1 Question 2

a)

Answer

Let B be a BST that satisifies the heap ordering property which contains the given elements with the associated keys and priorities. Now all other valid BSTs with the same elements can be discovered by a series of rotations on B. However, we see that any basic rotation is performed on a parent and one of its children, resulting in the child becoming the parent instead. Thus, a rotation on B clearly violates our heap order property since the priority of all parents must be strictly less than those of its children (reversing ancestry reverses the priorities). Thus, we see that it is not possible to perform ANY rotation on B. All other valid BSTs consistutes of at least one rotation from B, which is not possible. Thus, there are no other BSTs that satisfy the heap order property besides B. Thus, there is an unique tree with the ordering requirements.

b)

Answer

From our result above, we know that there is a unique tree that satisifies both ordering requirements for some set of elements with associated keys and priorities. We make the observation that this tree can be created using only BST insertion (no rotations) if we insert elements in order of increasing priorities. This is because every insertion will place the node as a leaf of some parent, and since the parent was inserted before, its priority is strictly less than the node that was inserted, maintaining the heap ordering property. Since there is an unique tree, we know that even if we create the tree by inserting nodes in any order and performing rotations, we will arrive at this single, unique tree.

As the unique tree is "synonmous" to one created via repeated BST insertions in order of increasing priorities, if we can reason that the expected height of the tree is $O(\log n)$ when we insert in ascending priority order, then this expected height will generalize to the unique tree as well, as they are the same tree at the end.

Priorities are generated randomly. Thus, ordering the elements to be inserted by a value which is randomly generated leads to a random permutation of the set of elements to be inserted, and consequently, a random permutation of keys to be inserted. From CLRS, theorem 12.4, we know that a ranodmly built BST has an expected height of $O(\log n)$. Since we randomly build a tree with respect to the key using strictly BST insertion, we see that the expected height of our tree must be $O(\log n)$.

As the tree built by insertion in increasing priorities has height $O(\log n)$, the unique tree that satisifies both the ordering properties will also have expected height $O(\log n)$.

c)

Answer

To insert a new element e, we first insert it into the tree according to BST ordering requirements. This is simply $O(h)$ as it is standard BST insertion. Now we check the priority of e. If its priority is bigger than its parent, we return as the insertion is finished and the ordering requirements are met. Otherwise, we do a rotation with e and its parent p such that e is now the parent of p. We then repeat the procedure for e and its new parent until the heap ordering property is satisfied. At most h rotations can occur, each one raising e one level up until it reaches the root. Each rotation is constant time as it is just adjusting pointers. Thus, the total amount of work done is $h * O(1) + O(h) = O(h)$ in the worst case.

Algorithm 1 Insertion

```
1: procedure INSERT(B, key, ele)
2:     ele.pri ← PRIORITY()
3:     BSTINSERT(B, key, ele)
4:     while ¬isRoot(ele) ∧ ele.parent.pri > ele.pri do
5:         ROTATE(ele.parent, ele)
6:
```