Why is Binary Heap Preferred over BST for Priority Queue?

geeksforgeeks.org/why-is-binary-heap-preferred-over-bst-for-priority-queue

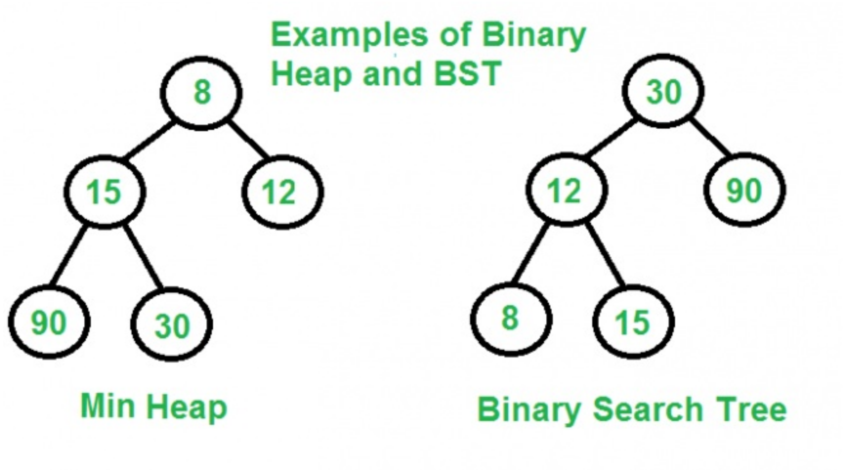
September 7, 2015

A typical Priority Queue requires following operations to be efficient.

1. Get Top Priority Element (Get minimum or maximum)
2. Insert an element
3. Remove top priority element
4. Decrease Key

A Binary Heap supports above operations with following time complexities:

1. O(1)
2. O(Logn)
3. O(Logn)
4. O(Logn)



A Self Balancing Binary Search Tree like AVL Tree, Red-Black Tree, etc can also support above operations with same time complexities.

1. Finding minimum and maximum are not naturally O(1), but can be easily implemented in O(1) by keeping an extra pointer to minimum or maximum and updating the pointer with insertion and deletion if required. With deletion we can update by finding inorder predecessor or successor.
2. Inserting an element is naturally O(Logn)
3. Removing maximum or minimum are also O(Logn)

4. Decrease key can be done in O(Logn) by doing a deletion followed by insertion. See this for details.

So why is Binary Heap Preferred for Priority Queue?

- Since Binary Heap is implemented using arrays, there is always better locality of reference and operations are more cache friendly.
- Although operations are of same time complexity, constants in Binary Search Tree are higher.
- We can build a Binary Heap in O(n) time. Self Balancing BSTs require O(nLogn) time to construct.
- Binary Heap doesn't require extra space for pointers.
- Binary Heap is easier to implement.
- There are variations of Binary Heap like Fibonacci Heap that can support insert and decrease-key in Θ(1) time

Is Binary Heap always better?
Although Binary Heap is for Priority Queue, BSTs have their own advantages and the list of advantages is in-fact bigger compared to binary heap.

- Searching an element in self-balancing BST is O(Logn) which is O(n) in Binary Heap.
- We can print all elements of BST in sorted order in O(n) time, but Binary Heap requires O(nLogn) time.
- Floor and ceil can be found in O(Logn) time.
- K'th largest/smallest element be found in O(Logn) time by augmenting tree with an additional field.

This article is contributed by **Vivek Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above