**Improved Round Robin Using Sorting Algorithm to Enhance Waiting and Turnaround Time**

Andres, Brandon Jules T.

Governor Pack Road, Baguio City

Philippines 2600

reaperrevo33@gmail.com

As-il, Jhaeus Herzian T.

Governor Pack Road, Baguio City

Philippines 2600

jhaeus8@gmail.com

Baleña, Julius Bernadin S.

Governor Pack Road, Baguio City

Philippines 2600

juliusbalena9@gmail

Castillo, John Ariel Dominic J.

Governor Pack Road, Baguio City

Philippines 2600

rendingscissor@gmail.com

**ABSTRACT**

The Improved Round Robin was created to make use of a Sorting Algorithm in order to improve the Average Waiting Time and Average Waiting Time of processes. In this paper, the researchers will discuss their methods on what type of Sorting Algorithm was used. The test results show that Improved Round Robin Algorithm was unsuccessful due to some factors.

**Keywords**

Round Robin Scheduling Algorithm; Sorting Algorithm; Waiting Time; Turnaround Time.

1. **INTRODUCTION**

Scheduling algorithms are used to organize work for the CPU to process. These scheduling algorithms are needed so that time spent waiting CPU will not be wasted during its free time. The Robin Scheduling Algorithm is a fair job-scheduling algorithm that makes use of time slicing which is used to assign processes in a queue which is then processed in a circular FIFO order for a certain amount of time in the CPU until finished. This eliminates starvation since all processes will be processed regardless of priority. Sorting algorithms are used for reorganizing items into a specific order either alphabetical, increasing order, decreasing order, etc. Sorting algorithms can be used in various ways that involve organization. The goal of this research is to improve an existing scheduling algorithm by adding a sorting algorithm.

The processes will be sorted in by twos from smallest to biggest. Hypothetically this would make the Round Robin algorithm’s Wait and Turnaround Time shorter because it will further decrease the starvation of each process.

1. **LITERATURE REVIEW**

A study on Sorting Algorithms was created to see which type is algorithm is best in sorting. It was found that Insertion Sort and Quicksort were the most optimal Sorting Algorithms and that the combination of the two can yield good results as well. [1] (Cook & Kim, 1980)

For our study we will adapt a pseudo ‘command and conquer’ type Sorting Algorithm since it is the most efficient.

1. **METHODOLOGY**

4.1. *Pseudocode:*

1. Check if the process is >2 if not continue to step 3 else go to step 2;

2. Start sorting the array

2.a. Get the values of the sorted array, alternatingly store the [max and min burst time] until you end up on its middle value;

2.b. If the total number of elements in the array is an odd number the [single] middle value will be at the last index;

3. Quantum time of the process, subtract the maximum burst time with the minimum burst time of the current process and divide it by two then return its value, if process size is <=2 return the minimum burst time;

4. Start to calculate the waiting time for each process

4.a. Create an empty array with the same size as the process to store the waiting time of each process, a temporary array with the same size and value as the process, and another variable to store up time value;

//While loop

4.b. An integer variable checker will be created with a value of 1;

//for loop

4.c Check if the current burst time of the selected process is greater than 0 [> 0]; if yes, the value of the checker will be set to [0] then it will continue to 5.d; if not go to step 4.e;

4.d If the current burst time of the selected process has a greater value than the quantum time, then quantum time will be subtracted to the current burst time [process[current] - qt] and the quantum time will be added to time variable [time += qt], continue to step 4.e;

If false, the remaining burst time value of the current process will be added to time variable [time += process[current]], then the waiting time array with the same index as the process will store a value of time subtracted to the original burst time value of the current process[waiting time[current] = time - process[current]];

Set the burst time of the current process to 0;

4.e If the loop is over, go to step 4.f, if false go to the next process and execute step 4.c

//checker

4.f If the checker has a value of 0 go to step 4.e, if the checker has a value of 1 go to step 5;

5. Start to calculate the turnaround time for each process

5.a Create an empty array with the same size of the process to store the turnaround time of each process;

//for loop

5.b current turn around process will get the sum of the current waiting time and current burst time [turnAround[current] = waitingTime[current] + process[current] repeat until last process then go to step 6;

6. Display the result

6.a Print the process value, burst time, waiting time, average time repeat until last process.

6.b Display the average waiting time, summation of all waiting time divided to number of process [totalWaitingTime/totalNumberOfProcess]

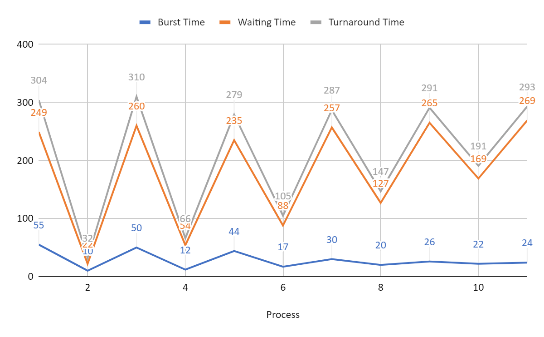
6.c Display the average turnaround time, summation of all turnaround time divided to number of process [totalTurnAroundTime/totalNumberOfProcess]

1. **FIGURES/CAPTION**

Both the Improved and Original Algorithms will run process of random values. The Waiting Time and Turnaround Time will be averaged and the algorithm whose value is greater than the other means that that algorithm has a slower Waiting and Turnaround time.

**TABLE 1.** TEST CASE 1 IMPROVED RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 55 | 249 | 304 |
| 2 | 10 | 22 | 32 |
| 3 | 50 | 260 | 310 |
| 4 | 12 | 54 | 66 |
| 5 | 44 | 235 | 279 |
| 6 | 17 | 88 | 105 |
| 7 | 30 | 257 | 287 |
| 8 | 20 | 127 | 147 |
| 9 | 26 | 265 | 291 |
| 10 | 22 | 169 | 191 |
| 11 | 24 | 269 | 293 |

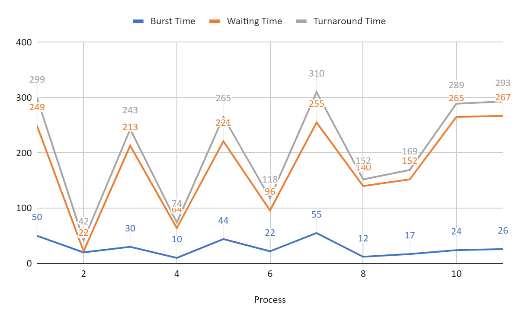
**Figure 1. Results for Test Case 1 IRR**

Average Waiting Time: 181.36

Average Turn Around Time: 209.55

**TABLE 2.** TEST CASE 2 ORIGINAL RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 50 | 249 | 299 |
| 2 | 20 | 22 | 42 |
| 3 | 30 | 213 | 243 |
| 4 | 10 | 64 | 74 |
| 5 | 44 | 221 | 265 |
| 6 | 22 | 96 | 118 |
| 7 | 55 | 255 | 310 |
| 8 | 12 | 140 | 152 |
| 9 | 17 | 152 | 169 |
| 10 | 24 | 265 | 289 |
| 11 | 26 | 267 | 293 |

**Figure 2. Results for Test Case 1 ORR**

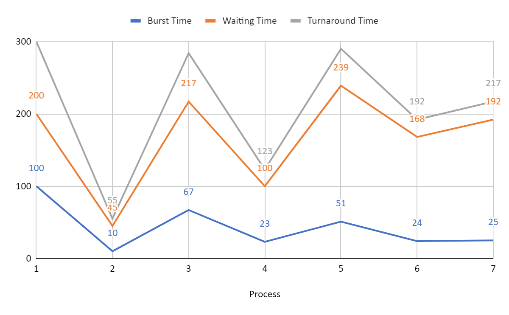
Average Waiting Time: 176.73

Average Turn Around Time: 204.91

For Test Case 1, both the Improved and Original algorithm ran 11 processes with random values. As the results show the Original RR has the Improved beat in Average Waiting Time and Turnaround time by a relatively small margin.

**TABLE 3.** TEST CASE 2 IMPROVED RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 100 | 200 | 300 |
| 2 | 10 | 45 | 55 |
| 3 | 67 | 217 | 284 |
| 4 | 23 | 100 | 123 |
| 5 | 51 | 239 | 290 |
| 6 | 24 | 168 | 192 |
| 7 | 25 | 192 | 217 |

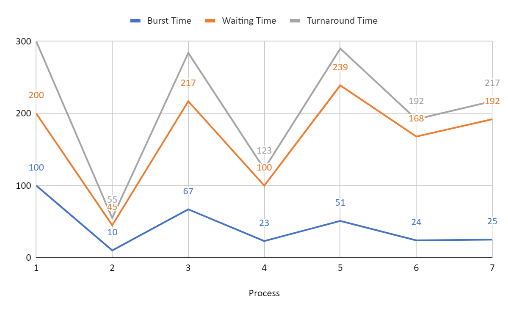
**Figure 3. Results for Test Case 2 IRR**

Average Waiting Time: 165.86

Average Turn Around Time: 208.71

**TABLE 4.** TEST CASE 2 ORIGINAL RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 24 | 0 | 24 |
| 2 | 51 | 172 | 223 |
| 3 | 10 | 69 | 79 |
| 4 | 23 | 79 | 102 |
| 5 | 67 | 178 | 245 |
| 6 | 25 | 147 | 172 |
| 7 | 100 | 200 | 300 |

**Figure 4. Results for Test Case 2 ORR**

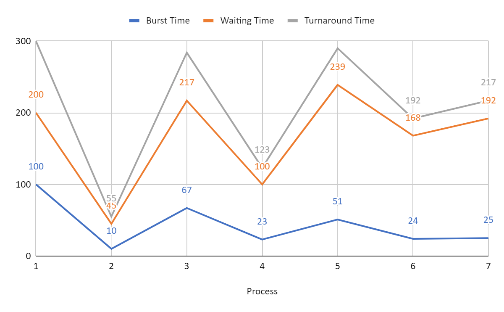
Average Waiting Time: 120.71

Average Turnaround Time: 163.57

For Test Case 2, both Improved and Original ran 7 processes with random values. The Original, once again, bested the Improved algorithm. The AWT of the Improved is slower by 45.15 compared to the Original. The Improved Algorithm’s ATT is also slower by 45.14.

**TABLE 5.** TEST CASE 3 IMPROVED RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 100 | 492 | 592 |
| 2 | 5 | 47 | 52 |
| 3 | 98 | 498 | 596 |
| 4 | 10 | 99 | 109 |
| 5 | 94 | 481 | 575 |
| 6 | 12 | 156 | 168 |
| 7 | 56 | 528 | 584 |
| 8 | 13 | 215 | 228 |
| 9 | 49 | 537 | 586 |
| 10 | 23 | 275 | 298 |
| 11 | 45 | 298 | 343 |
| 12 | 23 | 275 | 298 |
| 13 | 34 | 366 | 400 |
| 14 | 34 | 400 | 434 |

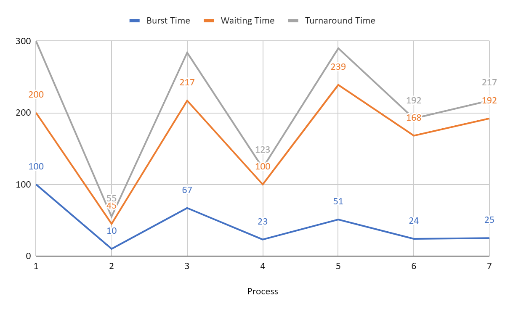
**Figure 5. Results for Test Case 3 IRR**

Average Waiting Time: 338.21

Average Turn Around Time: 380.79

**TABLE 6.** TEST CASE 3 ORIGINAL RR

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turnaround Time |
| 1 | 100 | 492 | 592 |
| 2 | 10 | 47 | 57 |
| 3 | 5 | 57 | 62 |
| 4 | 34 | 62 | 96 |
| 5 | 13 | 96 | 109 |
| 6 | 94 | 434 | 528 |
| 7 | 56 | 481 | 537 |
| 8 | 12 | 203 | 215 |
| 9 | 49 | 490 | 539 |
| 10 | 23 | 262 | 285 |
| 11 | 34 | 285 | 319 |
| 12 | 23 | 319 | 342 |
| 13 | 45 | 342 | 387 |
| 14 | 98 | 498 | 596 |

**Figure 6. Results for Test Case 1 ORR**

Average Waiting Time: 290.57

Average Turn Around Time: 333.14

For Test Case 3, both Improved and Original ran 14 processes with random values. The Original RR is still faster than the Improved RR with its AWT slower by 47.64 and ATT slower by 49.65.

The tests show that the Improved Round Robin Algorithm is slower than the Original by a small margin. This means that the Improved Algorithm with the Sorting Algorithm did not enhance the Original Round Robin Algorithm. The organization of the processes may have increased the number of times each process needed to be worked.

1. **CONCLUSION**

In this study, the Round Robin Scheduling Algorithm was supposed to be improved by adding a Sorting Algorithm. However, the test results show that the algorithm did not improve its Average Waiting and Turnaround Time.

The researchers recommend to improve the Sorting Algorithm that they created. The sorting may have an effect on the processes at hand and it may need more tweaking. The researchers did not use a specific type of Sorting Algorithms. The Test cases were at random so that may be a factor in its failure.

1. **ACKNOWLEDGEMENT**

The researchers would like to thank ACM SIGCHI for allowing us to modify templates they had developed. The researchers would also like to thank Rey Benjamin M. Baquirin for aiding them in the creation of this paper.

1. **REFERENCES**

[1] Cook, Curtis R. and Kim, Do Jin, 1980

Magazine Communications of the ACM

https://dl.acm.org/citation.cfm?id=359026

(Nov. 1980) 620-624