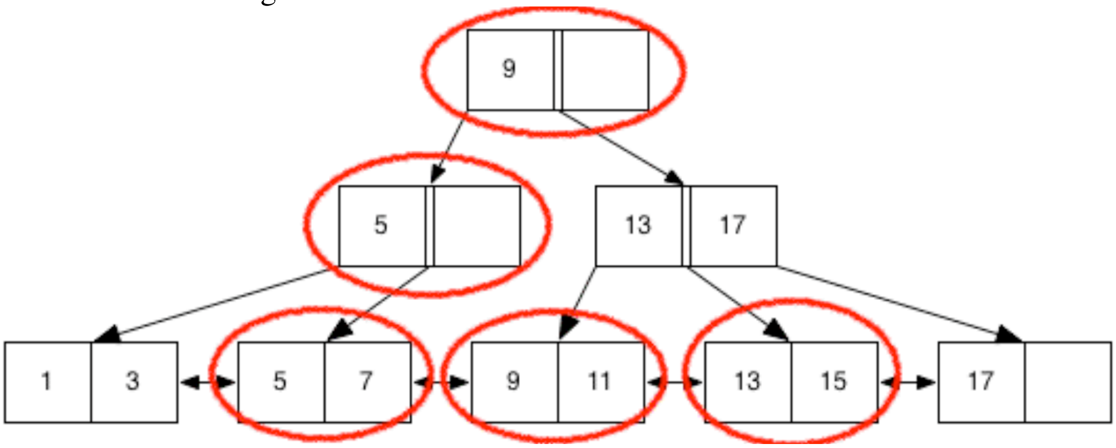
2. **B+ Trees [22 points]**

Consider the following B+ tree index of order 1:



a. *(2 points)* Circle all *nodes* (not index entries, but entire nodes) in the above figure that must be fetched to satisfy the query "Get all records with search key greater than or equal to 7 and less than 15".

   Explanation: In order to fetch records with search key greater than or equal to 7, you must first locate the first entry for 7. This requires fetching the root and traversing the tree down to the leaf containing 7. After this, you can use the fact that B+-trees have pointers between nodes (symbolized in the picture by the little two-way arrows) to traverse the tree that way.

   I took off a point for not circling the root, a point for circling (13,17) (if you only circled (13,17), I still only took off a point), and a point each for not circling correct nodes at the leaf level.

   Common mistakes included fetching the (13,17) node (which isn't necessary for range queries) and not fetching anything but the leaves.

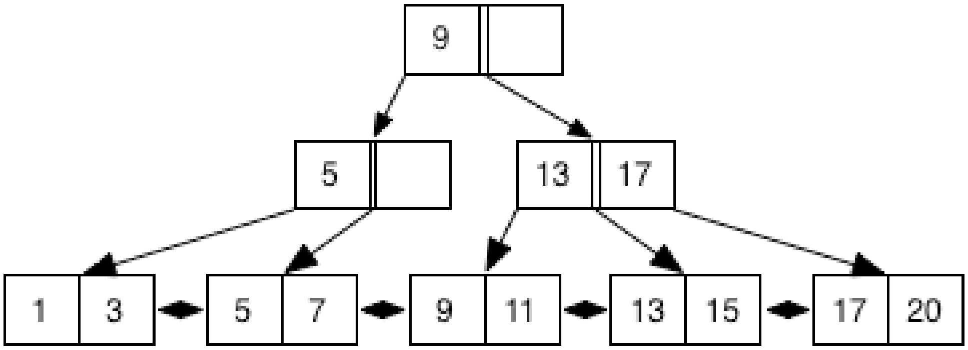
b. *(15 points)* Assume we modify the B+ tree by adding the following keys **in the following order**: **20, 27, 18, 30, 19**

   In the answer-boxes below, each row refers to a key being inserted in order, and each column asks if the insertion of that key results in a split of particular nodes. *Assume that when splitting up an odd number of entries, the left node gets one more than the right.* Place a Y mark in each box whose answer is “Yes”. Blank boxes will be interpreted as “No”.

   Since we had to correct ourselves (well, not really correct per se, we just had to put the question in line with the algorithm I taught my sections, which is one of a couple correct ones), we gave you the benefit of the doubt and allowed answers that assumed both an imbalance favoring the left on splits (which was what was given on the exam) and an imbalance favoring the right on splits (which was what we told you was the "correction" during the exam).

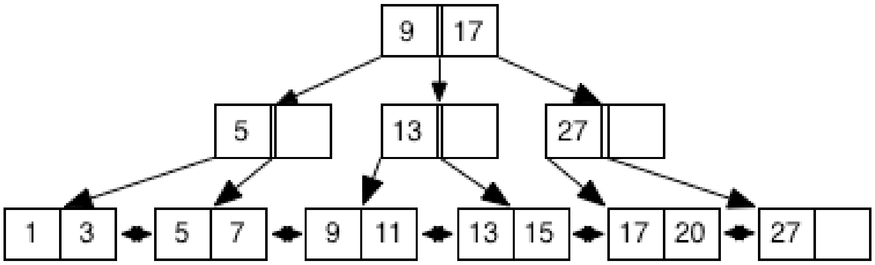
DISCLAIMER: I know that we said that the order in which you split shouldn't affect the solution. This was a mistake that would have been caught had any of us had the time to hash out the problem on the fly. Because we gave you some misinformation, I made sure that, had you answered correctly making either assumption, you would get full points.

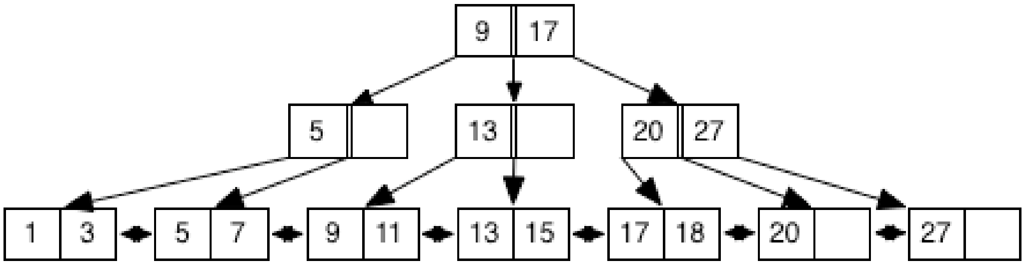
In either case, inserting 20 inserts 20 into the (17.-) node in the original tree, leaving the tree unmodified. This gives us a tree that looks like this:

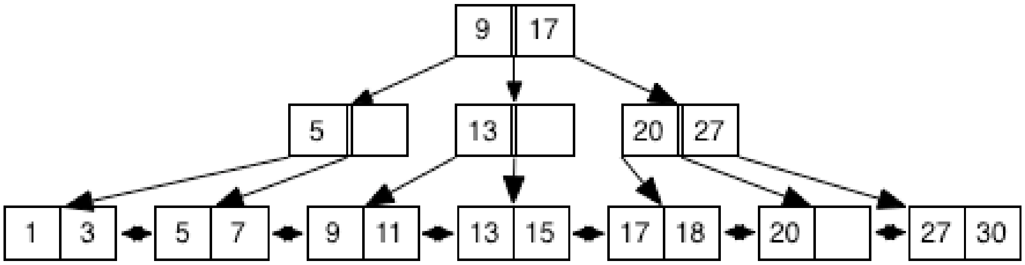


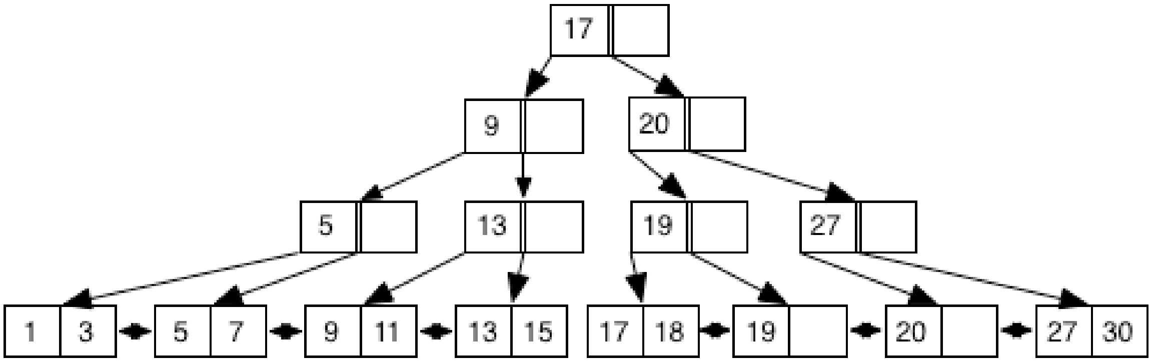
When inserting 27, both a leaf node and a non-leaf node (namely the (13,17) node) are split. They're just split in different ways. This causes a leaf split either when inserting 18 or when inserting 30, depending on which scheme you used. I gave you points either way.

Here are the trees after inserting each subsequent key, in sequence. Four more keys, so four more trees.

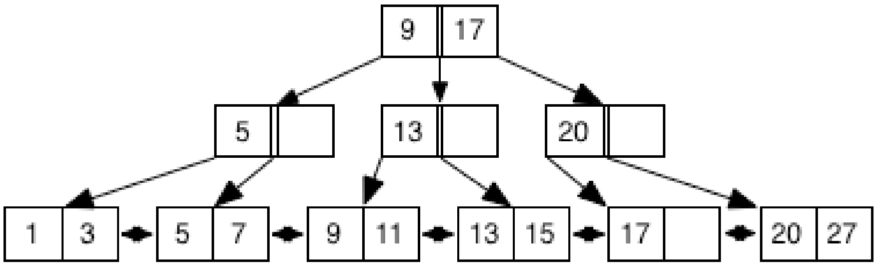


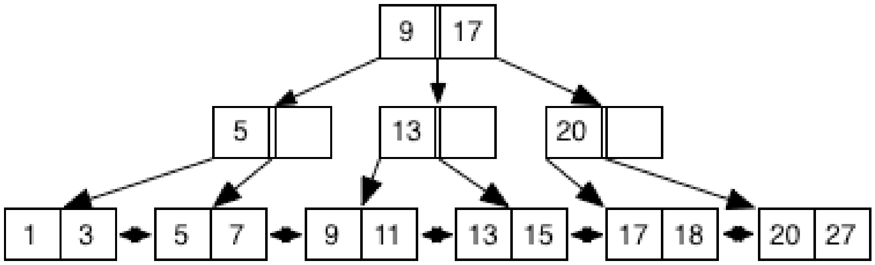


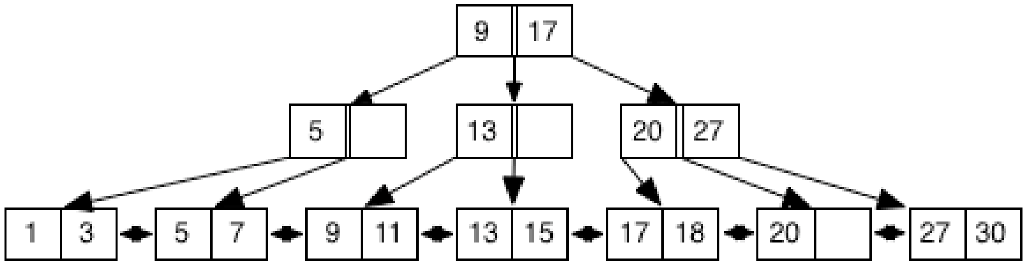


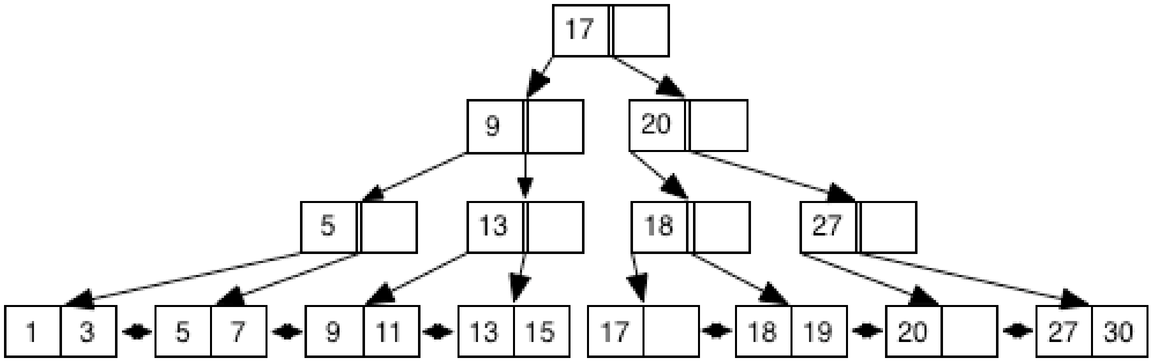


Assuming left gets one more than right









Assuming right gets one more than left

The answers, then, differ very little.

Here is the solution assuming that the left node gets one more than the right one on a split:

Key	Leaf Node Split?	Non-Leaf Split?	Root Split?
20			
27	Y	Y	
18	Y		
30			
19	Y	Y	Y

Here is the solution assuming that the right node gets one more than the left one on a split:

Key	Leaf Node Split?	Non-Leaf Split?	Root Split?
20			
27	Y	Y	
18			
30	Y		
19	Y	Y	Y

I deducted a point from each cell that didn't match either of these solutions, for a total of 15 deductions (if you happened to write the exact opposite of what I put in either grid, which is possible, but not probable). Answers were various.

c. *(5 points)* Suppose we were to insert all integers in the range 26 to 4112 inclusive (i.e. 26, 27, 28 ... , 143, 4112) into the tree in part (a), one at a time. *At most* how many levels would the resulting B+-tree have? (Hint: You should not need to draw a B+-tree to figure this out!)

Inserting 26 to 4112 inclusive inserts 4112-26+1 = 4087 keys into the tree. Add 9 existing keys to that and you get 4096, or 4 kilokeys.

The key to notice here is the *at most*. Since your tree has at least two pointers and at most three pointers, the tree is at its theoretical tallest when nodes with 2 pointers dominate its topology. Then you know from 61B that the tree is at most lg(4096) = 12 levels high. We took off 2 points if you were off by one or you took the log base 3 of a number close to 4096 instead of base 2. We took 5 points off if you were off by more than 1. Common mistakes were to not take the log or to take the log base some other number.