Heap Analysis

Matthew Galitz

University of Virginia, Charlottesville, VA 22904

Abstract.
The purpose of the experiment is to compare the performance of two MinHeap implementations using an ArrayList and Vector as the underlying data structure.

1 Introduction

1.1 Problem

The problem is to analyze the performance of different MinHeap implementations.

1.2 Description

The MinHeap data structure resembles a binary search tree in an abstract sense. The MinHeap must obey the following ordering principles: Every child node must be less than its parent, and nodes are added in a left-to-right fashion at the lowest level of the heap. MinHeap data is stored in a list. Index 0 is left empty while indexes 1 to n store the heap data, index 1 being the root node. The MinHeap is implemented using two underlying list structures: the ArrayList and the Vector. The former will be referred to as a ListHeap and the latter as a VectorHeap. Each implementation is evaluated on their runtimes for the push(), poll(), and peek() methods. The methods add an item to the heap, removes an item from the heap, and checks the next item to removed respectively.

2 Methods

The ListHeap and VectorHeap are evaluated for 10,000 items for 5 trials each; 10,000 items are pushed, polled, and peeked.

Trials Run for Each Experiment

	# of Trials
ListHeap	5
VectorHeap	5

The average of the data for each trial is calculated using the following equation

$$\boldsymbol{A = \frac{\sum_{n=1}^{N} x_n}{N}}$$

Where x_n is the time for trial n, N is the total number of trials, and A is the average time.

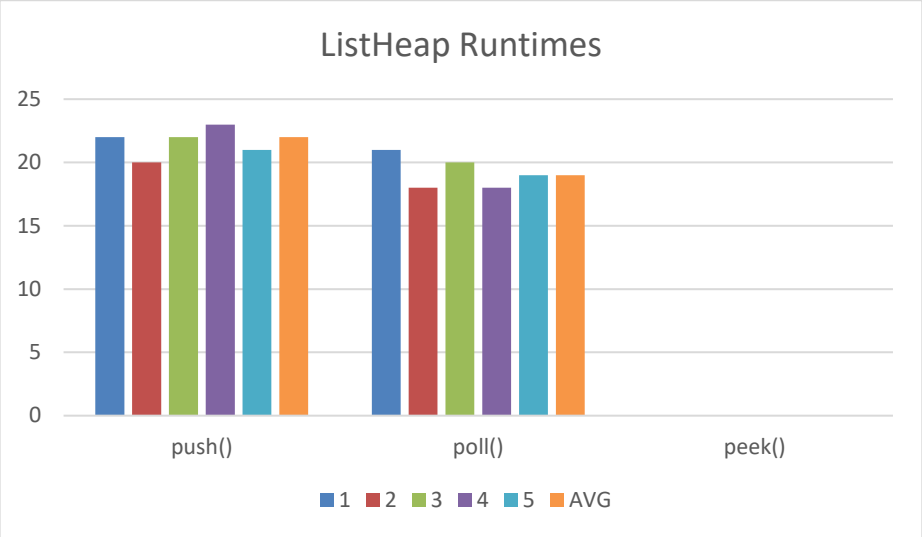
3 Results

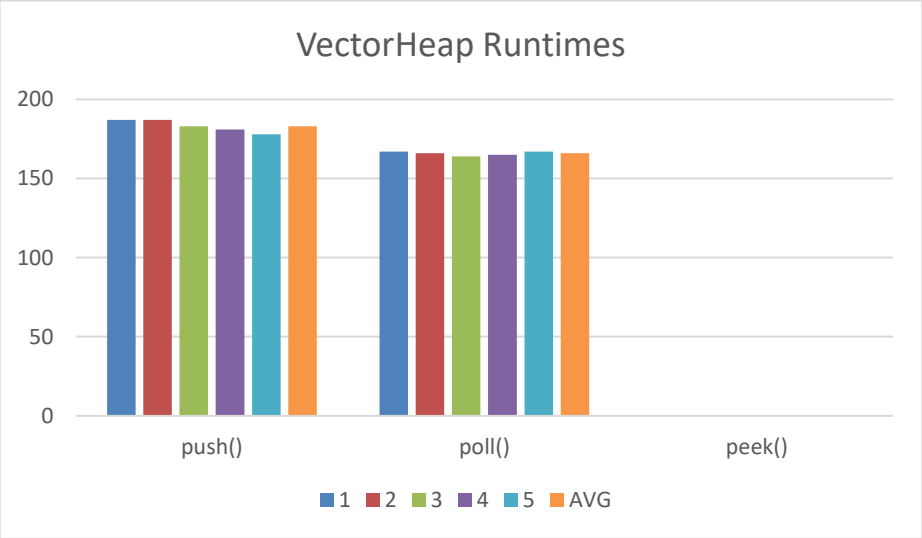
ListHeap Runtime

	1	2	3	4	5	AVG
push()	22	20	22	23	21	22
poll()	21	18	20	18	19	19
peek()	0	0	0	0	0	0

VectorHeap Runtime

	1	2	3	4	5	AVG
push()	187	187	183	181	178	183
poll()	167	166	164	165	167	166
peek()	0	0	0	0	0	0





4 Conclusion

The data indicates that the ListHeap was significantly faster than the VectorHeap with respect to the push() and poll() methods. The peek() method showed no discrepancy in runtime. This is a result of the underlying data structures used in each heap. An ArrayList is much faster than a Vector because it is an unsynchronized structure, such that it employs concurrency to achieve faster runtimes.