A Heap or Priority Queue Report.

Adrian Jonsson Sjödin

Fall 2022

1 Introduction

2 Task

The task in this assignment is to create a priority queue using different implementations and determine the pros and cons of each. The different priority queues to be implemented are listed bellow.

- List implementation of priority queue where smaller numbers have higher priority. Make one implementation where the cost of adding elements are $\mathcal{O}(1)$ and remove is $\mathcal{O}(n)$, and another where the costs are reversed. Benchmark the two implementations and discuss situations where one would be preferred over the other.

- Priority queue implementing a heap as a linked binary tree where each node holds the element, branches and the total number of elements in the subtree. This to make the tree balanced. You should be able to add and remove elements. Also implement a method `push(Integer incr)` that increments the root element by `incr` and then pushes it (by swapping values) to either the left or right branch.

  Set up a benchmark where you first add 64 elements with random values to a heap. The run a sequence of push operations where you increment a by a random number ($\in [10, 20]$). Collect some statistics on how deep the push operations need to go.

  Change the add method so that it also returns the depth. Run the same benchmark as above but now collect statistics on the add method.

- Priority queue implementing a heap using an array. Benchmark the array implementation and see how it compares to the

- linked list implementation.

3 Method & Theory

4 Discussion

Code

All the code can be found here: GitHub

Code Overview 1: Add and remove in first linked list queue

Code Overview 2: Enqueue and dequeue in second linked list queue

Code Overview 3: The `next()` method for the iterator

Code Overview 4: Array implementation of `enqueue(Item item)`

Code Overview 5: Array implementation of `dequeue()`

Code Overview 6: The `resize` method