



Algorithms and Analysis COSC 1285
Assignment 1: Process Scheduler

	Assessment Type	Group assignment. Submit online via Canvas → Assignments → Assignment Task 3: Assignment 1 → Assignment 1: Process Scheduler . Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums.
	Due Date	Week 6, Wednesday 8th April 2020, 11:59pm
	Marks	15

1 Objectives

There are three key objectives for this assignment:

- Understand how a real problem can be mapped to various representations and their operations.
- Use a number of fundamental data structures to implement the *runqueue* abstract data type.
- Evaluate and contrast the performance of the data structures with respect to different usage scenarios and input data.

2 Learning Outcomes

This assignment assesses learning outcomes CLO 2, 4 and 5 of the course. Please refer to the course guide for the relevant learning outcomes: <http://www1.rmit.edu.au/courses/004302>

3 Overview

Scheduling is the process of arranging, controlling and optimizing tasks and workloads. In computing, scheduling concerns about which task is assigned to resources that complete the task. The task may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards [2]. Scheduling is an intrinsic part of the execution model of a computer system. It makes it possible to have computer multitasking with a single central processing unit (CPU). For more detailed information about scheduling, please refer to [2].

A process scheduler carries out the scheduling activity in an operating system. It arranges active processes in a *runqueue*. The *runqueue* represents a timeline for the

scheduled processes. For example, the Completely Fair Scheduler (CFS) is a process scheduler for the 2.6.23 (October 2007) release of the Linux kernel. Schedulers like the CFS keep track of time (in nanoseconds) a process has executed on processor, called *vruntime*. In this assignment, we will implement and analysis CFS types of schedulers that keep track of *vruntime* of a process. Processes that has got a small amount of time are boosted to the front of the *runqueue* to get the processor, those who got bigger amount of time are thwarted [1]. When the scheduler is invoked to run a new process: the process with the **lowest** *vruntime* from the *runqueue* is chosen, and sent for execution, then

1. If the process simply completes execution, it is removed from the *runqueue*
2. Otherwise, if it is interrupted for a reason, it is reinserted into the *runqueue* based on its new *vruntime*. If there are multiple processes has the same *vruntime*, they follow a First-In-First-Out (**FIFO**) ordered.

This type of schedulers commonly require efficient data structure implementation of the *runqueue* to schedule new process, delete process, track process information and high-performance code to generate the schedules. Despite the name, the *runqueue* need not be implemented in the traditional way, e.g., as an array. In this assignment, we defines the *runqueue* as an abstract data type similar to a PriorityQueue and consider the following two representations:

- The Sequential Representation
- The Tree Representation

We will implement both representations for the *runqueue* and evaluate how well they perform when representing some given *runqueues* and compute the average speed of various operations.

4 Tasks

The assignment is broken up into a number of tasks, to help you progressively complete the project.

4.1 Task A: Implement the *runqueue* representations and its operations (8 marks)

In this task, you will implement the *runqueue* using linear and tree representations, respectively. Each representation will be implemented by various data structures. Each element of the *runqueue* represents a process to be scheduled, containing a process label, and its *vruntime*.

4.1.1 Data Structure Details

In this assignment, each element of the *runqueue* (i.e., process) will be represented with an abstract data type *Proc* that consists of the following information:

- **procLabel** - a string label of the process, i.e., the unique identifier of a process. The format of the process label starts with letter P followed by digits, e.g., P1, P240, P1000, etc.
- **vt** - vruntime of the process. For the sake of simplicity, we use time units (positive integers) to represent vruntime in this assignment.

The priority of each element/process is assessed by its vruntime. Process that has a **shorter** vruntime should be of **higher** priority, i.e., should be dequeue/removed before other processes that have a longer vruntime.

The *runqueue* can be implemented using a number of data structures. In this assignment, you are to implement the *runqueue* using the following representations and data structures:

- Sequential Representation
 - using an 1D Ordered Array
 - using an Ordered Linked-list
- Tree Representation
 - using a Binary Search Tree (BST)

Note that you must program your own implementation, and not use existing libraries in any kind, e.g. the PriorityQueue, the LinkedList, Set, MultiSet, Tree type of data structures in java.util or any other libraries. You must implement your own nodes and methods to handle the operations. If you use java.util or other implementation from libraries, this will be considered as an invalid implementation and attract 0 marks for that data structure.

4.1.2 Operations Details

Operations to perform on the implemented *runqueue* abstract data type are specified on the command line. They are in the following format:

<operation> [arguments]

where operation is one of {EN, DE, FP, RP, PT, ST, PA, Q} and arguments is for optional arguments of some of the operations. The operations take the following form:

- **EN <procLabel> <vt>** – Enqueue: create and add a process (with label **procLabel** and vruntime **vt**) into the *runqueue*. There is no output for this operation.
- **DE** – Dequeue: delete the process with the highest priority (i.e., shortest vruntime) from the *runqueue*. Processes that have the same vruntime follow the **FIFO** order. If the *runqueue* is not empty, the output should be the label of the deleted process. Otherwise, it outputs an empty string.
- **FP <procLabel>** – Find the process with the given label **procLabel** in the *runqueue*. The output should be **true** if the search is successful, otherwise **false** (i.e., the process does not exist).

- **RP** <procLabel> – Remove the process with the given label **procLabel** in the *runqueue*. The output should be **true** if successfully deleted, otherwise **false** (i.e., the process does not exist).
- **PT** <procLabel> – Calculate the total vruntime of the *preceding* processes of the process labelled as **procLabel**. Preceding processes are the processes that will be executed before the given process. If the process with **procLabel** does not exist in the *runqueue*, return -1; otherwise, the output is the calculated total vruntime.
- **ST** <procLabel> – Calculate the total vruntime of the *succeeding* processes of the process labelled as **procLabel**. Succeeding processes are the processes that will be executed after the given process. If the process with **procLabel** does not exist in the *runqueue*, return -1; otherwise, the output is the calculated total vruntime.
- **PA** – Print the labels of all the processes in the *runqueue* in ascending order of their vruntime, i.e., the time line. Please refer to the following example for the exact format of the output. If the *runqueue* is empty, this operation prints out an empty string.
- **Q** – quits the program.

As an example of the operations, consider an initial empty *runqueue* and the following list of operations (inputs) and their associated outputs:

```
[ Int : ] EN P2 2
[ Out : ]
[ Int : ] EN P1 1
[ Out : ]
[ Int : ] EN P3 2
[ Out : ]
[ Int : ] DE
[ Out : ] P1
[ Int : ] DE
[ Out : ] P2
[ Int : ] FP P2
[ Out : ] false
[ Int : ] RP P2
[ Out : ] false
[ Int : ] EN P1 1
[ Out : ]
[ Int : ] EN P2 4
[ Out : ]
[ Int : ] EN P4 4
[ Out : ]
[ Int : ] EN P5 5
[ Out : ]
[ Int : ] PA
[ Out : ] P1 P3 P2 P4 P5
[ Int : ] PT P2
```

```

[Out:] 3
[Int:] ST P2
[Out:] 9
[Int:] FP P2
[Out:] true
[Int:] RP P2
[Out:] true
[Int:] PT P2
[Out:] -1
[Int:] ST P2
[Out:] -1
[Int:] PA
[Out:] P1 P3 P4 P5

```

4.1.3 Testing Framework

We provide Java skeleton code (see Table 1) to help you get started and test the correctness of your code. You may add your own Java files to your final submission, but please ensure that they work with the supplied framework (see below).

file	description
<code>RunqueueTester.java</code>	Code for testing (details see below). <i>No need to modify this file.</i>
<code>Runqueue.java</code>	Interface for <code>Runqueue</code> . All implementations should implement the <code>Runqueue</code> class defined in this file. Read the javaDoc of each method carefully and <i>do not modify this file</i> .
<code>OrderedArrayRQ.java</code>	Code that implements the sequential representation of <i>runqueue</i> using an 1D <i>ordered</i> array. Complete the implementation (implement parts labelled “Implement me!”).
<code>OrderedLinkedListRQ.java</code>	Code that implements the sequential representation of <i>runqueue</i> using an <i>ordered</i> linked list. Complete the implementation (implement parts labelled “Implement me!”).
<code>BinarySearchTreeRQ.java</code>	Code that implements the tree representation of <i>runqueue</i> using an binary search tree. Complete the implementation (implement parts labelled “Implement me!”).

Table 1: Table of Java files.

We provide the Java file `RunqueueTester.java` that helps with automating testing. Once completed the implementations of the *runqueues* and compiled the necessary files, you can test your implementation using the `RunqueueTester` in either interactive or non-interactive mode:

Interactive mode reads in operation commands from `stdin` then executes those on the selected *runqueue* implementation.

```
java RunqueueTester <impl>
```

where `<impl>` is one of the implementation, `<impl> = [array | linkedlist | tree]`

For example:

```
java RunqueueTester array
```

The above command executes the `RunqueueTester` with the 1D ordered array implementation of the *runqueue* in the interactive mode. Please refer to the input output example in Section 4.1.2

Non-interactive mode reads input files of operations (such as example above). These are fed into the Java framework which calls your implementations. The outputs resulting from any print operations are stored. The format of the command call upon the non-interactive mode is as follows:

```
java RunqueueTester <impl> inputfile_name outputfile_name
```

For example:

```
Java RunqueueTester array input.in output.out
```

The above command executes the `RunqueueTester` with the 1D ordered array implementation of the *runqueue*. It reads the commands from `input.in` and stores all the output to `output.out`. We provide two pairs of input and expected output files for your testing and examination.

For our evaluation on the correctness of your implementation, we will use the same Java file `RunqueueTester.java` for auto testing, as well as input/expected output files of the same format as the provided examples. To avoid unexpected failures, please do not change the `RunqueueTester.java` file. If you wish to use the java file for your timing evaluation, make a copy and use the unaltered script to test the correctness of your implementations, and modify the copy for your timing evaluation.

Notes

- We will run the supplied test file on your implementation on the university's core teaching servers. If you develop on your own machines, please ensure your code compiles and runs on these machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing.
- All submissions should compile with no warnings on **OpenJDK 1.8** - this is the default `javac/java` version on the Core teaching servers.

4.2 Task B: Evaluate your Data Structures (7 marks)

In this second task, you will evaluate your three implemented structures both theoretically and empirically in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible use cases of a process scheduler.

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each type of implementation would be most appropriate. The report should be **6 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

4.2.1 Use Case Scenarios

There are many possibilities, but for this assignment, consider the following scenarios:

- **Scenario 1 Growing *runqueue* (Enqueue).** In this scenario, the *runqueue* is growing with increasing processes are adding into the *runqueue*. In this scenario, you are to evaluate the performance of your implementations in terms of the enqueue operation (EN).
- **Scenario 2 Shrinking *runqueue* (Dequeue).** In this scenario, the *runqueue* is shrinking with increasing processes are dequeuing from the *runqueue*. In this scenario, you are to evaluate the performance of your implementations in terms of the dequeue operation (DE).
- **Scenario 3 Calculating total vruntime of proceeding processes.** In this scenario, assume the *runqueue* is not changing, but important operation of calculating total **vruntime** of proceeding processes of a given process is required. In this scenario, you are to evaluate the performance of your implementations in terms of the PT operation.

4.2.2 Theoretical Analysis

At first, you are to conduct a theoretical analysis of the performance. Given a *runqueue* of size n , report the best case, worse case running time estimate, and the asymptotic complexity (Big O) of your implementations in terms of the different scenarios outlined above. You will also need to provide an example scenario and explanation on the each of the best case and worse case running time estimate, e.g. using an *runqueue* with a small number (e.g.,5) of elements/processes. Put the results in the follow format of a table:

Theoretical Analysis			
Scenarios	Best Case	Worse Case	Big O
Scenario 1	[time estimation] [example and explanation]	[time estimation] [example and explanation]	[asymptotic complexity]
Scenario 2	[time estimation] [example and explanation]	[time estimation] [example and explanation]	[asymptotic complexity]
Scenario 3	[time estimation] [example and explanation]	[time estimation] [example and explanation]	[asymptotic complexity]

4.2.3 Empirical Analysis

Typically, you use real usage data to evaluate your data structures. However, for this assignment, you will write data generators to enable testing over different scenarios of interest.

Data Generation

The time performance of the above mentioned scenarios and operations could vary significantly depends on the structure and elements/processes of the initial *runqueue*.

We provide an artificial processes dataset of about 5000 processes, each with a label (string) and a vruntime (integer number) in a file called “processes.txt”. Each line of this file stores a process in the format of: label,vruntime

For generating *runqueues* with different initial structure and elements, you may want to generate a series of add process operations (EN) to grow the *runqueue* to the desired size, by either:

- randomly choosing processes from the file we supplied (‘processes.txt’); or
- writing a process generator to generate a new process. For example, for each process, generate a unique string identifier/process label and a random integer (i.e. vruntime) uniformly sampled within a fixed range, e.g., [1, 100].

Note, whichever method you decide to use, remember to generate *runqueues* of different sizes to evaluate on (e.g., 10, 50, 100, 500, ..., 1000, 5000). Due to the randomness of the data, you may wish to generate a significant number of datasets with the same parameters settings (same size and a scenario) and take the average across a number of runs. In your analysis, you should evaluate each of your representations and data structures in terms of the different scenarios outlined above. Hence, we advise you to get started on this part relatively early.

5 Report Structure

As a guide, the report could contain the following sections:

- Theoretical analysis on running time and complexities of the different data structure implementation as outlined in Section 4.2.2.
- Explain your data generation and experimental setup. Things to include are (brief) explanations of the generated data you decide to evaluate on, the complexity parameters you tested on, describe how the scenarios were generated (a paragraph and perhaps a figure or high level pseudo code suffice), which approach(es) you decide to use for measuring the timing results.
- Evaluation of the data structures using the generated data. Analyse, compare and discuss your results across different complexity parameters, representations and scenarios. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation.
- For some particular data structure like Binary Search Tree (BST), you will also need to suggest possible improvement to reduce running time on different scenarios. In order to do that, you will need to understand well the complexity of various operations on a BST, and be able to research beyond.
- Summarise your analysis as recommendations, e.g., “for this certain data scenario of this parameters setup, I recommend to use this data structure because....”. We suggest you refer to your previous analysis for help.

5.1 Clarification to Specifications

Please periodically check the assignment FAQ for further clarifications about specifications. In addition, the lecturer will go through different aspects of the assignment each week, so even if you cannot make it to the lectures, be sure to check the course material page on Canvas to see if there are additional notes posted.

6 Team Structure

This assignment is designed to be done in *pairs* (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. Any issues (e.g., work division or workload) that result within the team should be resolved within the team if possible; if this is not possible, then this should be brought to the attention of the course coordinators as soon as possible. Marks are awarded to the individual team members, according to the contributions made towards the final work. Please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution $X\%$, and student B has contribution $Y\%$, and $X > Y$. The group is given a group mark of M . Student A will get M for assignment 1, but student B will get $\frac{M}{X}$.

Teams need to be formed using the self sign up tool in Canvas.

7 Submission

The final submission will consist of four parts:

- Your **Java source code** of your implementations. Your source code should be placed into in a flat structure, i.e., all the files should be in the same directory/folder, and that directory/folder should be named as **Assign1-<partner 1 student number>-<partner 2 student number>**. Specifically, if your student numbers are s12345 and s67890, then all the source code files should be in folder Assign1-s12345-s67890.
- Your **written report for part B** in PDF format, called “assign1.pdf”. Place this pdf within the Java source file directory/folder, e.g., Assign1-s12345-s67890.
- Your **data generation code**. Create a sub-directory/sub-folder called “generation” within the Java source file directory/folder. Place your generation code within that sub-folder. We will not run the code, but will examine their contents.
- Your group’s **contribution sheet**. See the following ‘Team Structure’ section for more details. This sheet should also be placed within your Java source file folder.
- Then, the Java source file folder (and files within) should be zipped up and named as **Assign1-<partner 1 student number>-<partner 2 student number>.zip**. E.g., if your student numbers are s12345 and s67890, then your submission file should be called **Assign1-s12345-s67890.zip**, and when we unzip that zip file, then all the submission files should be in the folder Assign1-s12345-s67890.

8 Assessment

The project will be marked out of 15. Late submissions will incur a deduction of 1.5 marks per day, and no submissions will be accepted 5 days beyond the due date.

The assessment in this project will be broken down into two parts. The following criteria will be considered when allocating marks.

Implementation (8/15):

- Your implementation will be assessed on the different implemented data structures, respectively, and on the number of tests it passes in our automated testing.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence these factors will contribute towards your marks.

Report (7/15):

The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your evaluation report. Use the criteria and the suggested report structure (Section 4) to inform you of how to write the report.

Late Submission Penalty: Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late. Submissions that are late by 5 days or more are not accepted and will be awarded zero, unless special consideration has been granted. Granted Special Considerations with new due date set after the results have been released (typically 2 weeks after the deadline) will automatically result in an equivalent assessment in the form of a practical test, assessing the same knowledge and skills of the assignment (location and time to be arranged by the coordinator). Please ensure your submission is correct (all files are there, compiles etc), re-submissions after the due date and time will be considered as late submissions. The core teaching servers and Canvas can be slow, so please do double check ensure you have your assignments done and submitted a little before the submission deadline to avoid submitting late.

Assessment declaration: By submitting this assessment, all the members of the group agree to the assessment declaration - <https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/assessment-declaration>

9 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the following: <https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity>.

10 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you

have with your Tutor or Lab Demonstrator. We will also be posting common questions on the assignment 1 Q&A section on Canvas and we encourage you to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic and data structure design.

A Marking Guide for the Report

Design of Evaluation (Maximum = 1.5 marks)	Analysis of Results (Maximum = 4 marks)	Report Clarity (Maximum = 1.5 marks)
<p>1.5 marks</p> <p>Data generation is well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of size of the <i>runqueue</i> were evaluated. Each type of test was run over a number of runs and results were averaged.</p>	<p>4 marks</p> <p>Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and different input sizes of the <i>runqueue</i>. All analysis, comparisons and conclusions are supported by empirical evidence and theoretical complexities. Well reasoned recommendations are given.</p>	<p>1.5 marks</p> <p>Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.</p>
<p>1 marks</p> <p>Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable size of the <i>runqueue</i>. Each type of test was run over a number of runs and results were averaged.</p>	<p>3 marks</p> <p>Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and input sizes of the <i>runqueue</i>. Most analysis and comparisons are supported by empirical evidence and theoretical analysis. Reasonable recommendations are given.</p>	<p>1 marks</p> <p>Clear and structured for the most part, with a few unclear minor sections.</p>
<p>0.5 mark</p> <p>Data generation is somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures or reasonable size of the <i>runqueue</i>. Each type of test may only have been run once.</p>	<p>2 marks</p> <p>Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and sizes of the <i>runqueue</i>. A portion of analysis and comparisons are supported by empirical evidence and theoretical analysis. Adequate recommendations are given.</p>	<p>0.5 mark</p> <p>Generally clear and well structured, but there are notable gaps and/or unclear sections.</p>
<p>0 marks</p> <p>Data generation is poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures or reasonable size of the <i>runqueue</i>. Each type of test has only have been run once.</p>	<p>1 mark</p> <p>Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and size setting. Little analysis and comparisons are supported by empirical evidence and theoretical analysis. Poor or no recommendations are given.</p>	<p>0 marks</p> <p>The report is unclear on the whole and the reader has to work hard to understand.</p>

References

- [1] Completely fair scheduler. https://en.wikipedia.org/wiki/Completely_Fair_Scheduler.
- [2] Scheduling (computing). [https://en.wikipedia.org/wiki/Scheduling_\(computing\)](https://en.wikipedia.org/wiki/Scheduling_(computing)).