

T-301-REIR, REIKNIRIT/ALGORITHMS
HAUST/FALL 2019
D4 - SYMBOL TABLES (AND SORTING)

Problems 4 through 7 are of the problem solving type. You are to give an efficient algorithmic method that solves the given problem. Give answers as a clear but succinct text description (few lines each). Grade is given for serious effort.

If not specified, you should assume that input numbers can have a very large range.

Problem 1. What is the most frequent word in "A tale of two cities" (tale.txt) of length at least 7 that starts with the same letter as your first name? How frequent is it? [Hint: Consider FrequencyCounter.java]

```
package D4_SymbolTables_Sorting; import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.ST;
import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class FrequencyCounter {
    final static char myLetter = 'b';
    final static int minlen = 7;
    public static void main(String[] args) {
        ST<String, Integer> st = new ST<String, Integer>();
        In in = new In("https://introcs.cs.princeton.edu/java/data/tale.txt");
        String[] words = in.readAllStrings();

        for (String word: words) {
            String curr_word = word.toLowerCase();
            char first_letter = curr_word.charAt(0);

            if (curr_word.length() >= minlen && first_letter == myLetter) {
                Integer count = st.get(curr_word);
```

```

        if (count == null)
            { st.put(curr_word, 1); }
        else
            { st.put(curr_word, count+1); }
    }
}

String max_word = "";
Integer max_count = 0;

for (String word : st.keys()) {
    int count = st.get(word);
    if (count > max_count) {
        max_word = word;
        max_count = count;
    }
}

StdOut.println("Word with max count: " + max_word);
StdOut.println("Count: " + max_count);
}}

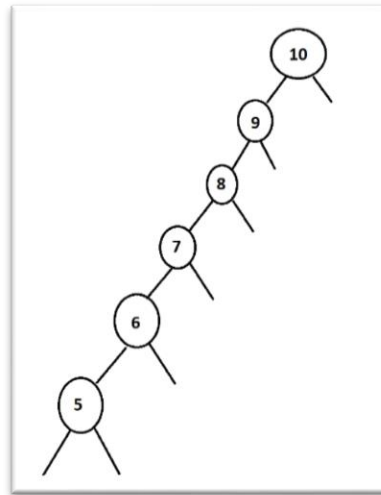
```

Word with max count: brought
Count: 78

Problem 2. (Problem 3.2.4) Suppose that a certain BST has keys that are integers between 1 and 10, and we search for 5. Which sequence below cannot be the sequence of keys examined?

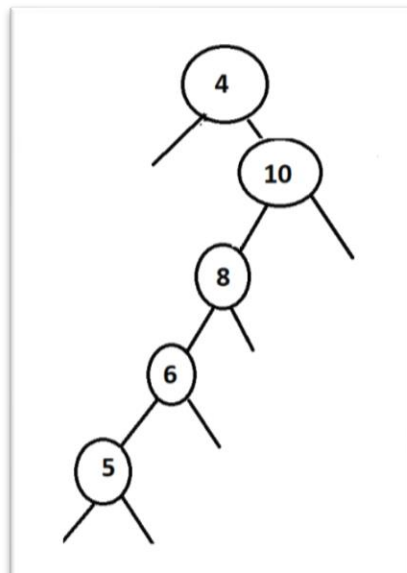
- A. 10, 9, 8, 7, 6, 5
- B. 4, 10, 8, 6, 5
- C. 1, 10, 2, 9, 3, 8, 4, 7, 6, 5
- D. 2, 7, 3, 8, 4, 5
- E. 1, 2, 10, 4, 8, 5

- Because a binary search tree (BST) compares values in nodes, moving either to the left node or the right node depending on the value being search, you can only move down the tree (not upwards). Thereby:
- a) A is a valid option, as the search goes from the root of the tree, downwards to the left until it finds the value of '5' – never heading upwards at any point (see picture below).



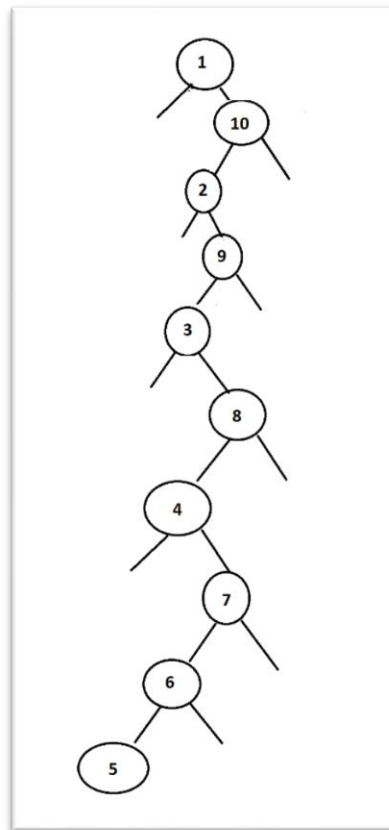
A) 10, 9, 8, 7, 6, 5

- b) B is a valid option, as the search heads down the tree until it finds the value of '5' – see picture below.



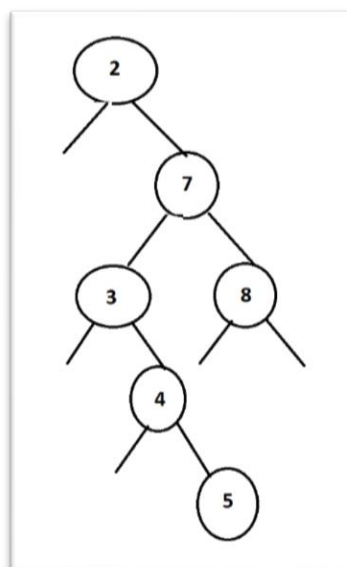
B) 4, 10, 8, 6, 5

c) C is a valid option as well – see picture below.



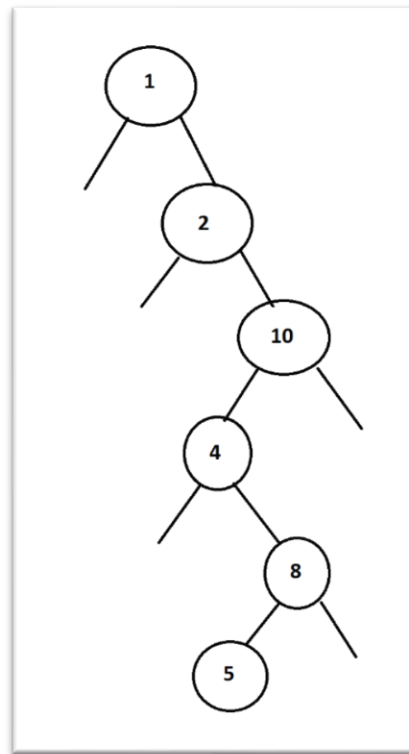
C) 1, 10, 2, 9, 3, 8, 4, 7, 6, 5

d) This sequence is not valid, as can be seen in the picture below. The reason being that we cannot move from the node 3 to the node 8, while searching for node number 5. The node 7 has a left child (3) and a right child (8), and as a child of a node cannot move to another child of the same node, the sequence is invalid.



D) 2, 7, 3, 8, 4, 5

- e) E is a valid sequence, as we once again move downwards, eventually finding the value '5' without moving upwards or sideways in our search for the node – see picture below.



E) 1, 2, 10, 4, 8, 5

Problem 3. Write the constructor `BST(double a[])`, that converts the unsorted array of numbers into a corresponding (properly ordered) binary search tree. The tree should be of minimum height, and the method efficient. You may use support functions.

```

public class BST<Key extends Comparable<Key>, Value> {
    private Node root;           // root of BST

    private class Node {
        private Key key;          // sorted by key
        private Value val;        // associated data
        private Node left, right; // left and right subtrees
        private int size;         // number of nodes in subtree

        public Node(Key key, Value val, int size) {
            this.key = key;
            this.val = val;
            this.size = size; }
    }

    /** Initializes an empty symbol table. */
    public BST()
    { }
  
```

```

    /** Inserts the specified key-value pair into the symbol table,
    overwriting the old value with the new value if the symbol table already
    contains the specified key. */
    public void put(Key key, Value val) {
        if (key == null) throw new IllegalArgumentException("calls put()
with a null key");
        root = put(root, key, val);
    }

    private Node put(Node x, Key key, Value val) {
        if (x == null) return new Node(key, val, 1);
        int cmp = key.compareTo(x.key);

        if (cmp < 0)
            x.left = put(x.left, key, val);
        else if (cmp > 0)
            x.right = put(x.right, key, val);
        else
            x.val = val;

        return x;
    }

    public static void main(String[] args) {
        StdOut.print("How many nodes in tree? ");
        int N = StdIn.readInt();

        double[] array = new double[N];

        for (int i = 0; i < N; i++)
            { array[i] = (Math.random() * 49+1); }

        BST<Double, Integer> tree = new BST<Double, Integer>();

        // Create new Binary Search Tree from array
        for (int j = 0; j < array.length; j++)
            { tree.put(array[j], j); }
    }
}

```

Problem 4. Josie needs a data structure that can handle the following operations: push, pop, contains, remove, where contains(Item x) answers whether a given item is in the data structure and remove(Item x) removes the given item from the data structure. How can this be implemented efficiently?

- I would most likely use a Binary Search Tree (BST) to implement this data structure, as then I could easily remove a node from the tree, inquire if a node is currently in the tree, as well as push and pop to/from the tree itself.

This would be an efficient implementation for both search(contains) and insert(push), as both would guarantee (on average) a time complexity of $1.39 \lg N$, and for deleting (pop/remove) a node in the tree the time complexity would be the square root of N , on average.

Problem 5. We are given k sorted arrays, with a total of N (distinct) elements, where $k < N$. How can we output all the numbers in sorted order, using time $O(N \log k)$?

- I would say that the sorting method of Mergesort would work best here, as the problem is like the sorting algorithm of Mergesort – that is, divide and conquer with dividing an array into smaller subgroups (which has already been done at the start of this particular problem), and then sorting each subgroup by combining the arrays into one single sorted array of (distinct) elements. In this scenario, and using this sorting method, the result would lead to the time complexity of $O(n \log n)$, which is then equal to $O(N \log k)$.


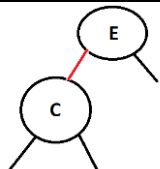
Problem 6. Given a collection of intervals (on the real line) and a real value x , a stabbing count query is the number of intervals that contain x . Design a data structure that supports interval insertions intermixed with stabbing count queries, in logarithmic time per operation.

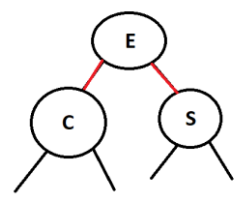
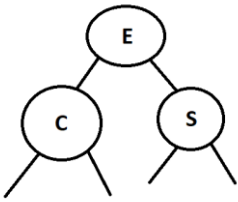
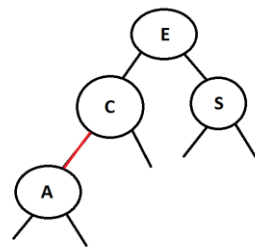
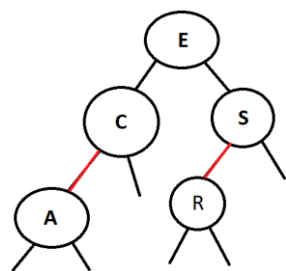
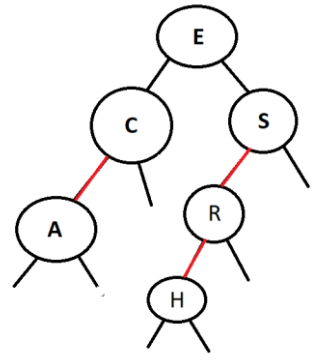
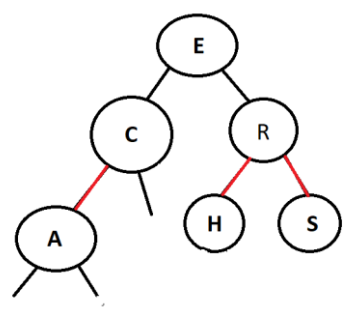
- I don't understand the question, as I am not familiar with 'stabbing count queries' and I do not completely understand what the problem is focused on (insertion and/or queries?)..... My guess would be to use a BST, as that would (probably) result in $\log N$ time complexity.

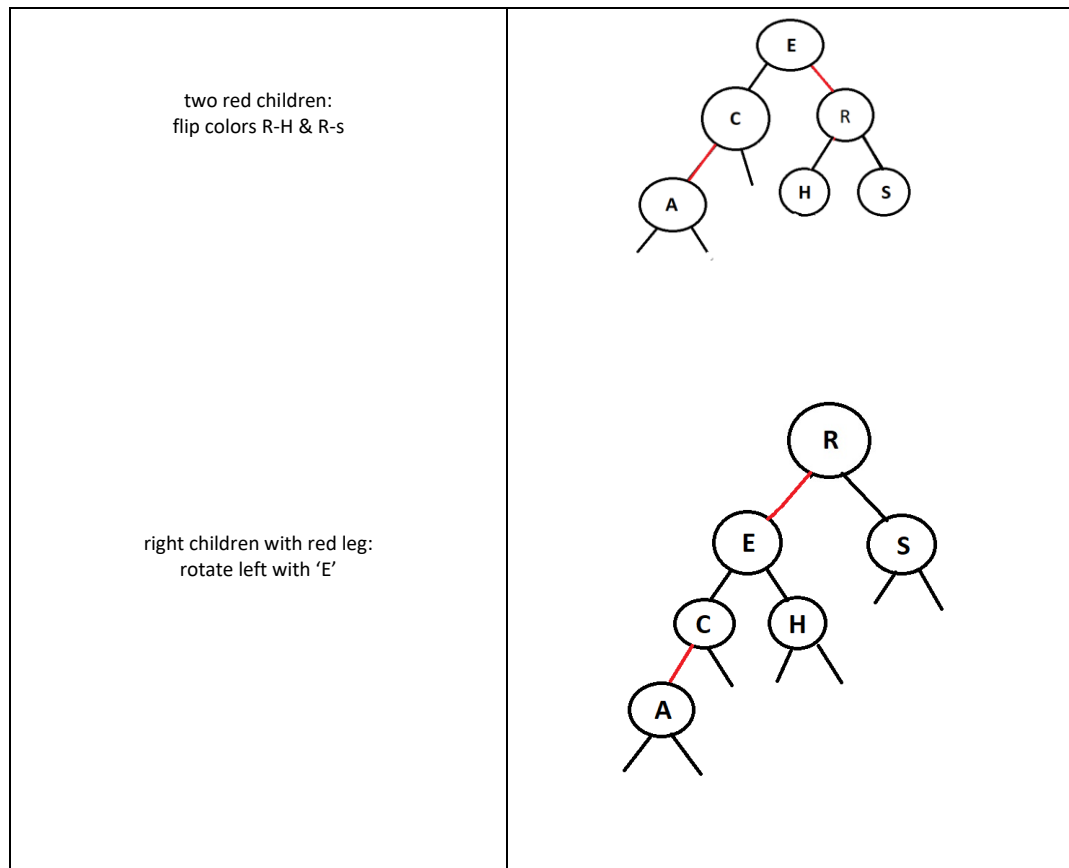
Problem 7. Give a sequence of N insertions into a red-black tree that lead to the tree having height $\sim 2\log N$. (You may assume $N = 2^k - 2$, for some k).

- Height of a red-black tree in the worst case is $2\log N$, that is when every other leg in the tree is red, duplicating the height of the tree with every red leg added – making the height of the tree double the height of the black legs in the tree.

Example:

Insert(E)	
Insert(C) red leg added E-C	

<p>Insert(S) red leg added E-S</p> <p>two red children: color flip E-C & E-S</p>	 
<p>Insert(A) red leg added C-A</p>	
<p>Insert(R) red leg added S-R</p>	
<p>Insert(H) red leg added R-H</p> <p>rotate right with 'S'</p>	 



Problem 8. Suppose that the keys A through G, with the hash keys given below, are inserted in some order into an initially empty table of size 7 using linear probing ($M=7$, no resizing).

key A B C D E F G

hash 2 0 5 4 4 4 2

Which of the following (more than one might apply) could not possibly result from inserting these keys? Explain briefly.

- 1) B E A G D F C
- 2) C F A G D E B
- 3) F B G A E C D
- 4) F C B G A D E

- Linear probing is a strategy for resolving collision of keys that map to the same index in a hash table. If two keys have the same hash value, the next available spot is used at a higher index. If the end of the array is reached, you go back to the front of the array and search for an available spot continuously until found. With that in mind, with the given examples above the result of the table would be as follows:

1)

0	1	2	3	4	5	6
B	C	A	G	E	D	F

2)

0	1	2	3	4	5	6
E	B	A	G	F	C	D

3)

0	1	2	3	4	5	6
B	D	G	A	F	E	C

4)

0	1	2	3	4	5	6
B	E	G	A	F	C	D

- Thereby resulting in that none of the above examples are impossible. Put in another way, all of the key insertion examples are possible.