Problem 4

Task: Worst case number of splits
The worst case number of splits is an element of $O(\log n)$. In fact, the very worst ratio of splits to keys is 1:6, when there is only one node with six children, and a single insertion causes a split. From then on, the worst case with two splits is one in which the root has six children and one of its children has six children, but the rest only have three. Then there are 21 nodes and 2 splits upon insertion. Generally, the worst case for some number of splits in terms of ration of keys to splits generally is one in which every node has three children except for a chain up to the root. In such cases, the number of splits is given by $\lfloor \log_3 n \rfloor$. Thus, the number of splits is bounded above by $\log_3 n$, and $\log_3 n \in O(\log n)$.

Task: Amortized cost of insertion
The costs that I consider are creating a leaf or a node, and changing the parent of a leaf or node, all of which have constant cost ($O(1)$). In the case of an insertion that causes a single split, a new leaf is created and has its parent set, then a new node is created, the new leaf and three old leaves have their parents changed to the new node, and the new node has its parent set. This is a total of 8 constant operations. Virtual cost, v, is 4 if the node being inserted into has 0 or 2 parents, 5 if the node inserted into has 3 or 4 parents, and 6 if the node inserted into has 5 or 6 parents. c is the actual cost of insertion. The balance changes at each step by $v - c$.

The table below shows the amortized analysis. The table, and my scheme for performing the analysis is a bit complex. n_β represents the number of nodes (NOT leaves) before insertion and n_α the number of nodes after. A node is represented by a number which indicates how many children it has. If the node is not terminal, it is followed by a set of parentheses surrounding its children nodes. For example, the node 2(4(3,3,3,4),4(3,3,3,4)) has two nodes under its root. Each of those nodes has 4 nodes underneath it, which contain 3, 3, 3, and 4 leaves under them, respectively. Each node has a separate balance, and the balances of the nodes are represented similarly. So a balance of 0(2(0,0,0,4),2(0,0,0,13)) indicates a root with a balance of 0, with two children both with balances 2, each of which has four children all with balances of 0 except one with a balance of 4 and one with a balance of 13. The sum of all the individual node balances is equal to the balance of the entire tree.

row	n_β	n_α	v	c	bal
1	3	4	4	2	2
2	4	5	4	2	4
3	5	6	4	2	6
4	6	2(3,4)	4	8	0(0,2)
5	2(3,4)	2(3,5)	4	2	0(0,4)
6	2(3,5)	2(3,6)	4	2	0(0,6)
7	2(3,6)	3(3,3,4)	4	8	0(0,0,2)
8	3(3,3,4)	3(3,3,5)	5	2	0(0,0,5)
9	3(3,3,5)	3(3,3,6)	5	2	0(0,0,8)
10	3(3,3,6)	4(3,3,3,4)	5	8	2(0,0,0,3)
11	4(3,3,3,4)	4(3,3,3,5)	5	2	2(0,0,0,6)
1	4(3,3,3,5)	4(3,3,3,6)	5	2	2(0,0,0,9)
12	4(3,3,3,6)	5(3,3,3,3,4)	5	8	4(0,0,0,0,4)
13	5(3,3,3,3,4)	5(3,3,3,3,5)	6	2	4(0,0,0,0,8)
14	5(3,3,3,3,5)	5(3,3,3,3,6)	6	2	4(0,0,0,0,12)
15	5(3,3,3,3,6)	6(3,3,3,3,3,4)	6	8	6(0,0,0,0,0,8)
16	6(3,3,3,3,3,4)	6(3,3,3,3,3,5)	6	2	6(0,0,0,0,0,12)
17	6(3,3,3,3,3,5)	6(3,3,3,3,3,6)	6	2	6(0,0,0,0,0,16)
18	6(3,3,3,3,3,6)	2(3(3,3,3),4(3,3,3,4))	6	16	0(0(0,0,0),2(0,0,0,10))
19	2(3(3,3,3),4(3,3,3,4))	2(3(3,3,4),4(3,3,3,4))	5	2	0(0(0,0,3),2(0,0,0,10))
20	2(3(3,3,4),4(3,3,3,4))	2(3(3,3,5),4(3,3,3,4))	5	2	0(0(0,0,6),2(0,0,0,10))
21	2(3(3,3,5),4(3,3,3,4))	2(3(3,3,6),4(3,3,3,4))	5	2	0(0(0,0,9),2(0,0,0,10))
22	2(3(3,3,6),4(3,3,3,4))	2(4(3,3,3,4),4(3,3,3,4))	5	8	0(2(0,0,0,4),2(0,0,0,10))

This table is sufficient to show that the "balance" will always be positive. From the first 3 inserts, you can generalize that a node's balance will remain positive if it has a balance of 0 and 1, 2, or 3 children, a balance of 2 and 4 children, a balance of 4 and 5 children, and a balance 6 and 6 children.

Rows 4-18 show that as a grandparent acquires more children, both it and its children will have enough balance distributed between them to meet or exceed the required balances listed above for each node given its number of chilren.

Row 18 shows the first possible recursive split. The total cost of 16 is given by a cost of 8 for the lowest level split, plus the cost of creating a new intermediate node and moving 4 nodes (5), plus the cost of creating a new root and moving the two top level nodes to it (3). Because the root stores enough of a balance given its number of children (bal = 6, chidren = 6), it is able to pay for the cost of splitting itself, and because each of its children has sufficient balance, they are guaranteed to be stay positive as well.

Task: Cost of insertion with lookup
The runtime of a lookup is an element of $O(\log n)$. At each node, between 3 and 6 keys need to be considered, which comes out to a constant operation. The number of nodes traversed is equal to the depth of the tree always (that is one of the invariants), which is an element

of $O(\log n)$ as discussed in the number of splits problem. Since the cost of insertion without lookup is constant (amortized), the cost of insertion with lookup is an element of $O(\log n)$.