MaxHeapSort code analysis report

Overview

The code implements maxheap with methods

- **Insert - adds a child element**
- **GetMax – returns root element**
- **IncreaseKey – increases value of the element by some index**
- **MaxHeapify - sorts the finished maxheap**

1. Asymptotic Complexity

Time Complexity

- Insert: Worst O(logn), Best Θ(1), Average Θ(log n)
- GetMax: Θ(1)
- IncreaseKey: O(log n)
- MaxHeapify: O(log n)

Space complexity

- Heap Array uses Θ(n) space
- Recursion stack O(log n)
- Mostly in-place implementation

Recurrence relation

- MaxHeapify: $T(n) = T(n/2) + O(1) \rightarrow O(\log n)$

2. Code revies & Optimization

Inefficiencies

- Main operations are optimal

Space improvements

- Iterative MaxHeapify avoids recursion stack

3. Empirical Validation

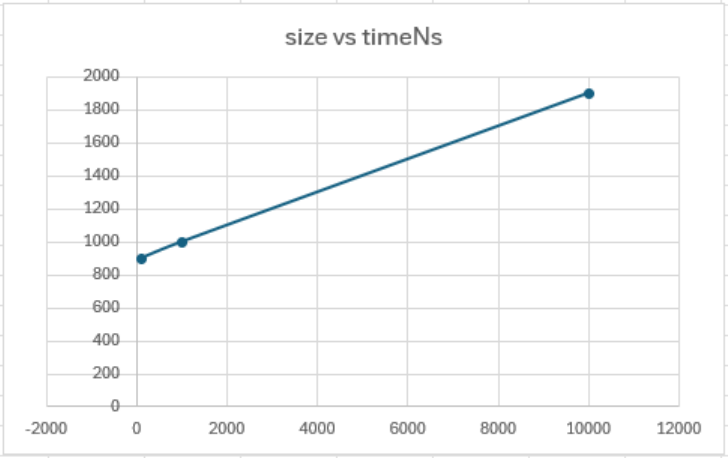
Perfomance expectations

- GetMax constant; Insert & ExtractMax grow logn
- Main operations much faster than O(n) sorting

Plot and measurments

operation	n	timeNs
Insert	100	900
GetMax	100	1100
IncreaseK	100	900
ExtractMa	100	300
Insert	1000	1000
GetMax	1000	800
IncreaseK	1000	1200
ExtractMa	1000	300
Insert	10000	1900
GetMax	10000	800
IncreaseK	10000	1200
ExtractMa	10000	500

operation	n	timeNs
Insert	100	900
	1000	1000
	10000	1900



Conclusion

Implementation is correct and mostly efficient. Time and space complexity are optimal for a heap. Improvements: iterative maxHeapify. This change will Improve performance for large heaps.