Report – Algorithm & Analysis, Assignment 01

Rajesh Choudhary – s3705430
Suraj Jaju – s3705440

TASK B:
Theoretical Analysis

This task required us to evaluate and analyse our 3 implemented structures i.e. Sequential (Ordered 1D Array and Ordered Linkedlist) and Tree Representations theoretically in terms of their complexities for the different operations performed and use case scenarios.

Scenario 1 – Growing *runqueue* (enqueue).

Here, the *runqueue* is growing as processes are being added in the *runqueue*. The addition of the processes via the INTERACTIVE mode attaches the processes to the end of the array and the linkedlist using the enqueue().

Scenario 2 – Shrinking *runqueue* (dequeue).

In this scenario, the runqueue is shrinking in size as the processes are being dequeued from the sequential representations of array and linkedlist. Using the INTERACTIVE mode, the deletion of the processes from the data structures is carried out using the dequeue().

Scenario 3 – Calculating total *vruntime* of proceeding processes.

Given the scenario, we were told to assume that the runqueue was not changing and evaluate the performance of the implementations of data structures in terms of PT operation.

Theoretical Analysis			
Scenarios	**Best Case**	**Worse Case**	**Big O**
1 (Enqueue)	**<u>Best-case :- O(1)</u>** E.g. :- Given an empty data structure, the best-case would be that it is empty and the process to be added has to be added at the start of the DS.	**<u>Worst-case :- O(n)</u>** E.g. :- Considering the runqueue of 5 processes, the worst-case scenario is when a new process has to be appended at the end traversing the whole list of processes.	O(1)
2 (Dequeue)	**<u>Best-case :- O(1)</u>** E.g. :- In a list of certain processes, the process chosen to be dequeued, has the lowest runtime i.e. higher priority. Explanation – As the chosen process has a higher priority, it is removed before the other processes.	**<u>Worst-Case :- O(n)</u>** E.g. :- The elements with higher vruntime have to wait to be removed. Explanation – As the processes with lower runtime have a higher priority, the processes with higher runtime have to wait.	O(1)
3 (PT)	**<u>Best-case :- O(n)</u>** E.g. The best-case scenario here is that the process chosen to calculate the runtimes after is at the last position in the data structure. Explanation – Here, there are no processes after the chosen process to calculate the vruntime.	**<u>Worst-case :- O(1)</u>** E.g. The worst-case scenario in this case is the exact opposite of the best-case one. Here, the chosen process is located at the start of the data structure. Explanation – Here, the function has to traverse through the entire data structure to give correct results.	O(1)

Experimental Setup

Data Generation

For generating data structures comprising of various sizes, we made a program Experiment.java which takes in the processes like, 5, 10, 100, 1000, 5000 at random and then calculates the total time in nanoseconds for the operations Enqueue (EN), Dequeue (DE) and PT (calculate the total runtime).

Sizes of Data

We generated several files for processes 10, 50, 100, 500, 1000, 2000, 5000 for both the representations, Sequential as well as Tree Representations. We used these sizes in order to compare the time the functions required to perform on the processes. And then we chose 3 categories like LOW – 10, MEDIUM – 500, and HIGH – 5000.

Generation of scenarios

We decided to analyse all text files generated, whose data was generated randomly for both, Sequential Representation as well as Tree Representation. Complexities chosen for all trees were the same for getting better results.

Evaluation

Scenario 1: Growing runqueue (Enqueue)

By observing the figures 1 & 2, it can be concluded that Enqueue for BST is more efficient when compared to sequential representation of Array & Linked List, as the time taken to perform the enqueue operation of 5000 processes is far less than the other representation.

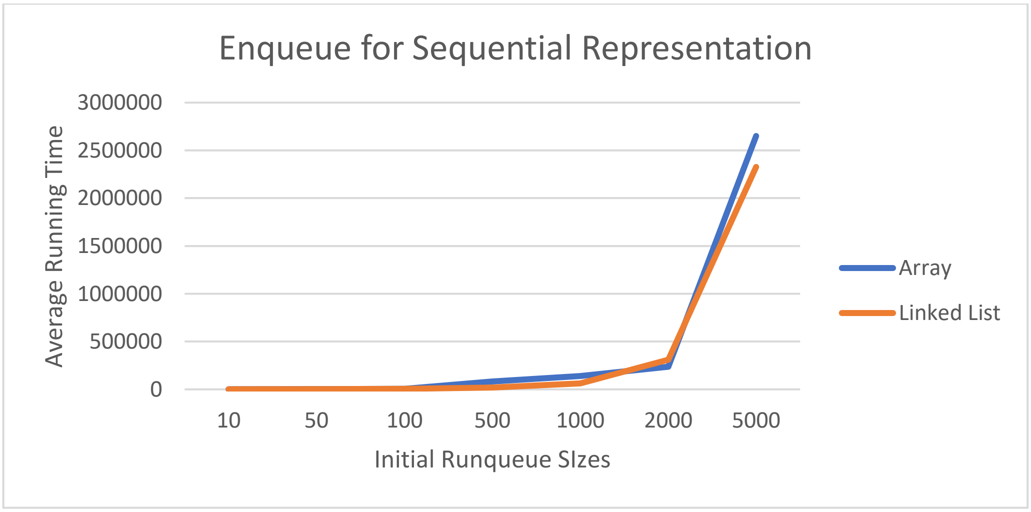
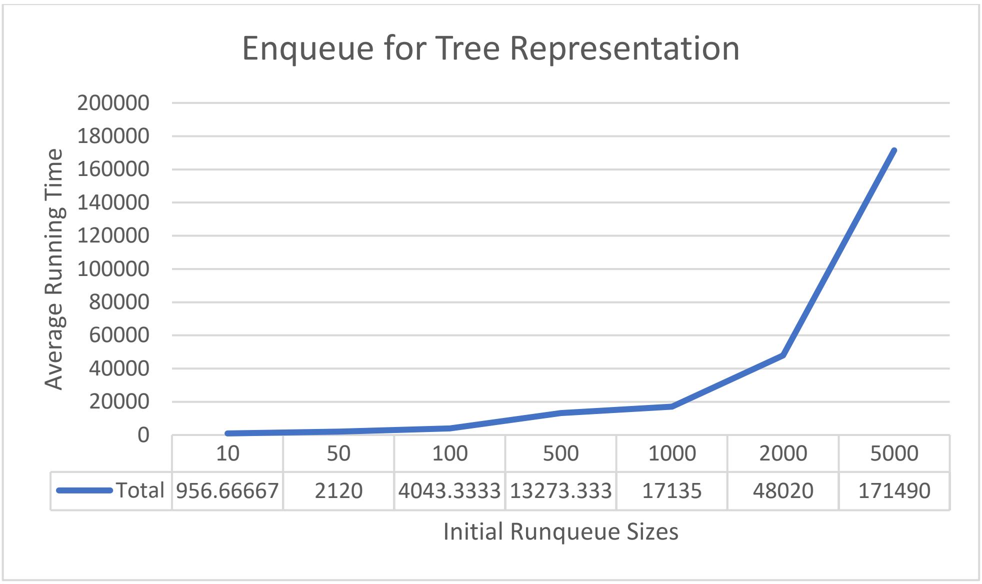


Figure 1: Average runtime for enqueue in sequential representation



	10	50	100	500	1000	2000	5000
Total	956.66667	2120	4043.3333	13273.333	17135	48020	171490

Figure 2: Average runtime for enqueue in tree representation

<u>Scenario 2: Shrinking runqueue (Dequeue)</u>

By observing the figures 3 & 4, it can be concluded that Dequeue for Array in sequential representation is much more efficient followed by Linked List when compared to tree representation of BST, as the time taken to perform the dequeue operation of 5000 processes is far less than the other representation.

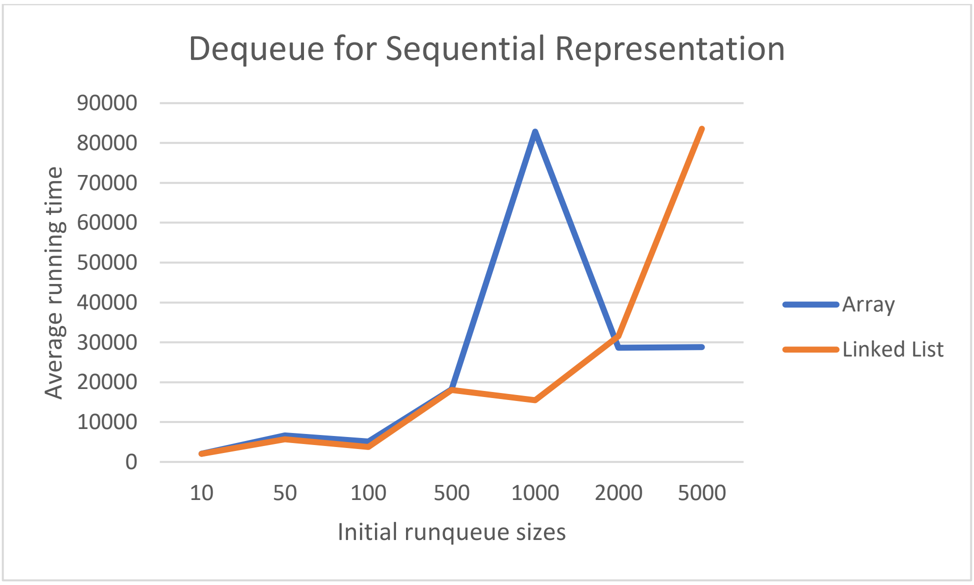


Figure 3: Average runtime for dequeue in sequential representation

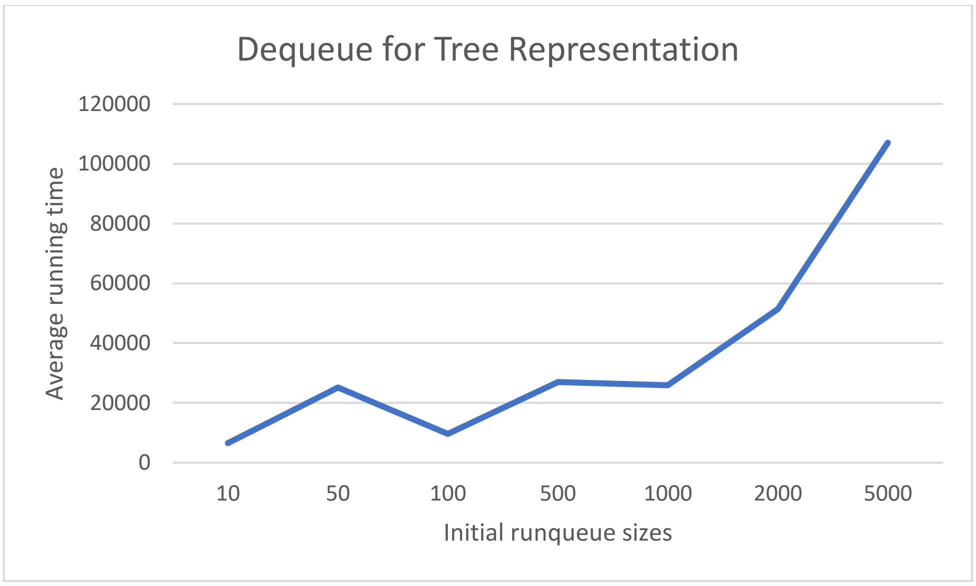


Figure 4: Average runtime for dequeue in tree representation

Scenario 3: Calculating total vruntime of proceeding processes

By observing the figures 5 & 6, it can be concluded that calculating the vruntime for Linked List in sequential representation is much more efficient followed by Array when compared to tree representation of BST, as the time taken to perform the dequeue operation of 5000 processes is far less than the other representation.

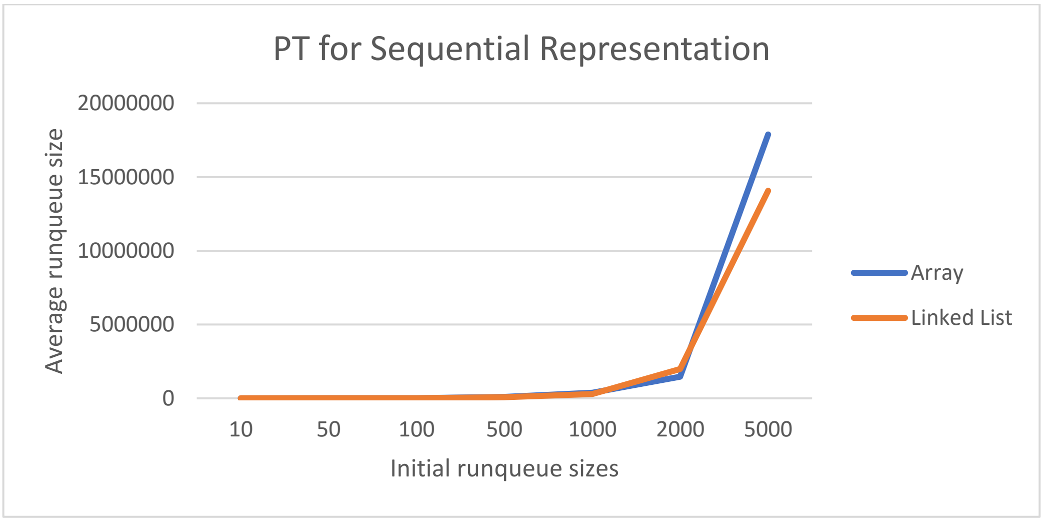


Figure 5: Average runtime for PT in sequential representation

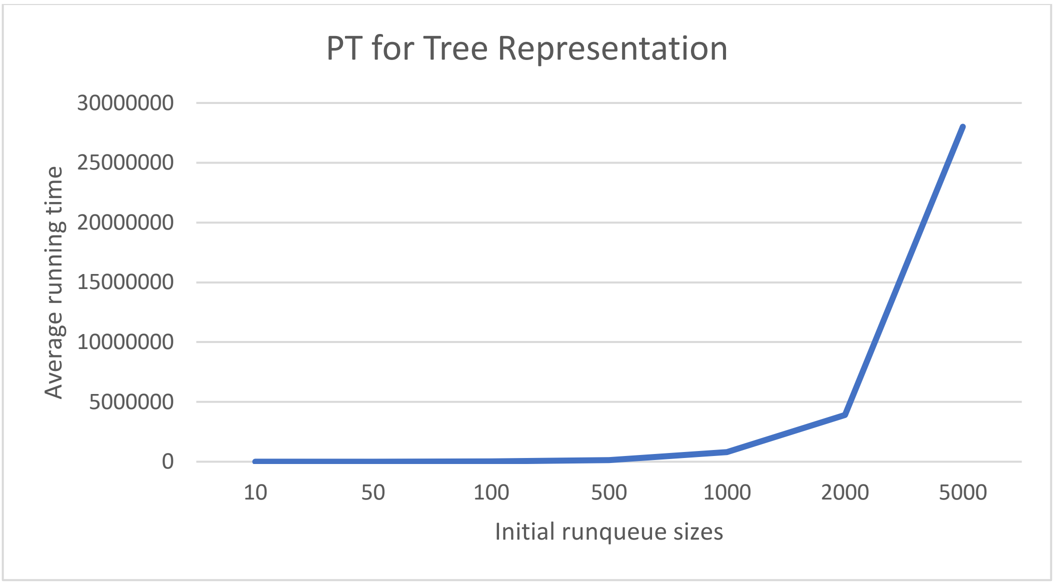


Figure 6: Average runtime for PT in tree representation

Recommendations

Finally, from the observed results, it can be concluded that there is no single data structure that can be used for all the given scenarios. It becomes much easier to understand the optimal data structure depending on the use case (scenario).