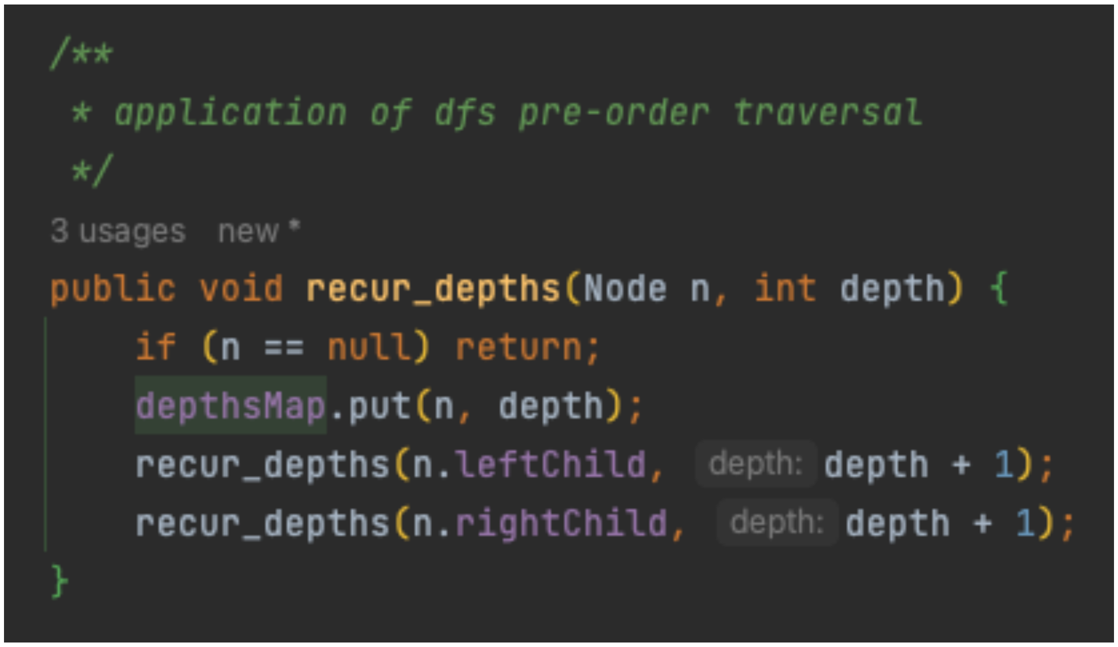
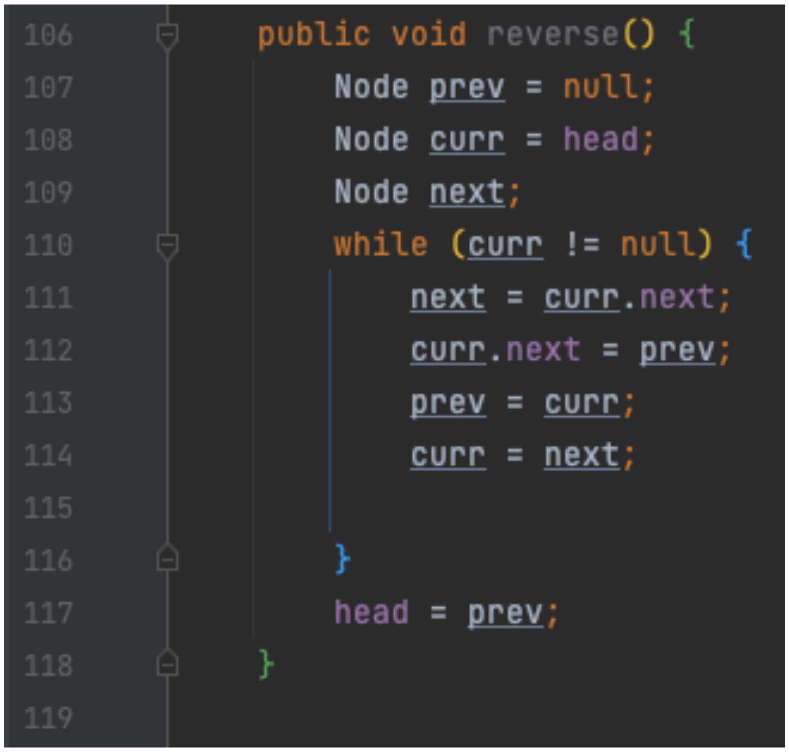
Lab 5

1. Working inside the BinaryTree class, write a method that will create a HashMap that associates with each node in the tree its depth in the tree. Hint. Make use of the depthfirst-search implementation of pre-order traversal.



```java
/**
 * application of dfs pre-order traversal
 */
3 usages  new *
public void recur_depths(Node n, int depth) {
    if (n == null) return;
    depthsMap.put(n, depth);
    recur_depths(n.leftChild, depth: depth + 1);
    recur_depths(n.rightChild, depth: depth + 1);
}
```

2. Devise an algorithm for reversing the elements in a singly linked list. Implement your solution in code. You can use the singly linked list in the lab folder. What is the running time of your reverse algorithm?



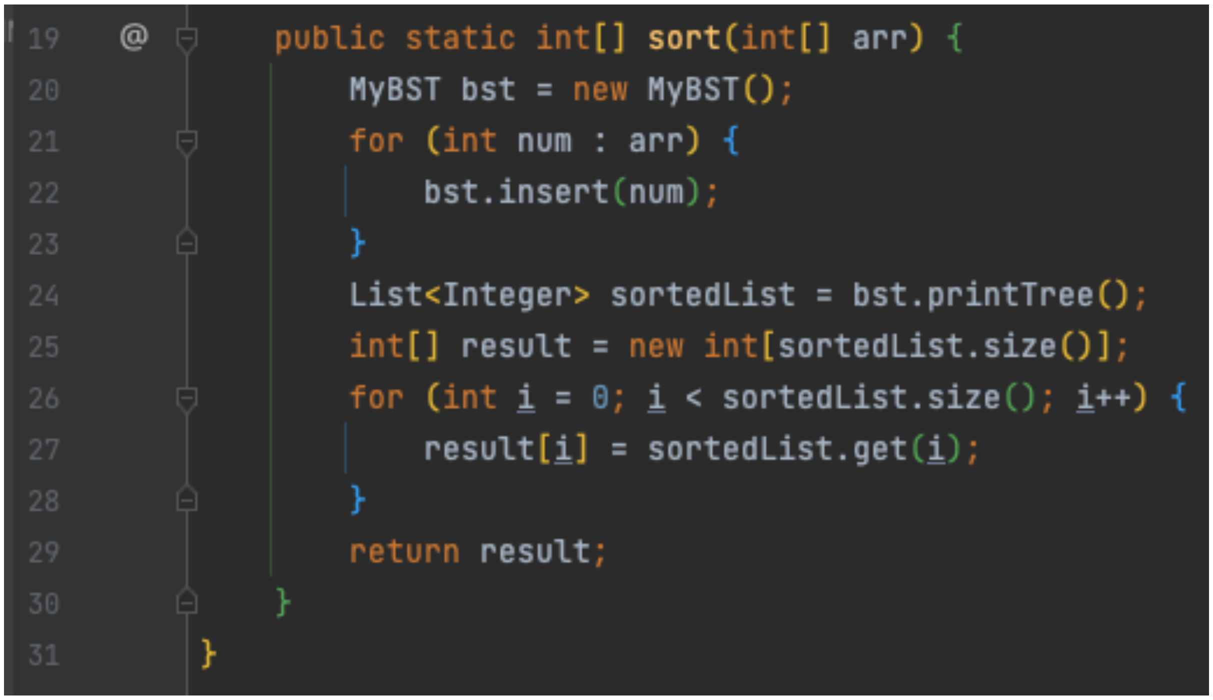
```java
    public void reverse() {
        Node prev = null;
        Node curr = head;
        Node next;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;

        }
        head = prev;
    }

```

Running time: O(n)

3. Create a sorting routine based on a BST and place it in the sorting environment, distributed earlier. For this exercise, your new class BSTSort should be a subclass of Sorter. Your BSTSort class can be essentially the same as the BST class given in the slides (see the folder in your labs directory for this lab), except that you will need to modify the printTree method so that it outputs values to an array (rather than printing to console). After you have implemented, discuss the asymptotic running time of your new sorting algorithm. Run an empirical test in the sorting environment and explain where BSTSort fits in with the other sorting routines (which algorithms is it faster than? which is it slower than?).



```java
    public static int[] sort(int[] arr) {
        MyBST bst = new MyBST();
        for (int num : arr) {
            bst.insert(num);
        }
        List<Integer> sortedList = bst.printTree();
        int[] result = new int[sortedList.size()];
        for (int i = 0; i < sortedList.size(); i++) {
            result[i] = sortedList.get(i);
        }
        return result;
    }
}
```

- creating BST takes O(nlogn), printTree() takes O(n) -> running time: O(nlogn)

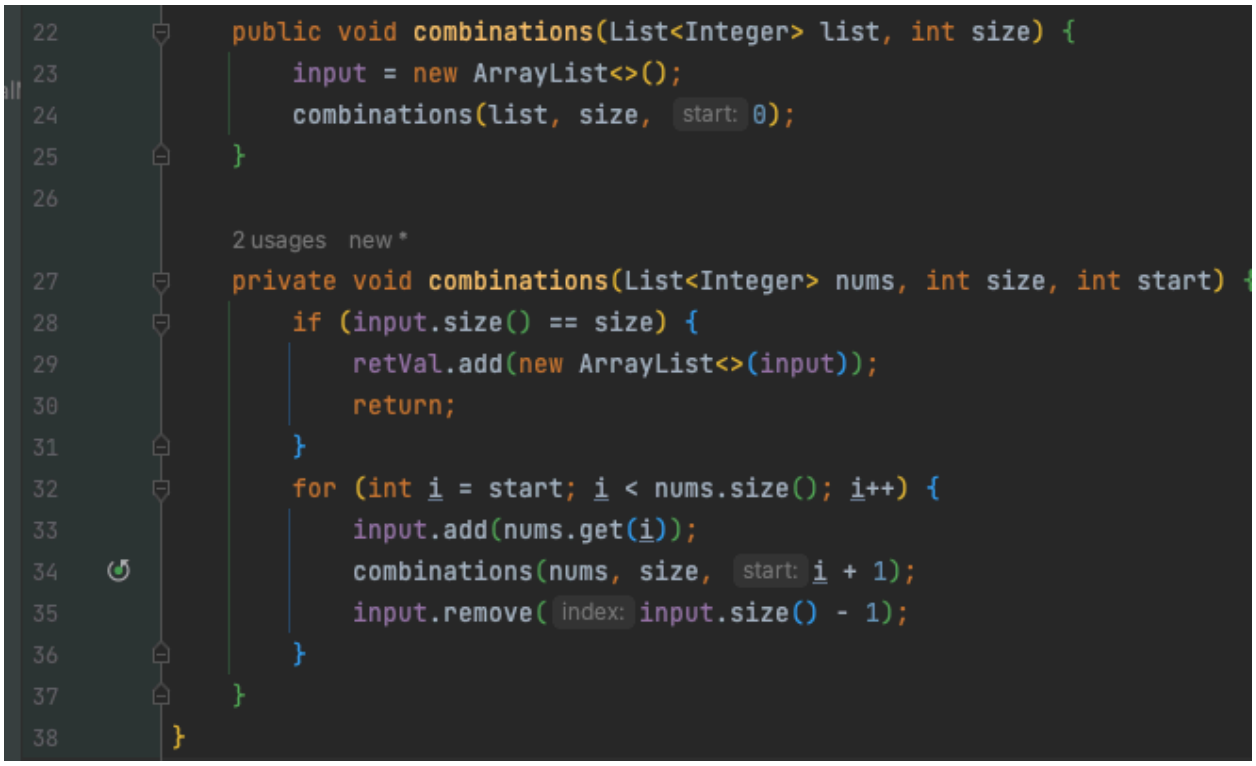
4. For each integer n = 1, 2, 3,…, 7, determine whether there exists a red-black tree having exactly n nodes, with all of them black. Fill out the chart below to tabulate the results:

Num nodes n	Does there exist a red-black tree with n nodes, all of which are black?
1	Yes
2	No
3	Yes
4	No
5	No
6	No
7	Yes

5. For each integer n = 1,2,3,…, 7, determine whether there exists a red-black tree having exactly n nodes, where exactly one of the nodes is red. Fill out the chart below to tabulate the results:

Num nodes n	Does there exist a red-black tree with n nodes, where exactly one of the nodes is red?
1	No
2	Yes
3	No
4	Yes
5	Yes
6	No
7	No

6. Write in Java a recursion with backtracking solution to the following problem: Given a set X having size at least 3 in which all elements are distinct, output a list of all subsets of size three. Example: If your input set is [3, 5, 2, 4], output should be: [[3,5,2], [3,5,4], [3,2,4], [5,2,4]]



```
    public void combinations(List<Integer> list, int size) {
        input = new ArrayList<>();
        combinations(list, size, start: 0);
    }

    2 usages   new *
    private void combinations(List<Integer> nums, int size, int start) {
        if (input.size() == size) {
            retVal.add(new ArrayList<>(input));
            return;
        }
        for (int i = start; i < nums.size(); i++) {
            input.add(nums.get(i));
            combinations(nums, size, start: i + 1);
            input.remove( index: input.size() - 1);
        }
    }
}
```