1.
   a) First off, this is reasonable because we will generally want faster algorithms that have lower runtimes. Secondly, it gives a reasonable way to compare different types of algorithms, despite the many differences in how they are implemented. Lastly, we only need to look at the largest order term in any given big-O formula, because everything else will not be relevant as input sizes increase. This includes constants and lower order terms.
   b) They ignore small input sizes because asymptotes only happen at large values of n.
   c) When we write O(n), O is actually the upper bound. This means that O(4n^3 + 1000n) is some value that 4n^3 + 1000n cannot cross or surpass. This results in n^4, since it is the next order of magnitude from n^3 if we ignore lower order terms and numerical constants.
   d) Array based lists are faster at searching/direct access, while linked lists are faster at inserting and removing.
   e) Push() and Pop() are two that come to mind, along with Top() and size(), although size is generic and doesn’t only apply to Stacks. Push adds an item to the top, top returns the item, and pop removes the item and returns it.
   f) You don’t have to keep shifting data values when using the FIFO method, because we can simply have two indices for the head and tail, making it faster and less complicated.
   **g) Check PAPER**
   h) Any element on the left of another node must be less than it. Any element on the right side must be greater. This applies to all elements from top to bottom, even if they aren’t directly related. For instance, the grandfather of a node on the left side must be less than the element(which is on the right), even though they are not directly related.
   i) An AVL Tree has insert times of log(n), which is smaller than BST’s insert time of n. This also applies to the contains and remove functions.
   j) Red-Black Trees have much faster insert and remove times due to the fact that they need fewer rotations for rebalancing.

2.
   a) This code has a big O of O(n^2). This is because there are two for loops which each go up to a max int value. There are also some other functions, but they do not matter because none of them are of a higher order than O(n^2).
   b) This code is O(n) because it only has one for loop.

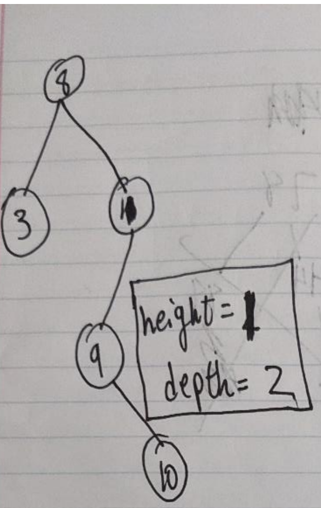
3.
```
Int Array x;
for(int i : n){
if(S1.top()!=S2.top()){
S3.push(S1.pop())
}

}
```

4. The order matters. Suppose you have this binary tree:
   6, (3, 9), (N, N, 7, N)
   If we want to delete 6 and 3 from this tree, we can do this in two orders. First lets try doing this with 6 first.
   We get 7, (3,9).

Then we delete 3 to get 7, (N, 9).
But if we do it the other way around, we will get 6, (N, 9), (N,N,7,N) and then 9, (7, N)
These are not the same trees, so a counterexample has been given.

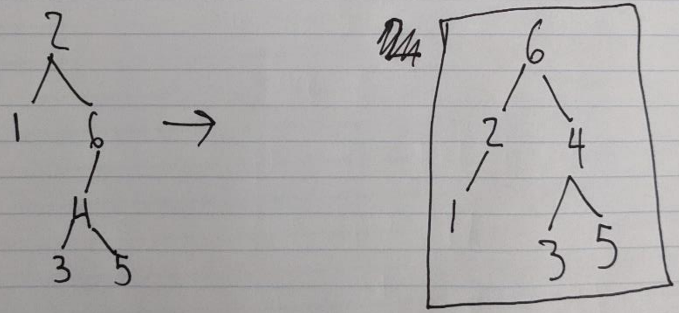
5. **Check PAPER**
6. 
   ```
   Int search(int keys, int key){
   If root == null
   Return null
   ```

1.g.

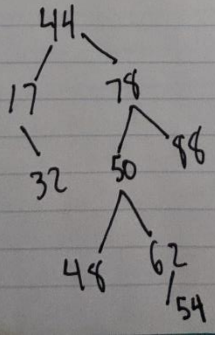


height = 1
depth = 2

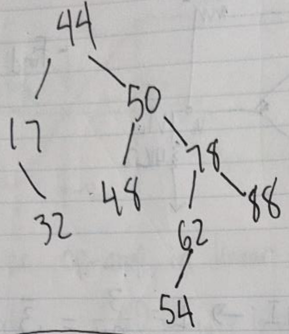
5a.

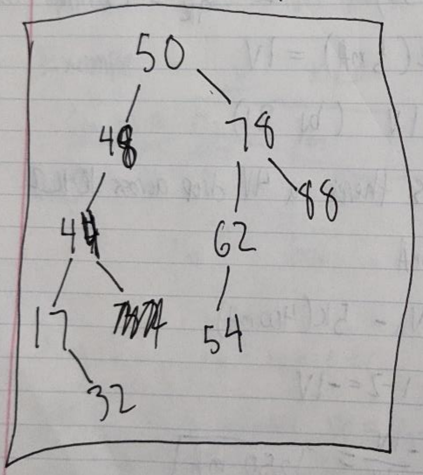


5b.



5b. cont.


44
17
50
32
48
78
62
88
54


50
48
78
44
62
88
17
54
32