherwise the scheduling scheme is called as a preemptive one.

Under Non-Preemptive scheduling scheme, once the CPU has been allocated to a process, the process is kept in the CPU until it gets terminated or by switches to the waiting state. Under Preemptive scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is already in running state. Therefore, the running process is interrupted for some time and resumed later when the priority task has finished its entire execution.

2. SURVEY:

S.NO	TITLE	AUTHOR	YEAR	PROPOSED WORK
1)	Round Robin based Scheduling Algorithms, A Comparative Study [1]	• Kamal ElDahshan • Afaf Abd Elkader	2017	There are large numbers of algorithms proposed to enhance the traditional Round Robin algorithm. This paper presents a survey with results analysis of comparison between different Round Robin algorithms proposed.
2)	An Improved Round Robin CPU Scheduling Algorithm [2]	• Neha • Ankita Jiyani	2018	In IRR algorithm, we take the first process in the ready queue and execute it till the allocated time slice then we check the remaining burst time, If it is less than the time quantum then the process is executed again till it finishes its execution, if it is more than time slice then the next process in the ready queue is executed. And we remove the currently running process from the ready queue and put it at the end of the ready queue.
3)	An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm [3]	• Abdulrazaq Abdulrahim • Salisu Aliyu • Ahmad M Mustapha, • Saleh E Abdullahi	2014	This modified RR algorithm firstly picks the first process that arrives to the ready queue and allocates the CPU to it for a time interval of up to 1-time quantum. After its completion, it checks the remaining CPU burst time of the currently running process, if it is less or equal to 1-time quantum, the CPU is again allocated to the currently running process for remaining CPU burst time and then it will be removed from the ready queue. The scheduler then proceeds to the next shortest process in the ready queue. else, 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
4)	An Enhanced Round Robin CPU Scheduling Algorithm [4]	• Jayanti Khatri	2016	The proposed algorithm is similar to traditional RR algorithm with a small improvement. The proposed algorithm allocates the processor to the first process of the ready queue for a time interval of up to 1-time quantum. Then it checks the remaining burst time of the currently running process and if it is less than or equal to 1-time quantum, the processor again allocated to the same process and then removed from the ready queue. If the remaining burst time of the currently running process is longer than 1-time quantum, the process will be added at the tail of the ready queue. <u>Assumptions:</u> It has been assumed that the system where all the experiments are performed is single processor system and all the processes have equal priority, number of processes, burst time and time quantum are well known even before submitting the processes to the processor. All the processes arrive at same time. All processes are only CPU bound.
5)	Analysis of Adaptive Round Robin Algorithm and Proposed	• Arpita Sharma • Gaurav Kakhani	2015	This algorithm improved the Adaptive Round Robin Scheduling algorithm. It assumes that all processes arrive at the same time in the ready queue, then they are arranged in an ascending order

	Round Robin Remaining Time Algorithm [5]			according to their burst time. TQ is calculated by Σ pi / 2n. If the remaining CPU burst time of the currently running process is less than the TQ, the CPU is again allocated to the currently running process for the remaining CPU burst time. Otherwise, the process will be added to the end of the ready queue
6)	Achieving Stability in the Round Robin Algorithm [6]	• Kamal ElDahshan • Afaf Abd El-kader • Nermeen Ghazy	2017	This algorithm improved the Adaptive Round Robin Scheduling algorithm. It assumes that all processes arrive at the same time in the ready queue, then they are arranged in an ascending order according to their burst time. TQ is calculated by 0.75*(Σ pi / N). If the remaining CPU burst time of the currently running process is less than the TQ, the CPU is again allocated to the currently running process for the remaining CPU burst time. Otherwise, the process will be added to the end of the ready queue
7)	An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum (IRRVQ) [7]	• Manish Kumar • Dr. Faizur Rashid	2014	(IRRVQ) combines the features of SJF and RR scheduling algorithms with varying time quantum. Initially the processes in the ready queue are arranged in the ascending order of their remaining burst time. CPU is allocated to the processes using RR scheduling with time quantum value equal to the burst time of first process in the ready queue. After each cycle processes in the ready queue are arranged in the ascending order of their remaining burst time and CPU is allocated to the processes using RR scheduling with time quantum value equal to the burst time of first process in the ready queue. Assumptions: For performance evaluation of various Scheduling Algorithms, it has been assumed that all the processes are having equal priority in a single processor environment.
8)	Survey on various Scheduling Algorithms [8]	• Subrata Chowdhury	2018	Comparison between simple Round Robin and Hybrid Round Robin algorithms
9)	A Comparative Review of CPU Scheduling Algorithms [9]	• Mohd Shoaib • Mohd Zeeshan Faraoqui	2014	Summarizing of major CPU Scheduling Algorithms proposed till date such as: FCFS, SJF, SRTF, RR, PRIORITY, HRRN, LJF.
10)	CPU Scheduling Algorithms: A Survey [10]	• Imran Qureshi	2014	Review of different CPU Scheduling Algorithms like FCFS, SJT, PRIORITY, RR executing under different parameters such as running time, burst times and waiting times

11)	A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average [11]	• Abbas Noon • Ali Kalakech • Seifedine Kadry	2011	In this modified RR algorithm, Time quantum is taken as dynamic mean, i.e., mean is updated after any process gets terminated and accordingly Time quantum is updated.
12)	A Paper on Modified Round Robin Algorithm [12]	• Neha Mittal • Khushbu Garg • Ashish Ameria	2015	In this modified RR algorithm, Time quantum is taken as: TQ=Mean of all Burst times, if no. of processes is Odd = Median of all Burst times, if no. of processes is Even
13)	Modified Round Robin Scheduling Algorithm by Dynamic Time Quantum [13]	• Keerthana Baskaran	2017	If the number of process is less than or equal to 3, then normal FCFS approach is followed. Else if the number of process is greater than 3, the time quantum is decided by dividing the no. of process by 2. While running for a particular process, and the process does not end before the given time quantum, then check for the remaining time left to finish the process (remaining burst time). If it is less than time quantum, then execute the entire job, else preempts the job. When the no. processes in ready queue becomes less than or equal to 3, then again FCFS approach is used for the remaining number of process left to be scheduled. <u>Assumptions:</u> The algorithm also requires priority of each process and the default priority of the first entering process into ready queue would be set to 1.
14)	Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes [14]	• Rami J. Matarneh	2009	In this modified RR algorithm, Time quantum is taken as: TQ=Median of all burst times(M), M>25 =25, M<=25
15)	A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean (HARM) [15]	• Ashkan Emami Ale Agha • Somayyeh Jafarali Jassbi	2013	In this modified RR algorithm, Time quantum is taken as: TQ=Harmonic mean of all burst times, if the variation between Burst times is more (Heterogeneous) and some of them are very smaller than others. =Arithmetic mean of all burst times, if the variation between Burst times is more (Heterogeneous) and some of them are very larger than others

16)	An Efficient Dynamic Round Robin Algorithm for CPU scheduling [16]	• Muhammad Umar Farooq • Aamna Shakoor • Abu Bakar Siddique	2017	In this modified RR algorithm, Time quantum is set as 0.8th part of the maximum burst time. Now the scheduler assigns the CPU to all the processes in ready queue with burst time less than that of the Time quantum while larger ones are kept hold on. As soon as all the smaller processes complete their execution; the time quantum is set equal to maximum burst time and then the remaining processes finishes their execution.
170)	A Novel and Efficient Round Robin Algorithm with Intelligent Time Slice and Shortest Remaining Time First [17]	• Saqib Sabha	2018	In this modified RR algorithm, first all the processes in ready queue are arranged in ascending order of their burst times. Then the time quantum is set as: TQ=original TQ, remaining BT <= original TQ =original TQ*(3/2), original TQ < remain BT <= original TQ*(3/2) = original TQ, remaining BT > original TQ*(3/2)
18)	A Modified Version of Round Robin Algorithm "Modulo Based Round Robin Algorithm" [18]	• Ankush Joshi • S. B. Goyal • Kalpana Sharma	2020	In this approach, they inherited the concept of priority scheduling in simple round robin algorithm. We can say it is a blend of round robin and priority scheduling algorithm <u>Assumption:</u> Arrival time of all the processes is same which is '0' and We have all the processes and their burst time in advance. None of the process arrives at execution time. The ready queue is initially empty.
19)	A Hybrid Round Robin Scheduling Mechanism for Process Management [19]	• Khaji Faizan Ali • Abhijeet Marikal • Kakelli Anil Kumar	2020	In phase-1, TQ is calculated using mean and min of BT's. Process which has min burst time and is not executed is given high priority and allocated to CPU for 1 TQ. Phase-1 is repeated until every process gets allocated in CPU for atleast once. In phase-2, ready queue processes are ordered in ascending based on their remaining BT's. After arrangement, process which is first in ready queue is allocated in CPU for 1 TQ. After allocation, if currently executing process remained burst time in CPU is less than or equal to 1TQ then current process is reallocated in CPU for its execution. phase-2 is repeated until the ready queue becomes empty. [TQ= ceiling (avg BT+ min BT)/2]
20)	An Improved Round Robin Scheduling Algorithm for CPU scheduling [20]	• Rakesh Kumar Yadav • Abhishek K Mishra • Navin Prakash • Himanshu Sharma	2010	(1) Initially, we allocate all processes to the CPU only one time as like conventional Round Robin scheduling algorithm, after first time we select shortest job from the waiting queue and its shortest job assigned to the CPU. (2) After that we select next shortest job and do step 2. (3) Till the complete execution of all processes we repeat steps 2 and 3 that means while all the processes has not been completed(executed).
21)	An Optimized Round Robin	• Ajit singh • Priyanka Goyal	2010	Phase 1: We allocate every process to CPU, a single time by applying RR scheduling with a initial time quantum (say t units).

	Scheduling Algorithm for CPU Scheduling [21]	• Sahil Batra		Phase 2: After completing first cycle perform the following steps: Double the initial time quantum (2t units), next we Select the shortest process from the waiting queue and assign to CPU, After that we have to select the next shortest process for execution by excluding the already executed process in this phase. Phase3: For the complete execution of all the processes we have to repeat phase 1 and 2 in cyclic order.

3. SCHEDULING

When we think about Scheduling, various questions come to our mind like:

- Which process should be given higher priority?
- If suddenly any running process gets terminated, which other process is assigned to CPU by Operating system?
- which process is assigned to CPU by Operating system, If suddenly any running process invokes I/O request?
- What is the order of priority of all the processes available in ready queue?

So, all these questions can be answered based on the type of CPU scheduling algorithm we apply. Different types of Scheduling algorithms give different order of Priorities among the available processes. There are variable types of CPU scheduling algorithms available to perform multiprogramming like First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling (PS), Shortest Remaining Time First (SRTF), Multilevel feedback Queue and Round Robin (RR). Performance of these different scheduling algorithms can be measured and compared among themselves using different criteria like Waiting time, Turnaround time, Throughput, etc.

3.1 SCHEDULING CRITERIA

Different CPU scheduling algorithms have different steps of execution and the choice of a particular algorithm favours one particular class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the various parameters of algorithms.

- CPU utilization: - The main aim of any algorithm is to keep the CPU as busy as possible.
- Throughput: - If the processor is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, defined as throughput.
- Turnaround time: - At the level of a particular process, the important criteria is how long it takes to execute that process. The interval between the time of termination of a process and the time of its arrival is the turnaround time.
- Waiting time: - Any scheduling algorithm does not affect the amount of time during which a process executes or does some I/O operation; they majorly affect the amount of time that a process spends waiting in the ready queue.
- Response time: - In a general interactive system, turnaround time may not be the best criteria. Thus, another measure i.e., the interval between the time of arrival of a process and the time at which it gets first response is introduced, defines as response time.

3.2 SCHEDULING MAIN OBJECTIVES

- Minimizing turnaround time.
- Minimizing waiting time.
- Maximizing throughput.
- Maximizing CPU utilization.
- Minimizing response time.
- Providing fairness to all processes.

4. DIFFERENT VERSIONS OF ROUND ROBIN ALGORITHM:

4.1 TRADITIONAL ROUND ROBIN SCHEDULING ALGORITHM

The Round Robin (RR) scheduling algorithm is designed especially for time-sharing systems (A time sharing system allows many users to share the computer resources parallelly.). It is updated version of FCFS similar, in which pre-emption is added to switch between processes. A small unit of time, called a time quantum, is defined. The ready queue is treated as a circular queue in case of Round Robin

4.1.1 IMPLEMENTATION OF ROUND ROBIN SCHEDULING:

- We consider the ready queue as a FIFO queue of processes.
- If any new process enters, then it is added to the tail of the ready queue.
- The short-term scheduler (dispatcher) picks the first process from the ready queue, sets a timer to interrupt the process exactly after 1-time quantum, and dispatches the process.
- In case of the process having a CPU burst of less than 1-time quantum: -
  - Then the process itself will release the processor voluntarily.
  - And then the scheduler will proceed to the next process in the ready queue.
- Otherwise, In the case of CPU burst of the current process under execution being longer than 1-time quantum: -
  - The timer will go off exactly after 1_time quantum and will cause an interrupt to the OS. A context switch will be executed, and the process will be put at the tail of the ready queue.
  - Then the scheduler selects the next process available in the ready queue.
- These recursive process goes on until all the available processes gets executed completely.

4.1.2 ADVANTAGES OF ROUND ROBIN SCHEDULING ALGORITHM:

- Round Robin Scheduling Algorithm doesn't suffer from the problem of starvation.
- All the jobs get a fare allocation of processor.
- Average Response time would be very less, compared to any other algorithm.

4.1.3 DISADVANTAGES OF ROUND ROBIN SCHEDULING ALGORITHM:

- Very important jobs may wait in line, as priority of processes doesn't matter in Round Robin.
- Setting the time quantum too short causes too many context switches and lower the efficiency of CPU, especially if the dispatcher latency is considerably large.
- Setting the time quantum too large may lead to poor response time and, RR indirectly runs in the form of FCFS in that case.

4.2 IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM (IRR)

We know that, Round Robin algorithm is used for executing the processes on CPU with a small unit of time slice. Improved Round Robin Scheduling Algorithm is almost similar to Round Robin scheduling algorithm using time slice but little different in the execution of processes on CPU. In improved round robin scheduling algorithm, we take the first process in the ready queue and execute it till the allocated time slice then we check the remaining burst time of the current running process. If the remaining burst time is less than the allocated time quantum then the running process is executed again till it finishes its execution. But, if the remaining burst time of the running process is more than the allocated time slice then the next process in the ready queue is executed. And we remove the currently running process from the ready queue and put it at the end of the ready queue [2] .

4.2.1 IMPLEMENTATION OF IRR CPU SCHEDULING ALGORITHM:

Step-1: START

Step-2: Make a ready queue of the Processes.

Step-3: Repeat steps 4, 5 and 6 TIL queue ready queue becomes empty.

Step-4: Select the first process from the ready queue and assign it to the CPU for the given time slice.

Step-5: If the remaining burst time of the currently running process is less than that of the allocated time quantum then assign CPU again to the currently running process for execution of remaining CPU burst time. After complete execution of current process, we remove it from the ready queue and go to step 3.

Step-6: Remove the currently running process from the ready queue and put it at the end of the ready queue.

Step-7: END

4.3 AN ADDITIONAL IMPROVEMENT ROUND ROBIN CPU SCHEDULING ALGORITHM (AAIRR)

AAIRR selects the 1st process that arrives in the ready queue and allocates it to the CPU for a time interval of 1-time quantum. After completion of 1-time quantum, the remaining CPU burst time of the currently running process is checked. If the remaining CPU burst time of the currently running process is less than or equal to 1-time quantum, then the CPU is again allocated to the currently running process for the execution for remaining burst time of the process and after execution it removed from the ready queue, then it selects the next shortest process from the ready queue. Otherwise, the process will be added at the end of the ready queue and the next process in the ready queue will be selected and allocated to CPU [3] .

4.3.1 IMPLEMENTATION OF AAIRR CPU SCHEDULING ALGORITHM:

Step-1: START

Step-2: Make a ready queue of the Processes.

Step-3: Pick the 1st process that arrives to the ready queue and allocate the CPU to it for a time interval of 1-time quantum.

Step-4: If the remaining burst time of the currently running process is less or equal to 1-time quantum. We reallocate CPU again to the currently running process for the remaining CPU burst time. After completion of the execution, we remove it from the ready queue.

Step-5: Otherwise, remove the currently running process from the ready queue and we put it at the end of the ready queue.

Step-6: We pick the next shortest process from the ready queue and allocate the CPU to it for a time interval of 1-time quantum then go to step 4.

Step-7: while ready queue in not empty

Step-8: Calculate Average Waiting Time, Average Turnaround Time and Number of Context Switch.

Step-9: END

4.4 AN ENHANCED ROUND ROBIN CPU SCHEDULING ALGORITHM (ENHANCED RR)

This algorithm is an improved version of IRR algorithm. ERR selects the 1st process of the ready queue and allocate it to the CPU for a time interval of 1-time quantum. Then it checks the remaining burst time of the currently running process, if the remaining burst time is less than or equal to 1-time quantum, the current process is again allocated to the CPU for the execution of remaining burst time of the process. After completion of the execution, this process is removed from the ready queue. If the remaining burst time of the currently running process is longer than that of 1-time quantum, then the process will be added at the end of the ready queue [4] .

4.4.1 IMPLEMENTATION OF ENANCED RR CPU SCHEDULING ALGORITHM:

Step-1: Start

Step-2: Make a ready queue of the processes.

Step-3: Allocate the CPU to the 1st process of the ready queue for a time interval of 1-time quantum.

Step-4: If the remaining burst time of the currently running process is less than or equal to that of 1-time quantum then we reallocate CPU again to the currently running process for the remaining burst time. After completion of execution remove it from the ready queue and go to step 3.

Step-5: Repeat 3 and 4 while ready queue becomes empty.

Step-6: Remove the currently running process from the ready queue and put it at the end of the ready queue.

Step-7: END

4.5 ROUND ROBIN REMAINING TIME ALGORITHM (RRRT)

This algorithm improved the Adaptive Round Robin Scheduling algorithm. It assumes that all processes arrive at the same time in the ready queue, then they are arranged in an ascending order according to their burst time. TQ is calculated by Σ pi / 2n. If the remaining CPU burst time of the currently running process is less than the TQ, the CPU is again allocated to the currently running process for the remaining CPU burst time. Otherwise, the process will be added to the end of the ready queue [5] .

4.5.1 IMPLEMENTATION OF RRRT CPU SCHEDULING ALGORITHM:

Step-1: Start

Step-2: Make a ready queue of the processes.

Step-3: Rearrange all the process in increasing order of their CPU burst times.

Step-4: While (ready queue! = EMPTY)

Step-5: Calculate TQ as time quantum = Σ BT(Pi) / 2*n

Step-6: if (remaining burst time < time quantum) we reallocate CPU to the current running process for remaining burst time's execution

Else

We remove the current running process from the ready queue and put it at the end of the ready queue.

Step-7: If no of process > 0 Go to step 6

Step-8: End while

Step-9: Calculate average waiting time, average turnaround time.

Step-10: End

4.6 ENRICHED ROUND ROBIN ALGORITHM (ENRICHED RR)

This algorithm improved the Adaptive Round Robin Scheduling algorithm. It assumes that all processes arrive at the same time in the ready queue, then they are arranged in an ascending order according to their burst time. TQ is calculated by 0.75*(Σ pi / N). If the remaining CPU burst time of the currently running process is less than the TQ, the CPU is again allocated to the currently running process for the remaining CPU burst time. Otherwise, the process will be added to the end of the ready queue [6] .

4.6.1 IMPLEMENTATION OF ENRICHED RR CPU SCHEDULING ALGORITHM:

Step-1: Start

Step-2: Make a ready queue of the processes.

Step-3: Rearrange all the process in increasing order of their CPU burst times.

Step-4: While (ready queue! = EMPTY)

Step-5: Calculate TQ as time quantum = 0.75*(Σ pi / N)

Step-6: if (remaining burst time < time quantum) we reallocate CPU to the current running process for remaining burst time's execution

Else

We remove the current running process from the ready queue and put it at the end of the ready queue.

Step-7: If no of process > 0 Go to step 6

Step-8: End while

Step-9: Calculate average waiting time, average turnaround time.

Step-10: End

5. PROPOSED ALGORITHM

Our proposed algorithm focuses on the drawbacks of traditional Round Robin algorithm i.e., every different case has a particular Time quantum at which the execution of Round Robin algorithm will be at its best. Thus, the CPU efficiency and performance are primarily dependent on the value of Time quantum provided. As the value of Time quantum in traditional Round Robin is independent of provided data of processes, the CPU performance decreases. This project proposes a modified Round Robin algorithm which uses the approach of smart Time quantum to make the traditional Round Robin Algorithm much more efficient. Our proposed algorithm improves the efficiency of the traditional Round Robin algorithm where the smart Time quantum is generated taking burst times of the processes into consideration. As a consequence, waiting time, turn-around time, and the number of context switches are reduced effectively.

5.1 ASSUMPTIONS:

It has been assumed that the system where all the experiments are performed is single processor system and all the processes have equal priority, number of processes, burst time and time quantum are well known even before submitting the processes to the processor. All processes are only CPU bound.

In our algorithm, the smart time quantum will be calculated based on the Burst times of all the processes by the formula:

$$STQ = int \left(\left| \frac{MAX(BT) - MIN(BT) - MEAN(BT's)}{2} \right| + 1 \right)$$

If in any drastic case, the above STQ results to be 1, to reduce the number of context switches due to less TQ (1 unit time), the STQ is calculated from the formula (in above formula MIN(BT's) is ignored to get the below result i.e. Min(BT's) is too small in most of the cases and hence can be neglected to get better results in the views of number of context switches):

$$STQ = int \left(\frac{MAX(BT) - MEAN(BT's)}{2} + 1 \right)$$

5.2 TERMINOLOGIES:

- STQ-Smart Time Quantum
- MAX(BT)-Maximum Burst Time among the available processes
- MIN(BT)-Minimum Burst Time among the available processes
- MEAN(BT's)-Mean of the Burst times of all processes

5.3 DIFFERENT STAGES OF OUR ALGORITHM ARE:

Step-1: START

Step-2: Arrange the process in the expanding request of CPU burst time in the ready queue.

Step-3: Calculate the Mean burst times of the total number of Processes

$$\text{Mean (M)} = (BT1+BT2+BT3+\ldots+BTn)/n$$

Step-4: Set the Smart time quantum (STQ) as indicated by the accompanying technique:

$$STQ = \text{int}\left(\left| \frac{MAX(BT)-MIN(BT)-MEAN(BT's)}{2} \right| +1 \right)$$

In case of the above formula resulting in STQ=1; then consider and reset the STQ as:

$$STQ = \text{int}\left(\left| \frac{MAX(BT)-MEAN(BT's)}{2} \right| +1 \right)$$

Step-5: Allocate the CPU to the First process in the Ready Queue for the age of 1 Smart Time Quantum.

Step-6: If the rest of the CPU burst time of the present process is less than or equivalent to 1 smart time quantum. Reallocate the CPU to current process again for the rest of the burst time. After the total execution of the present process, expel it from the ready queue. Then update the ready queue if any new process arrives.

→Otherwise expel the present process from the ready queue and put it on the tail of the ready queue, only after updating the ready queue with newly arrived processes, for further execution.

Step-7: Pick the following process from the ready queue and give the CPU to it up to the age of 1 Smart time quantum and afterward again revert back to the stage 6.

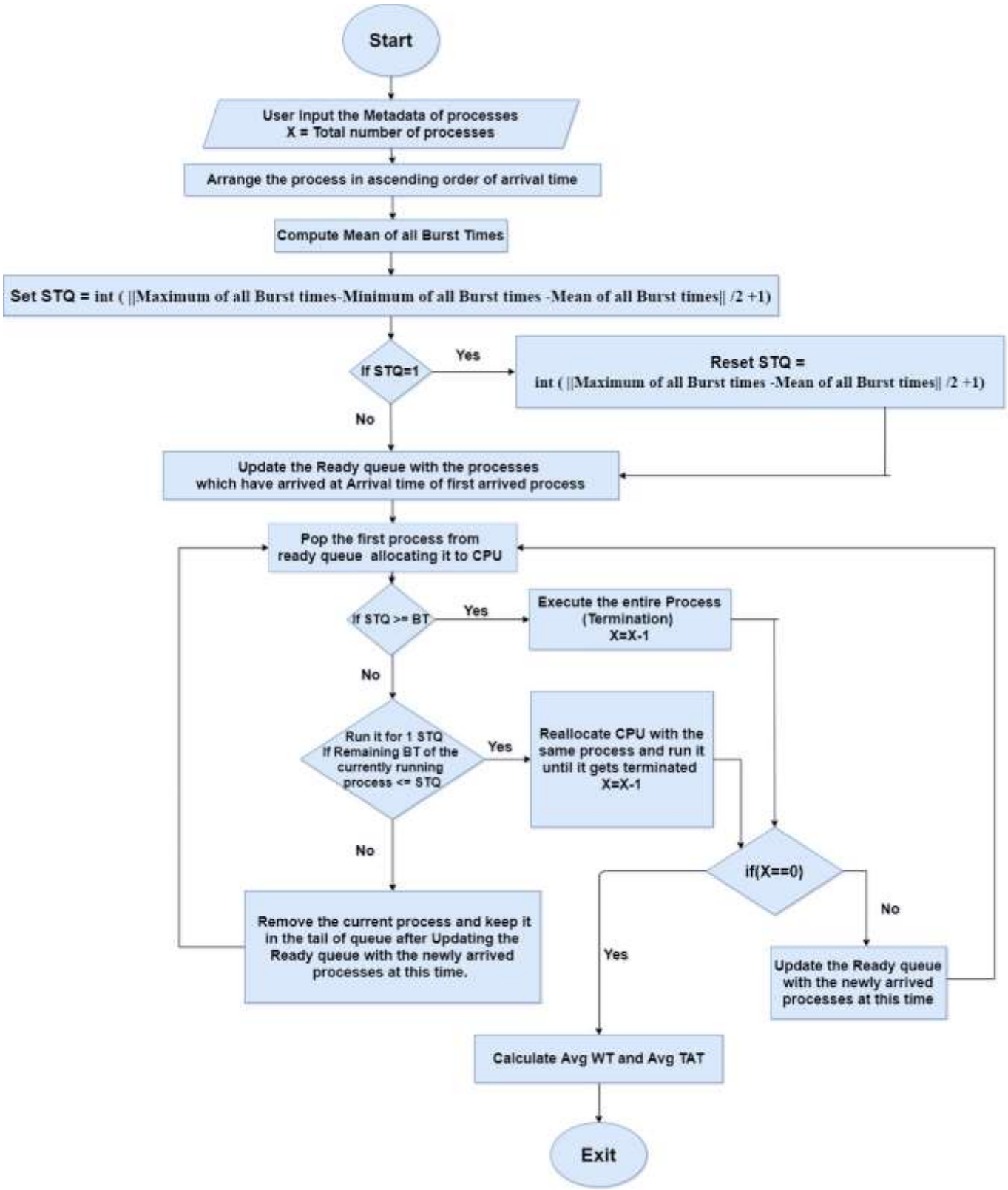
Step-8: Perform this recursive process until all the available processes gets executed completely.

Step-9: Calculate the Average Waiting time and Average Turnaround time of all processes.

Step-10: END

Note: In any Step, if we arrive with a case of 2 or more processes having same arrival time, then we implement SJF i.e., the process with least Burst Time is executed first among the conflicting processes.

5.4 THE FLOWCHART FOR PROPOSED ALGORITHM APPEARS BENEATH:



6. EXPERIMENTAL ANALYSIS

6.1 COMPARISION OF OUR PROPOSED WORK WITH TRADITIONAL ROUNDROBIN ALGORITHM:

6.1.1 ASSUMPTIONS:

It has been assumed that the system where all the experiments are performed is single processor system and all the processes have equal priority, number of processes, burst time and time quantum are well known even before submitting the processes to the processor. All processes are only CPU bound.

6.1.2 EXPERIMENTS PERFORMED:

Two different cases have been taken for performance evaluation of our proposed algorithm.

- In the case 1, CPU burst time is in random orders and processes arrival time is assumed to be zero.
- In the case 2, CPU burst time is in random orders and processes arrival time is assumed to be non-zero.

6.1.3 DATABASE DESCRIPTION:

In all the above-mentioned cases inputs are adopted from the research done by Manish Kumar Mishra and Dr. Faiur Rashid [7] .

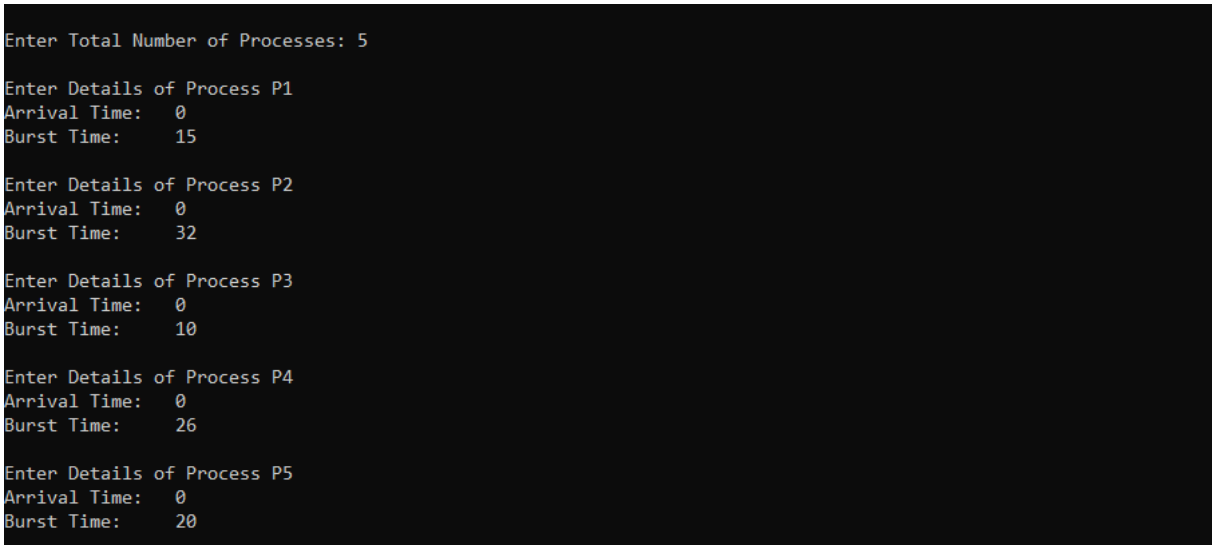
6.1.4 CASE-1: ZERO ARRIVAL TIMES

In this case, CPU burst time is in random orders and processes arrival time is assumed to be zero. A ready queue with 6 processes P1, P2, P3, P4, P5 and P6 has been considered as shown in table 1.

Table 1: Processes with their Arrival and Burst time (Case 1)

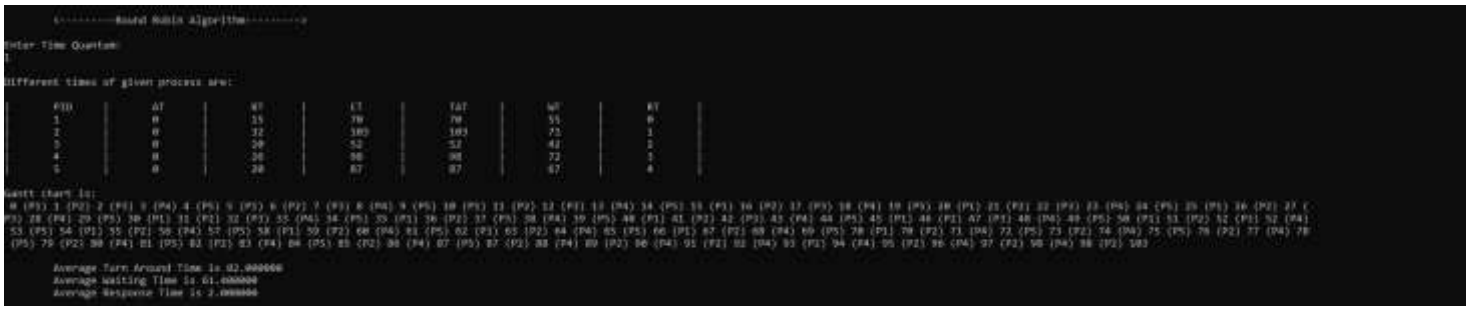
Process	Arrival Time	Burst Time
P1	0	15
P2	0	32
P3	0	10
P4	0	26
P5	0	20

6.1.4.1 DATA ENTRY: (CASE 1)

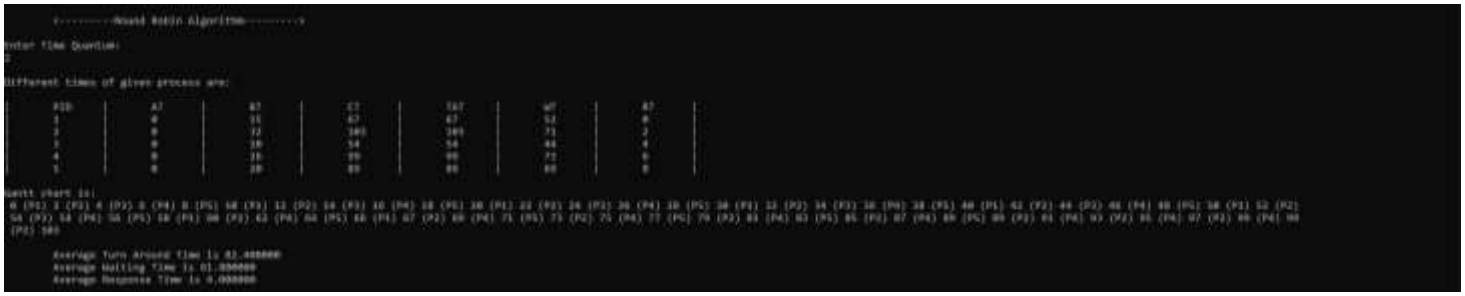


6.1.4.2 OUTPUTS: (CASE 1)

❖ **Round Robin at TQ=1:**



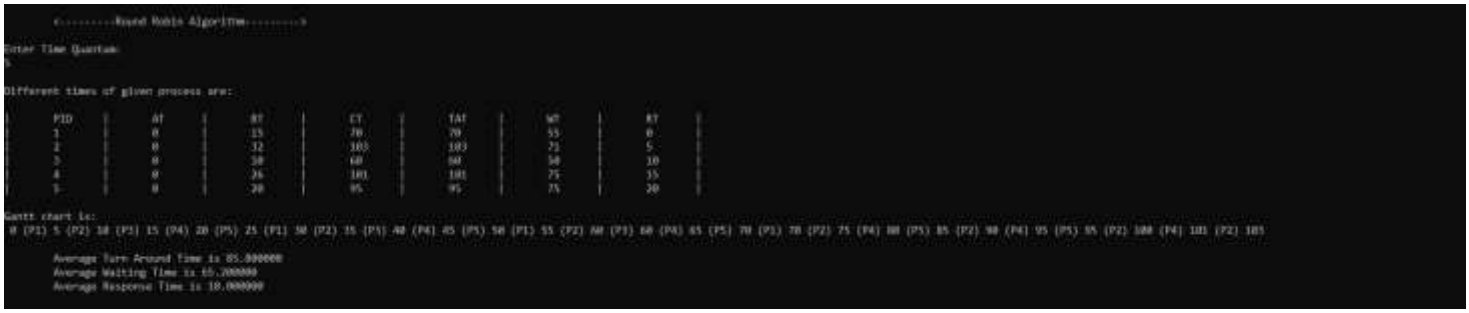
❖ **Round Robin at TQ=2:**



❖ **Round Robin at TQ=3:**

❖ **Round Robin at TQ=4:**

❖ **Round Robin at TQ=5:**



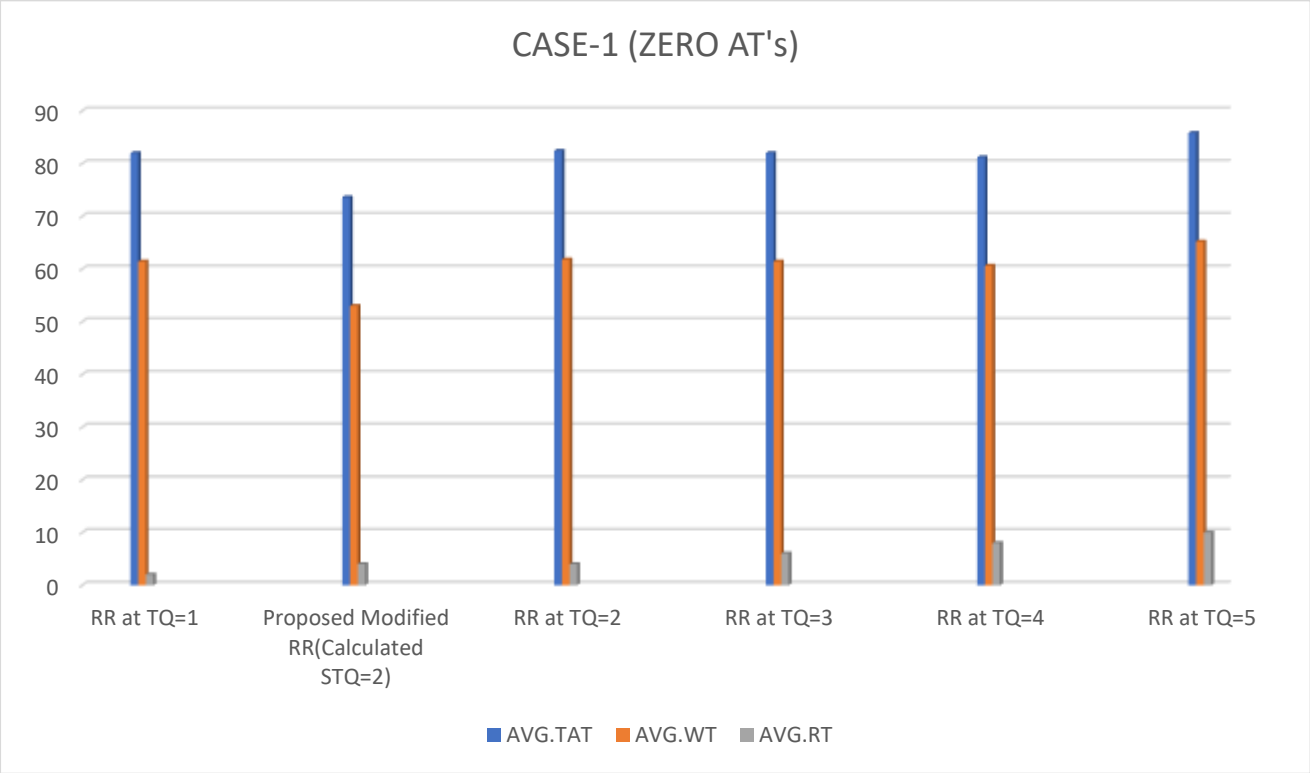
❖ **Proposed Algorithm (Calculated STQ=2):**

6.1.4.3 COMPARISON BETWEEN PROPOSED MODIFIED ROUND ROBIN AND TRADITIONAL RR AT DIFFERENT TIME QUANTUM'S: (CASE 1)

TABULAR COMPARISON:

Algorithm employed	Time Quantum	Average Turn Around Time	Average Waiting Time	Average Response Time
Round Robin	1	82.00	61.40	2.00
Proposed Modified RR	Calculated (STQ=2)	73.60	53.00	4.00
Round Robin	2	82.40	61.80	4.00
Round Robin	3	82.00	61.40	6.00
Round Robin	4	81.20	60.60	8.00
Round Robin	5	85.80	65.20	10.00

GRAPHICAL COMPARISON:



6.1.5 CASE 2: NON-ZERO ARRIVAL TIMES

In this case, CPU burst time is in random orders and processes arrival time is assumed to be non-zero. A ready queue with 5 processes P1, P2, P3, P4 and P5 has been considered as shown in table 2.

Table 2: Processes with their Arrival and Burst time (Case 2)

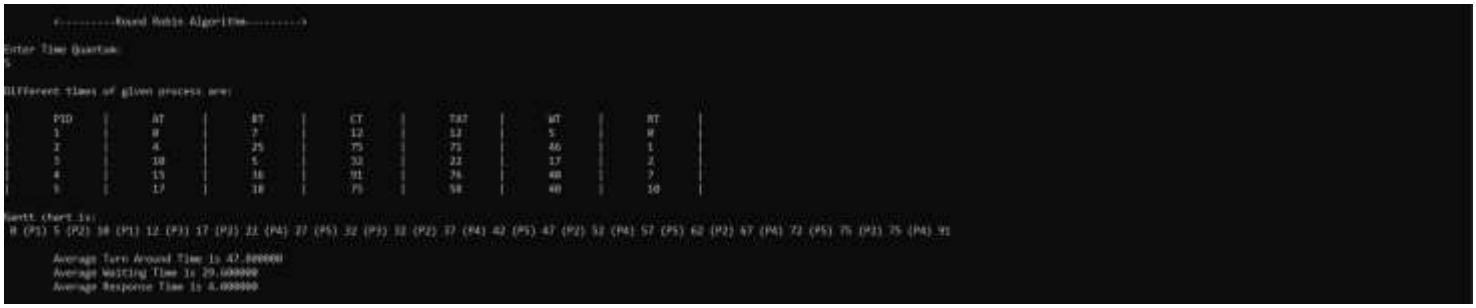
Process	Arrival Time	Burst Time
P1	0	7
P2	4	25
P3	10	5
P4	15	36
P5	17	18

6.1.5.1 DATA ENTRY: (CASE 2)

```
Enter Total Number of Processes: 5

Enter Details of Process P1
Arrival Time:   0
Burst Time:     7

Enter Details of Process P2
Arrival Time:   4
Burst Time:     25

Enter Details of Process P3
Arrival Time:   10
Burst Time:     5

Enter Details of Process P4
Arrival Time:   15
Burst Time:     36

Enter Details of Process P5
Arrival Time:   17
Burst Time:     18
```

6.1.5.2 OUTPUTS: (CASE 2)

❖ **Round Robin at TQ=5:**



❖ **Round Robin at TQ=6:**

❖ **Round Robin at TQ=7:**

❖ **Round Robin at TQ=8:**

❖ **Round Robin at TQ=9:**

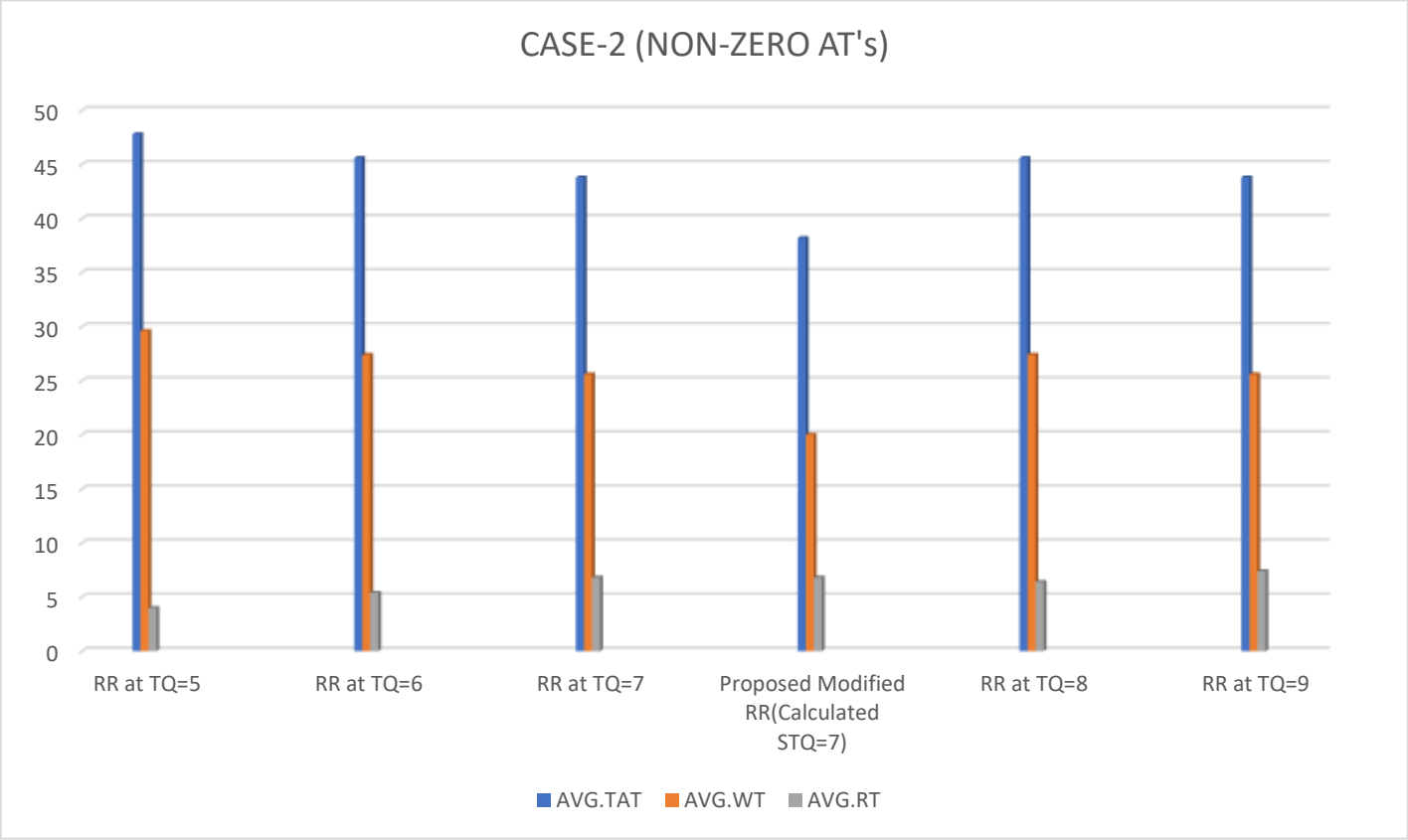
❖ **Proposed Algorithm (Calculated STQ=7):**

6.1.5.3 COMPARISON BETWEEN PROPOSED MODIFIED ROUND ROBIN AND TRADITIONAL RR AT DIFFERENT TIME QUANTUM'S: (CASE 2)

TABULAR COMPARISON:

Algorithm employed	Time Quantum	Average Turn Around Time	Average Waiting Time	Average Response Time
Round Robin	5	47.80	29.60	4.00
Round Robin	6	45.60	27.40	5.40
Round Robin	7	43.80	25.60	6.80
Proposed Modified RR	Calculated (STQ=7)	38.20	20.00	6.80
Round Robin	8	45.60	27.40	6.40
Round Robin	9	43.80	25.60	7.40

GRAPHICAL COMPARISON:



6.2 COMPARISION OF OUR PROPOSED WORK WITH THE ABOVE-MENTIONED IRR, AAIRR, ERR, and RRRT ALGORITHMs:

6.2.1 ASSUMPTIONS:

It has been assumed that the system where all the experiments are performed is single processor system and all the processes have equal priority, number of processes, burst time and time quantum are well known even before submitting the processes to the processor. All the processes arrive at same time. All processes are only CPU bound.

6.2.2 EXPERIMENTS PERFORMED:

Three different cases have been taken for performance evaluation of our proposed algorithm.

- In the case 1, CPU burst time is in random orders and processes arrival time is assumed to be zero.
- In the case 2, CPU burst time is in increasing order and pocesses arrival time is assumed to be zero.
- In the case 3, CPU burst time is in decreasing order and processes arrival time is assumed to be zero.

6.2.3 DATABASE DESCRIPTION:

In all the above-mentioned cases inputs are adopted from the research done by Afaf Abd Elkader and Kamal Eldahshan [1] .

6.2.4 CASE 1: THE BURST TIME OF THE PROCESSES IS RANDOM

In this case, CPU burst time is in random orders and processes arrival time is assumed to be zero. A ready queue with 5 processes P1, P2, P3, P4 and P5 has been considered as shown in table 1.

Table 1: Processes with their Arrival and Burst time (Case 1)

Process	Arrival Time	Burst Time
P1	0	24
P2	0	40
P3	0	5
P4	0	12
P5	0	34

6.2.4.1 DATA ENTRY: (CASE 1)

```
Enter Total Number of Processes: 5

Enter Details of Process P1
Arrival Time:   0
Burst Time:     24

Enter Details of Process P2
Arrival Time:   0
Burst Time:     40

Enter Details of Process P3
Arrival Time:   0
Burst Time:     5

Enter Details of Process P4
Arrival Time:   0
Burst Time:     12

Enter Details of Process P5
Arrival Time:   0
Burst Time:     34
```

6.2.4.2 OUTPUTs: (CASE 1)

- **Proposed Algorithm (Calculated STQ=7):**

- **IRR (TQ=7):**

- **AAIRR (TQ=7):**

- **ERR (TQ=6):**

❖ **RRRT (Calculated TQ=11):**

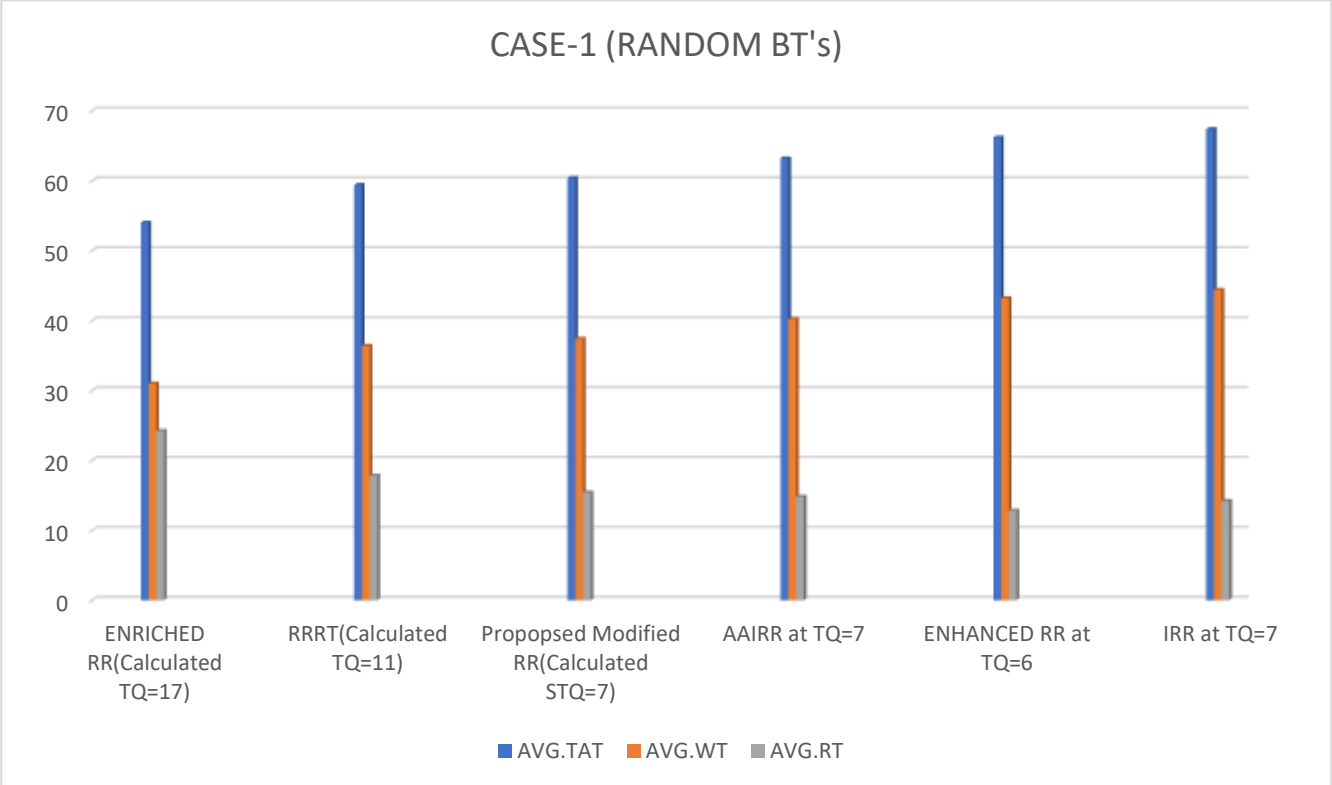
❖ **ENRICHED RR (Calculated TQ=17):**

6.2.4.3 COMPARISON BETWEEN PROPOSED MODIFIED ROUND ROBIN AND VARIOUS MODIFIED RRs (CASE 1)

TABULAR COMPARISON:

Algorithm employed	Time Quantum	Average Turn Around Time	Average Waiting Time	Average Response Time
Proposed Modified RR	Calculated (STQ=7)	60.40	37.40	15.40
IRR	7	67.40	44.40	14.20
AAIRR	7	63.20	40.20	14.80
ENHANCED RR	6	66.20	43.20	12.80
RRRT	Calculated (TQ=11)	59.40	36.40	17.80
ENRICHED RR	Calculated (TQ=17)	54.00	31.00	24.20

GRAPHICAL COMPARISON:



6.2.5 CASE 2: THE BURST TIME OF THE PROCESSES IS IN AN INCREASING ORDER

In this case, CPU burst times are in increasing order and processes arrival time is assumed to be zero. A ready queue with 5 processes P1, P2, P3, P4 and P5 has been considered as shown in table 2.

Table 2: Processes with their Arrival and Burst time (Case 2)

Process	Arrival Time	Burst Time
P1	0	7
P2	0	10
P3	0	20
P4	0	26
P5	0	30

6.2.5.1 DATA ENTRY: (CASE 2)

```
Enter Total Number of Processes: 5

Enter Details of Process P1
Arrival Time:   0
Burst Time:     7

Enter Details of Process P2
Arrival Time:   0
Burst Time:     10

Enter Details of Process P3
Arrival Time:   0
Burst Time:     20

Enter Details of Process P4
Arrival Time:   0
Burst Time:     26

Enter Details of Process P5
Arrival Time:   0
Burst Time:     30
```

6.2.5.2 OUTPUTs: (CASE 2)

- **Proposed Algorithm (Calculated STQ=3):**

- **IRR (TQ=3):**

- **AAIRR (TQ=3):**

- **ENHANCED RR (TQ=2):**

❖ **RRRT (Calculated TQ=9):**

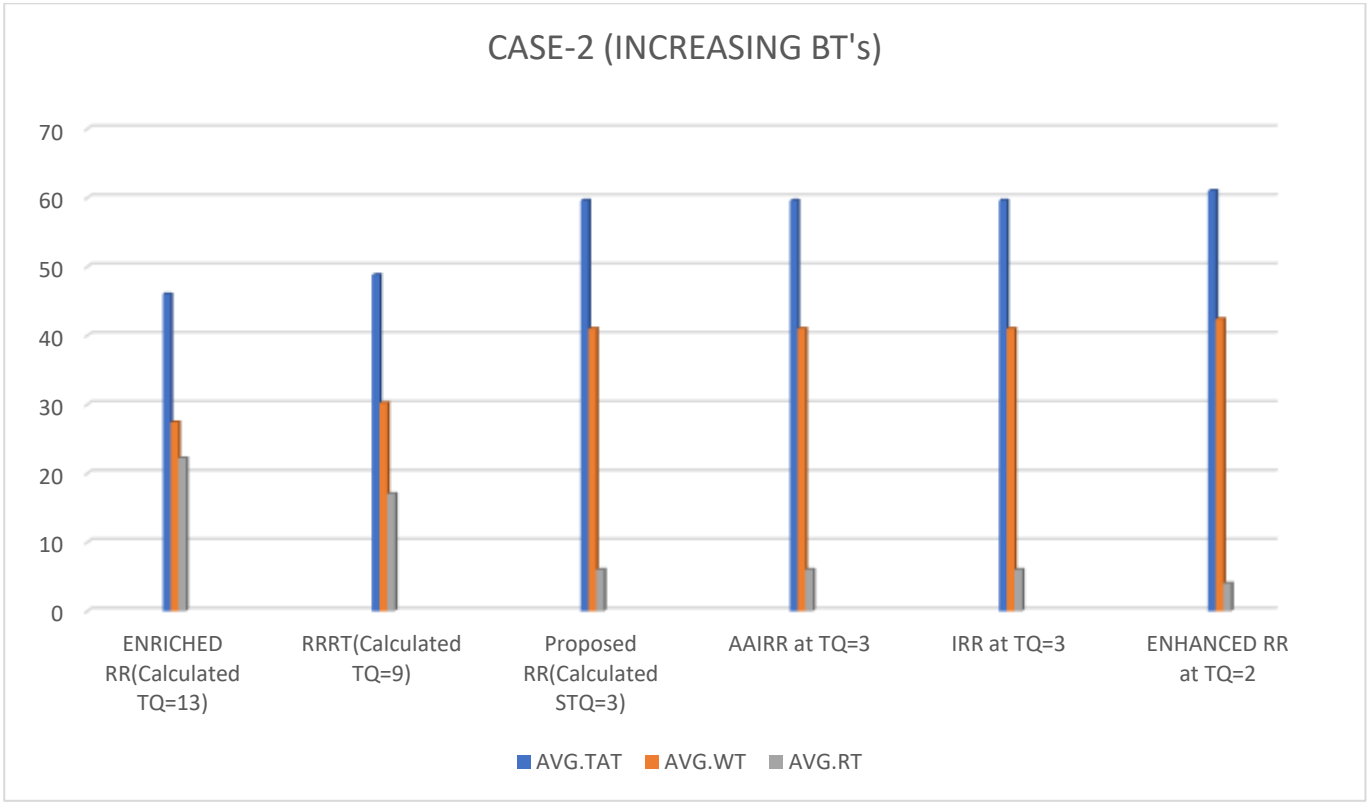
❖ **ENRICHED RR (Calculated TQ=13):**

6.2.5.3 COMPARISON BETWEEN PROPOSED MODIFIED ROUND ROBIN AND VARIOUS MODIFIED RRs (CASE 2)

TABULAR COMPARISON:

Algorithm employed	Time Quantum	Average Turn Around Time	Average Waiting Time	Average Response Time
Proposed Modified RR	Calculated (STQ=3)	59.60	41.00	06.00
IRR	3	59.60	41.00	06.00
AAIRR	3	59.60	41.00	06.00
ENHANCED RR	2	61.00	42.40	04.00
RRRT	Calculated (TQ=9)	48.80	30.20	17.00
ENRICHED RR	Calculated (TQ=13)	46.00	27.40	22.20

GRAPHICAL COMPARISON:



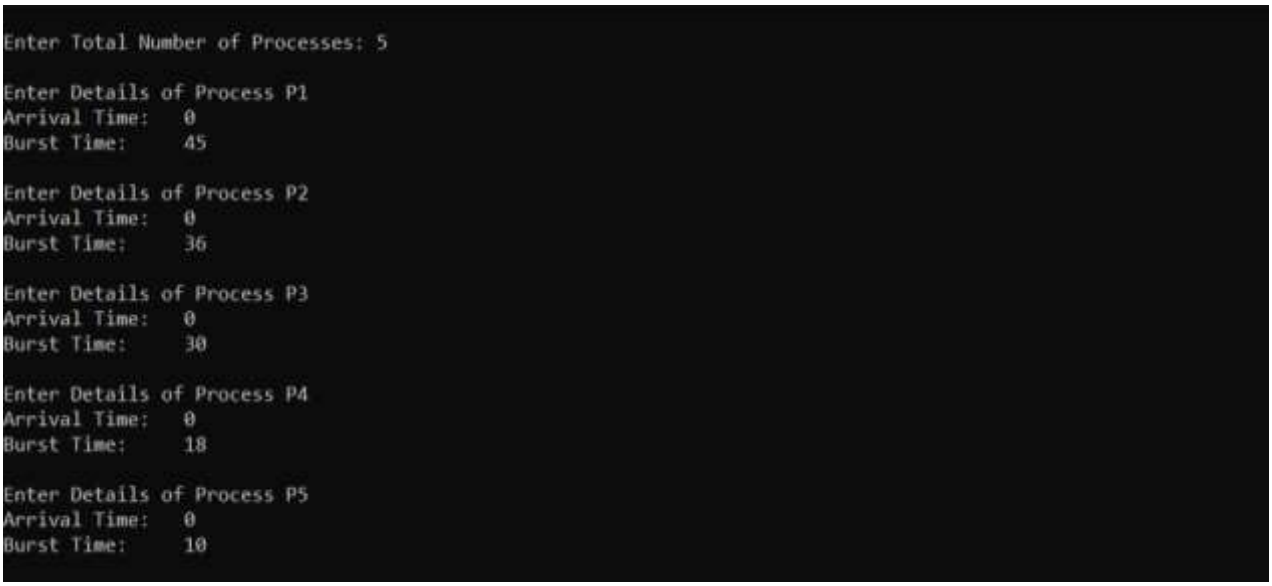
6.2.6 CASE 3: THE BURST TIME OF THE PROCESSES IS IN DECREASING ORDER

In this case, CPU burst times are in decreasing order and processes arrival time is assumed to be zero. A ready queue with 5 processes P1, P2, P3, P4 and P5 has been considered as shown in table 3.

Table 3: Processes with their Arrival and Burst time (Case 3)

Process	Arrival Time	Burst Time
P1	0	45
P2	0	36
P3	0	30
P4	0	18
P5	0	10

6.2.6.1 DATA ENTRY: (CASE 3)



6.2.6.2 OUTPUTs: (CASE 3)

- **Proposed Algorithm (Calculated STQ=5):**

- **IRR (TQ=5):**

- **AAIRR (TQ=5):**

- **ENHANCED RR (TQ=4):**

❖ **RRRT (Calculated TQ=13):**

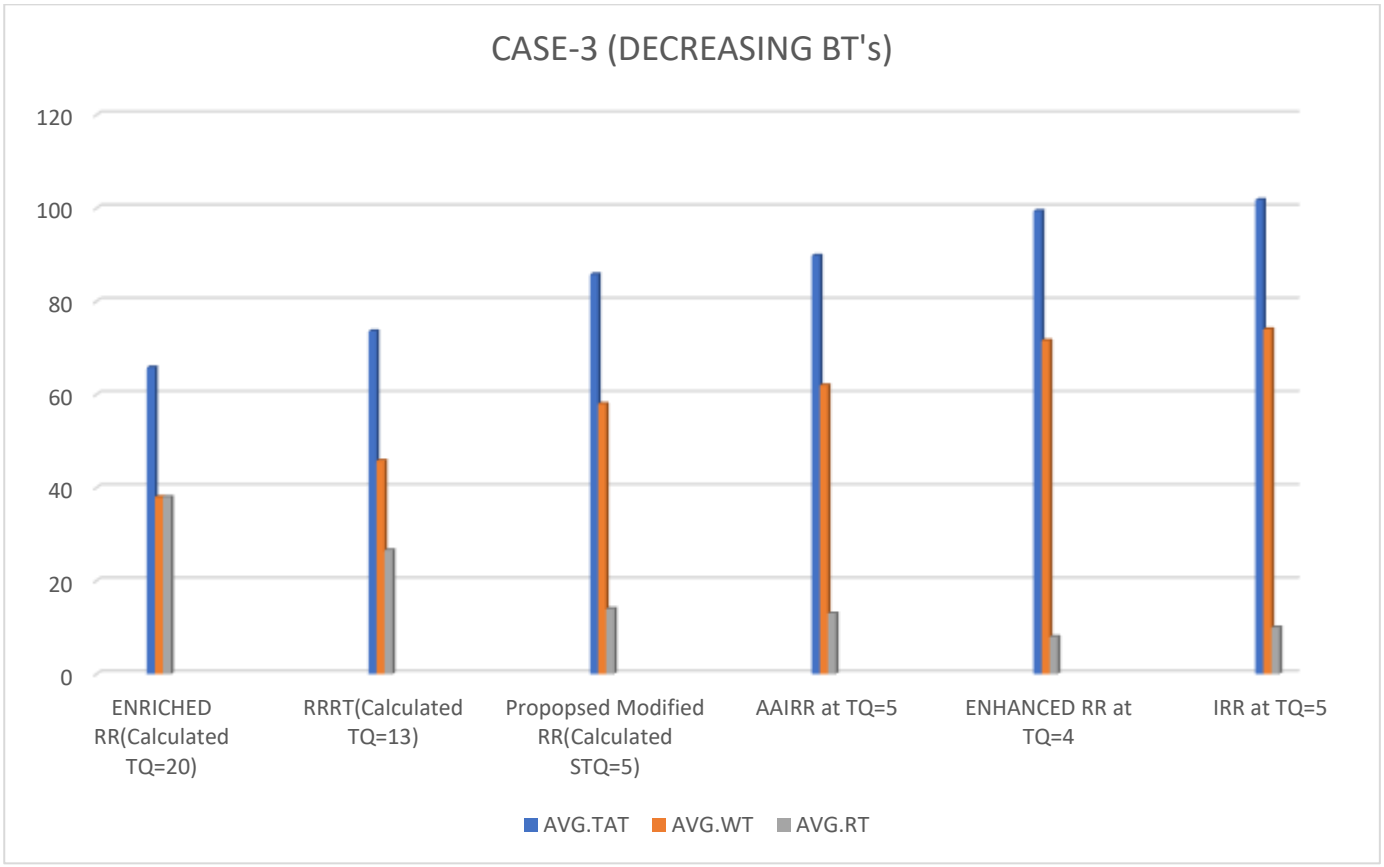
❖ **ENRICHED RR (Calculated TQ=20):**

6.2.6.3 COMPARISON BETWEEN PROPOSED MODIFIED ROUND ROBIN AND VARIOUS MODIFIED RRs (CASE 3)

TABULAR COMPARISON:

Algorithm employed	Time Quantum	Average Turn Around Time	Average Waiting Time	Average Response Time
Proposed Modified RR	Calculated (STQ=5)	85.80	58.00	14.00
IRR	5	101.80	74.00	10.00
AAIRR	5	89.80	62.00	13.00
ENHANCED RR	4	99.40	71.60	8.00
RRRT	Calculated (TQ=13)	73.60	45.80	26.60
ENRICHED RR	Calculated (TQ=20)	65.80	38.00	38.00

GRAPHICAL COMPARISON:



7. OTHER FIELDS WHERE OUR ALGORITHM CAN BE EMPLOYEED:

Cloud computing is one of the prime technologies we are using in recent times, it allows users (individuals or organizations) to access computing resources like: software and hardware as services remotely through the Internet. As cloud computing is serving millions of users parallelly, it must have the ability to meet all users requests with high efficiency and guarantee of quality of service. Therefore, we need to implement an appropriate scheduling algorithm to fairly and efficiently meet all these requests of users simultaneously. As the most well-known Conventional Round Robin algorithm is executing less efficiently when compared our proposed approach, our approach can be implemented in this field with making little modifications related to that field.

8. MATHEMATICAL LOGIC BEHIND SELECTION OF STQ:

Let us consider a Case, where, when a process of Burst Time 10 units is under execution of Round Robin with time quantum 9 units, then after execution of these 9 units of Burst Time i.e. after 1 TQ, the remaining Burst Time of this process reduces to 1 unit, but for execution for this just 1 unit of time, the process have to wait until its next turn comes and then executes. If directly 10 units Burst Time us executed at once, we can find greater extent of reduce in Average Waiting Time and Average Turn Around Time when compared to previous case. In previous case, there is also a possibility that the process waiting for execution for just 1 unit of time may go under starvation if a drastic situation takes place. This Starvation or increases in Average Waiting Time and Average Turn Around Time decreases the CPU performance to a greater extent. In order to reduce this problem, we modified the Round Robin such that the secondary check is performed for the remaining BT after execution of 1TQ for process. In our approach, we perform the secondary check [2] [3] [4] [5] [6], and find that whether the remaining BT is equivalent or less than 1 TQ, if it is satisfied, then CPU is once again allocated to the process immediately, and after completion, it is removed from CPU.

While going through many research papers, we found that in most of the proposed modifications of RR, TQ quantum is only dependent on mean of all the BT's [5] [6]. But we thought that mean is not the exact measure of central tendency. So, we decided to include Max BT and Min BT in our STQ (Smart Time Quantum) proposal. Our STQ is inspired from research done by Khaji Faizan Ali, Abijeet Marikal, and Kakelli Anil Kumar [19] . We modified their work and after performing several experiments, we found that our STQ is giving greater results when compared to conventional Round Robin.

Smart time quantum (STQ) is calculated by the technique:

$$STQ = \text{int}\left(\left| \frac{MAX(BT) - MIN(BT) - MEAN(BT's)}{2} \right| + 1 \right)$$

9. RESULT AND ANALYSIS:

One of the prime tasks of the operating system is the allocation of CPU to the processes which are waiting for execution. Many CPU scheduling algorithms have been introduced with some advantages and disadvantages. Our proposed modification in round robin scheduling algorithm with calculation of smart time quantum is giving better performance than conventional Round Robin algorithm which is shown by taking 2 different cases to ensure a fair comparison between the proposed approach and traditional Round Robin approach, we even adopted the inputs for these cases from a valid source of different journals [1]. In both the cases, average waiting time, average turnaround time and average response time have been reduced in the proposed scheduling algorithm and hence the system performance has been improved. Simulation of results also prove the correctness of the theoretical results. Even our proposed modification in round robin scheduling algorithm with calculation of smart time quantum is giving better performance than existing modified versions of Round Robin algorithm which is shown by taking 3 different cases to ensure a fair comparison between the proposed approach and existing modified versions of Round Robin algorithm, we even adopted the inputs for these cases from a valid source of different journals [7] .

In case-1, where burst times of all the processes are taken in random order, we find that our proposed approach is executing better than IRR, AAIRR, and ENHANCED RR in terms of Average Waiting Time, Average Turn Around Time, and Average Response Time; although both RRRT and ENRICHED RR are better than our approach in terms of both Average Waiting Time and Average Turn Around Time, Our approach is far better than both RRRT and ENRICHED RR in terms of Average Response Time. Here, even Average Response Time is also an important factor of measuring efficiency because conventional Round Robin finds its speciality and uniqueness among other conventional scheduling algorithms mainly because of minimum Average Response Time.

In case-2, where burst times of all the processes are taken in increasing order, we find that our proposed approach is executing equally efficient to IRR and AAIRR, executing better than ENHANCED RR in terms of Average Waiting Time, Average Turn Around Time, and Average Response Time; although both RRRT and ENRICHED RR are better than our approach in terms of both Average Waiting Time and Average Turn Around Time, Our approach is far better than both RRRT and ENRICHED RR in terms of Average Response Time.

In case-3, where burst times of all the processes are taken in decreasing order, we find that our proposed approach is executing better than IRR, AAIRR, and ENHANCED RR in terms of Average Waiting Time, Average Turn Around Time, and Average Response Time; although both RRRT and ENRICHED RR are better than our approach in terms of both Average Waiting Time and Average Turn Around Time, Our approach is far better than both RRRT and ENRICHED RR in terms of Average Response Time.

10. CONCLUSION:

Many algorithms are proposed to improve the efficiency of the Traditional Round Robin algorithm to minimize the average waiting time, average turnaround time and average response time. TQ is the most important parameter in the performance of these proposed algorithms.

We can conclude from the above experiments that the proposed algorithm performs better than the previous developed approaches in terms of performance parameters such as average waiting time, average turnaround time and average response time.

But we can't generalize and say that a particular algorithm is best, even a best algorithm gives worst results in some drastic cases. But as per our experimental analysis our approach is performing very well when compared to traditional round robin in almost all the cases we performed and our approached can be ranked 2nd or 3rd among all the other existing modified versions of Round Robin algorithm: IRR, AAIRR, ENHANCED RR, RRRT and ENRICHED RR.

Future work can be done to implement our proposed approach for adaptive and hard real time system and even it can be converted to dynamic time quantum executable algorithm.

11. CODE (in C):

```
//Menu-Driven program to execute multiple versions of modified Round Robin Algorithm including Traditional Round Robin
#include<stdio.h>
#include<math.h>

int main()
{

    int i,j,n,x,TQ,sum=0,maxBT,minBT,q,total,f,sumWT=0,sumTAT=0,sumRT=0; //total : stores instant of time
    int wait_time=0,turnaround_time=0,mean;
    int choice,k,l,m=0;
    printf("\nEnter Total Number of Processes: ");
    scanf("%d",&n);
    x=n;
    int PID[n],AT[n],RT[n],BT[n],temp[n],CT[n],WT[n],TAT[n]; //RT-Response Time's; CT-Completion Time's
    for(i=0;i<n;i++)
    {
        PID[i]=i+1;
        printf("\nEnter Details of Process P%d\n",i+1);
        printf("Arrival Time:\t");
        scanf("%d",&AT[i]);
        printf("Burst Time:\t");
        scanf("%d",&BT[i]);
        temp[i]=BT[i];
        sum=sum+BT[i];
    }
    maxBT=BT[0];
    minBT=BT[0];
    for(i=0;i<n;i++) //loop to find Minimum and Maximum Burst Time's
    {
      if(maxBT<BT[i])
        maxBT=BT[i];
      if(minBT>BT[i])
        minBT=BT[i];
    }
    printf("\nEnter the Algorithm of your choice:\n"); //for menu-driven mode
    printf("-->Enter (1) for execution of Round Robin\n");
    printf("-->Enter (2) for execution of our Proposed Work \n");
    printf("-->Enter (3) for RRRT\n");
    printf("-->Enter (4) for IRR\n");
```

```
printf("-->Enter (5) for ENHANCED RR\n");
printf("-->Enter (6) for AAIRR\n");
printf("-->Enter (7) for ENRICHED RR\n");
printf("\n\tEnter your choice: ");
scanf("%d",&choice);
if(choice==1)
{
   k=1; //k=1 means 2nd time check for remaining BT is not done
   l=0;
   printf("\n\t<---------Round Robin Algorithm--------->\n\n");
   printf("Enter Time Quantum:\n");
   scanf("%d",&TQ);
}
else if(choice==2)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=1; //l=1 means, while checking 2nd time, it is checked whether the remaining BT<=TQ or not
   printf("\n\t<---------Proposed Algorithm--------->\n\n");
   if(((maxBT-minBT-(sum/n))/2)!=0)
   {
      mean=(sum/n);
      TQ=((maxBT-minBT-mean)/2);
      if(TQ>0)
      {
         TQ++;
      }
      else if(TQ<0)
      {
         TQ=(-TQ)+1;
      }
   }
   if(TQ==1)
   {
      TQ=((maxBT-(sum/n))/2)+1;
   }
   printf("\n-->Calculated Smart TQ is %d\n",(TQ));
}
else if(choice==3)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=0; //l=0 means, while checking 2nd time, it is checked whether the remaining BT<TQ or not
   printf("\n\t<---------RRRT-------->\n\n");
   TQ=((sum)/(2*n));
   printf("\n-->Calculated TQ is %d\n",(TQ));
}
```

```
else if(choice==4)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=0; //l=0 means, while checking 2nd time, it is checked whether the remaining BT<TQ or not
   printf("\n\t<---------IRR--------->\n\n");
   printf("Enter Time Quantum:\n");
   scanf("%d",&TQ);
}
else if(choice==5)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=1; //l=1 means, while checking 2nd time, it is checked whether the remaining BT<=TQ or not
   printf("\n\t<---------ENHANCED RR--------->\n\n");
   printf("Enter Time Quantum:\n");
   scanf("%d",&TQ);
}
else if(choice==6)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=1; //l=1 means, while checking 2nd time, it is checked whether the remaining BT<=TQ or not
   m=1; //m=1 means, first process is executed without any changes in order of execution
   printf("\n\t<---------AAIRR--------->\n\n");
   printf("Enter Time Quantum:\n");
   scanf("%d",&TQ);
}
else if(choice==7)
{
   k=2; //k=2 means 2nd time check for remaining BT is done
   l=0; //l=0 means, while checking 2nd time, it is checked whether the remaining BT<TQ or not
   printf("\n\t<---------ENRICHED RR--------->\n\n");
   mean=((sum)/(n));
   TQ=(3*mean)/4;
   printf("\n-->Calculated TQ is %d\n",(TQ));
}

for(i=0+m;i<n-1;i++) //nested loops for changing order of execution of processes
{
   for(j=i+1;j<n;j++)
   {
      if(AT[i]>AT[j]) //process with less AT will execute first
      {
         q=PID[i];
         PID[i]=PID[j];
         PID[j]=q;
         q=AT[i];
```

```
            AT[i]=AT[j];
            AT[j]=q;
            q=BT[i];
            BT[i]=BT[j];
            BT[j]=q;
            q=temp[i];
            temp[i]=temp[j];
            temp[j]=q;

        }
        if(choice!=4 && choice!=5 && choice!=1) //reordering the processes accordind to BT's if AT's are same (not followed for RR,IRR and ENHANCED RR)
        {
            if(AT[i]==AT[j])
            {
                if(BT[i]>BT[j])
                {
                    q=PID[i];
                    PID[i]=PID[j];
                    PID[j]=q;
                    q=AT[i];
                    AT[i]=AT[j];
                    AT[j]=q;
                    q=BT[i];
                    BT[i]=BT[j];
                    BT[j]=q;
                    q=temp[i];
                    temp[i]=temp[j];
                    temp[j]=q;
                }
            }
        }

    }
}

int GC[(2*sum + 2)],t=0,RQ[2*sum + 2],w=0,r=0;
//GC- Gantt Chart; RQ- Ready Queue
//w: records the writing(when a new process arrives) index of RQ at a particular instant
//r: records the reading(executing) index of RQ at a particular instant
if(AT[0]==0) //if any process arrived at t=0
{
    total=AT[0]; //initial t=0
    GC[t]=total;
    t++;
```

```
}
else
{
   total=AT[0];
   GC[t]=0;
   t++;
   GC[t]=-1; //for NULL (NO PROCESS EXECUTED)
   t++;
   GC[t]=total;
   t++;
}
//checking for arrival of any new processes (initial updation of RQ)
for(i=0;i<n;i++)
{
   f=0;
   for(j=0;j<w;j++)
   {
      if(RQ[j]==i)
      {
         f++;
         break;
      }
   }
   if(AT[i]<=total && temp[i]!=0 && f==0)
   {
      RQ[w]=i;
      w++;
   }
}
while(x!=0) //x records the remaining no. of processes to be executed
{
      if(r<w)
      {
            if(temp[RQ[r]] < k*TQ+l )
            {
               if(GC[t-2]!=RQ[r]) //previous process is not the same process
               {
                  GC[t]=RQ[r];
                  t++;
                  if(temp[RQ[r]]==BT[RQ[r]]) //a process executing for first time
                  {
                     RT[RQ[r]]=total-AT[RQ[r]]; //So, RT is updated
                  }
                  total=total+temp[RQ[r]];
                  GC[t]=total;
```

```
    t++;
    temp[RQ[r]]=0;
    CT[RQ[r]]=total;
    TAT[RQ[r]]=CT[RQ[r]]-AT[RQ[r]];
    sumTAT=sumTAT+TAT[RQ[r]];
    WT[RQ[r]]=TAT[RQ[r]]-BT[RQ[r]];
    sumWT=sumWT+WT[RQ[r]];
    x--;
    r++;
}
else //previous process is the same process.So, simply old proccess values are updated
{
    if(temp[RQ[r]]==BT[RQ[r]]) //a process executing for first time
    {
        RT[RQ[r]]=total-AT[RQ[r]]; //So, RT is updated
    }
    total=total+temp[RQ[r]];
    GC[t-1]=total;
    temp[RQ[r]]=0;
    CT[RQ[r]]=total;
    TAT[RQ[r]]=CT[RQ[r]]-AT[RQ[r]];
    sumTAT=sumTAT+TAT[RQ[r]];
    WT[RQ[r]]=TAT[RQ[r]]-BT[RQ[r]];
    sumWT=sumWT+WT[RQ[r]];
    x--;
    r++;
}
    //checking for arrival of any new process (updation of RQ)
    for(i=0;i<n;i++)
    {
        f=0;
        for(j=0;j<w;j++)
        {
            if(RQ[j]==i)
            {
                f++;
                break;
            }
        }
        if(AT[i]<=total && temp[i]!=0 && f==0)
        {
            RQ[w]=i;
            w++;
        }
    }
```

```
}
else //BT>TQ ( according to algorithm employed )
{
   if(GC[t-2]!=RQ[r]) //previous process is not the same process
   {
      GC[t]=RQ[r];
      t++;
      if(temp[RQ[r]]==BT[RQ[r]]) //a process executing for first time
      {
          RT[RQ[r]]=total-AT[RQ[r]]; //So, RT is updated
      }
      total=total+TQ;
      GC[t]=total;
      t++;
      temp[RQ[r]]=temp[RQ[r]]-TQ;
      //checking for arrival of any new process (updation of RQ)
      for(i=0;i<n;i++)
      {
         f=0;
         for(j=0;j<w;j++)
         {
            if(RQ[j]==i)
            {
               f++;
               break;
            }
         }
         if(AT[i]<=total && temp[i]!=0 && f==0)
         {
            RQ[w]=i;
            w++;
         }
      }
      RQ[w]=RQ[r];
      w++;
      r++;
   }
   else //previous process is the same process.So, simply old values are updated
    {
      if(temp[RQ[r]]==BT[RQ[r]]) //a process executing for first time
      {
          RT[RQ[r]]=total-AT[RQ[r]]; //So, RT is updated
      }
      total=total+TQ;
```

```
            GC[t-1]=total;
            temp[RQ[r]]=temp[RQ[r]]-TQ;
            //checking for arrival of any new process (updation of RQ)
            for(i=0;i<n;i++)
            {
               f=0;
               for(j=0;j<w;j++)
               {
                  if(RQ[j]==i)
                  {
                     f++;
                     break;
                  }
               }
               if(AT[i]<=total && temp[i]!=0 && f==0)
               {
                  RQ[w]=i;
                  w++;
               }
            }
            RQ[w]=RQ[r];
            w++;
            r++;
         }

      }

}
else //all the processes in RQ are executed, So,for every 1 unit of time ,a check is performed in RQ
{
   if(GC[t-2]!=-1) //if last sec NULL(no process) is not performed
   {
      GC[t]=-1; //for NULL
      t++;
      total=total+1; //for every 1 unit of time ,a check is performed in RQ
      GC[t]=total;
      t++;
   }
   else //if last sec also NULL(no process) is performed, so,simply we modify the values
   {
      total=total+1;
      GC[t-1]=total;
   }
   //checking for arrival of any new process (updation of RQ)
   for(i=0;i<n;i++)
```

```
            {
                f=0;
                for(j=0;j<w;j++)
                {
                    if(RQ[j]==i)
                    {
                        f++;
                        break;
                    }
                }
                if(AT[i]<=total && temp[i]!=0 && f==0)
                {
                    RQ[w]=i;
                    w++;
                }
            }
        }

    }
    printf("\nDifferent times of given process are:\n");
    printf("\n|\tPID\t|\tAT\t|\tBT\t|\tCT\t|\tTAT\t|\tWT\t|\tRT\t|\n");
    for(i=0;i<n;i++)
    {
        printf("|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\n",PID[i],AT[i],BT[i],CT[i],TAT[i],WT[i],RT[i]);
    }
    printf("\nGantt chart is:\n ");
    for(i=0;i<t-1;i=i+2)
    {
        printf("%d ",GC[i]);
        if(GC[i+1]!=(-1))
          printf("(P%d) ",PID[GC[i+1]]);
        else
          printf("(NULL) ");

    }
    printf("%d \n\n",GC[t-1]); //last term of Gantt Chart
    for(i=0;i<n;i++) //loop to find sum of Response time
    {
        sumRT=sumRT+RT[i];
    }
    printf("\tAverage Turn Around Time is %f\n",((float)sumTAT/n));
    printf("\tAverage Waiting Time is %f\n",((float)sumWT/n));
    printf("\tAverage Response Time is %f\n",((float)sumRT/n));
    return 0;
}
```

References

[1] K. Eldahshan and A. A. Elkader, "Round Robin based Scheduling Algorithms, A Comparative Study," *Automatic Control and System Engineering Journal,* vol. 17, no. 2, pp. 29-42, 2017.

[2] Neha and A. Jiyani, "An Improved Round Robin CPU Scheduling Algorithm," *IRE Journals,* vol. 1, no. 9, pp. 82-86, 2018.

[3] S. Aliyu, S. E. Abdullahi, A. M. Mustapha and S. E. Abdullahi, "An Additional Improvement in Round Robin (AAAIRR) CPU Scheduling Algoritm," *International Journal of Advanced Research in Computer Science and Software Engineering ,* vol. 4, no. 2, pp. 601-610, 2014.

[4] J. Khatri, "An Enhanced Round Robin CPU Scheduling Algorithm," *IOSR Journal of Computer Engineering (IOSR-JCE) ,* vol. 18, no. 4, pp. 20-24, 2016.

[5] A. Sharma and G. Kakhani, "Analysis of Adaptive Round Robin Algorithm and Proposed Round Robin Remaining Time Algorithm," *International Journal of Computer Science and Mobile Computing,* vol. 4, no. 12, pp. 139-147, 2015.

[6] K. ElDahshan, A. A. Elkader and N. Ghazy, "Achieving Stability in the Round Robin Algorithm," *International Journal of Computer Applications,* vol. 172, pp. 15-20, 2017.

[7] D. F. Rashid and M. Kumar, "An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum," *International Journal of Computer Science, Engineering and Applications (IJCSEA) ,* vol. 4, no. 4, pp. 1-8, 2014.

[8] S. Chowdhury, "Survey on various Scheduling Algorithms," *imperial journal of interdisciplinary research(IJIR),* vol. 3, no. 5, pp. 1749-1752, 2017.

[9] M. Shoaid and M. Z. Faraoqui, "A Comparative Review of CPU Scheduling Algorithms," *Proceedings of National Conference on Recent Trends in Parallel Computing (RTPC - 2014),* pp. 20-28, 2014.

[10] I. Qureshi, "CPU Scheduling Algorithms: A Survey," *Int. J. Advanced Networking and Applications,* vol. 05, no. 04, pp. 1968-1973, 2014.

[11] A. Noon, A. Kalakech and S. Kadry, "A New Round Robin Based Scheduling Algorithm for," *IJCSI International Journal of Computer Science Issues,* vol. 8, no. 3, pp. 224-229, 2011.

[12] N. Mittal, K. Garg and A. Ameria, "A Paper on Modified Round Robin Algorithm," *International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS),* vol. IV, no. XI, pp. 93-98, 2015.

[13] K. Baskaran, "Modified Round Robin Scheduling Algorithm by Dynamic Time Quantum," *International Journal of Creative Research Thoughts (IJCRT),* vol. 5, no. 4, pp. 1654-1660, 2017.

[14] R. J. Matarneh, "Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes," *American Journal of Applied Sciences ,* pp. 1831-1837, 2009.

[15] A. E. Ale Agha and S. J. Jassbi, "A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean (HARM)," *International Journal of Information Technology and Computer Science,* pp. 56-62, 2013.

[16] M. U. Farooq, A. Shakoor and A. B. Siddique, "An Efficient Dynamic Round Robin Algorithm for CPU scheduling," *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE),* 2017.

[17] S. Sabha, "A Novel and Efficient Round Robin Algorithm with Intelligent Time Slice and Shortest Remaining Time First," *Materials today,* pp. 12009-12015, 2018.

[18] A. Joshi, S. B. Goyal and K. Sharma, "A Modified Version of Round Robin Algorithm “Modulo Based Round Robin Algorithm”," *International Journal of Advanced Science and Technology,* vol. 29, pp. 576-578, 2020.

[19] K. Faizan Ali, A. Marikal and K. Anil Kumar, "A Hybrid Round Robin Scheduling Mechanism for," *International Journal of Computer Applications,* vol. 177, pp. 14-19, 2020.

[20] R. Kumar Yadav, A. K Mishra, N. Prakash and H. Sharma, "An Improved Round Robin Scheduling Algorithm for CPU scheduling," *International Journal on Computer Science and Engineering,* vol. 2, pp. 1064-1066, 2010.

[21] A. singh, P. Goyal and S. Batra, "An Optimized Round Robin Scheduling Algorithm," *International Journal on Computer Science and Engineering,* vol. 2, pp. 2383-2385, 2010.

MY CONTRIBUTION (19BCE0316):

I collected all the data available in reference papers which was based on our topic analysis of modified round robin algorithms like IRR, AAIRR, Enhanced RR, RRRT ,Enriched RR and its implementation methods and how it is different and effective in computing the results better compared to traditional approach ,method ,and created SURVEY table ,and given necessary inputs required to my teammates based on our proposed round robin algorithm like picking few mathematical approaches(not all) from many reference papers (nearly 25) available which are helpful my teammates to compute all the above mentioned algorithms like writing code and analyzing all the algorithms using graphs and to produce minimum waiting time, minimum response time and maximize throughput by our proposed algorithm compared to traditional approach.

And helped my teammates in writing necessary codes ,flowchart of our proposed algorithm and analyzing using graphs in order to make our project very effective**.**