Lab assignment solution

Problem 1:

```
Algorithm reverseWords(s):
    Input: a string s consisting of n words separated by a space
    Output: string sr, a version of s where each word is reversed.
    sr ← ""
    for(i ← 0 to s.length())
        if(i=s.length || s[i] = '')
            while(!charStack.isEmpty())
                c ← charStack.pop()
                sr.append(c)
            sr.append(' ')
        else
            c ← s[i]
            charStack.push(c)
    return sr
```

To show that the algorithm is $O(n)$, assuming a word can have a maximum of m characters

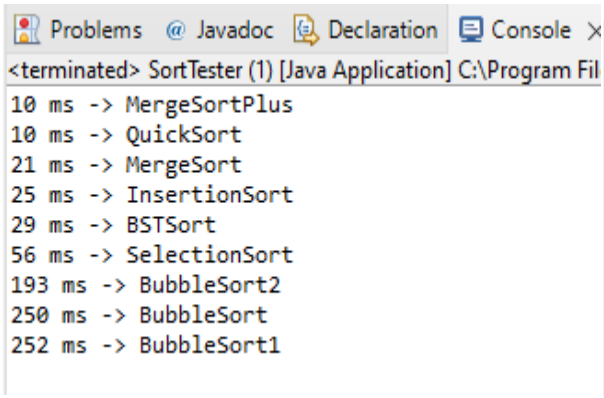
- The for loop runs n times. So $O(n)$
- The while loop has a worst-case run time of $O(m * constant)$, as append and pop take $O(1)$

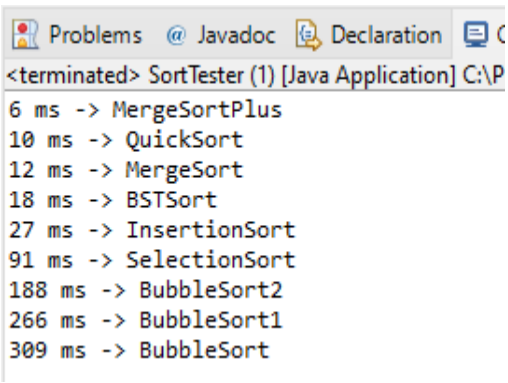
  Accessing from array and pushing to a stack are also a constant time operation.

  Therefore, the runtime is $O(m * n)$, and if m is a fixed constant or negligible compared to n, the runtime becomes $O(n)$

Problem 2:

The average running time of Binary search tree sort seems to be less than Insertion sort but larger than Selection sort. But not always. There are few cases where the tree sort is faster than Insertion sort.





```java
package sortroutines;

import java.util.Arrays;

import runtime.*;

/**
 * This is a BST that handles Integer data. The insert method works only if
 * there are no duplicates.
 */
public class BSTSort extends Sorter {
    int[] arr;
    int[] sorted;
    int i;
    /** The tree root. */
    private Node root;

    // start with an empty tree
    public BSTSort() {
        root = null;
    }

    public void sortedInt() {
        traverseTree(root);
    }
    private void traverseTree(Node t) {
        if (t != null) {
            traverseTree(t.left);
            sorted[i++] = (t.element);
            traverseTree(t.right);
        }
    }

    // /////insertion methods
    public void insert(Integer x) {
        if (root == null) {
            root = new Node(x, null, null);
        } else {
            Node n = root;
            boolean inserted = false;
            while (!inserted) {
                if (x.compareTo(n.element) < 0) {
                    // space found on the left
```

```
                if (n.left == null) {
                    n.left = new Node(x, null, null);
                    inserted = true;
                } else {
                    n = n.left;
                }
            }

            else if (x.compareTo(n.element) > 0) {
                // space found on the right
                if (n.right == null) {
                    n.right = new Node(x, null, null);
                    inserted = true;
                } else {
                    n = n.right;
                }
            } else {
                inserted = true;
            }

        }

    }
}

public static void main(String[] args) {
    BSTSort bst = new BSTSort();
    int[] array = new int[]{2,15,71,95,97,3,75,34,23};
    int[] sortedArr = bst.sort(array);
    System.out.println(Arrays.toString(sortedArr));
}

private void populate() {
    for(int a: arr)
        this.insert(a);
}

// //////// Node class

private class Node {

    /////////////// Constructors

    @SuppressWarnings("unused")
    Node(Integer theElement) {
```

```
            this(theElement, null, null);
        }

        Node(Integer element, Node left, Node right) {
            this.element = element;
            this.left = left;
            this.right = right;
        }

        private Integer element; // The data in the node
        private Node left; // Left child
        private Node right; // Right child
    }

    @Override
    public int[] sort(int[] arr) {
        this.arr = arr;
        sorted = new int[arr.length];
        i = 0;
        populate();
        sortedInt();
        return sorted;
    }
}
```

Problem 3:

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	Red Black tree with all blacks exists
1	Yes
2	No
3	Yes
4	No
5	No
6	No
7	Yes

Problem 4:

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	Red black tree with exactly one red exist
1	No
2	Yes
3	No
4	Yes
5	Yes
6	No
7	No