CS 61BL Summer 2026

BSTs, Midterm Review

Reciation 02: July 13th, 2026

1 BST Asymptotics

Below we define the **find** method of a BST (Binary Search Tree) as in lecture, which returns the BST rooted at the node with key **sk** in our overall BST. In this setup, assume a **BST** has a **key** (the value of the tree root) and then pointers to two other child BSTs, **left** and **right**.

```
public static BST find(BST tree, Key sk) {
    if (tree == null) {
        return null;
    }
    if (sk.compareTo(tree.key) == 0) {
        return tree;
    } else if (sk.compareTo(tree.key) < 0) {
        return find(tree.left, sk);
    } else {
        return find(tree.right, sk);
    }
}
```

(a) Assume our BST is perfectly bushy. What is the runtime of a single **find** operation in terms of N, the number of nodes in the tree? Can we generalize the runtime of **find** to a theta bound?

(b) Say we have an empty BST and want to insert the keys **[6, 2, 5, 9, 0, -3]** (in some order). In what order should we insert the keys into the BST such that the runtime of a single **find** operation after all keys are inserted is $O(N)$? Draw out the resulting BST.

2 Is This a BST?

In this setup, assume a `BST` (Binary Search Tree) has a `key` (the value of the tree root represented as an `int`) and pointers to two other child BSTs, `left` and `right`. Additionally, assume that `key` is between `Integer.MIN_VALUE` and `Integer.MAX_VALUE` non-inclusive.

(a) The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which `brokenIsBST` fails.

```
public static boolean brokenIsBST(BST tree) {
    if (tree == null) {
        return true;
    } else if (tree.left != null && tree.left.key >= tree.key) {
        return false;
    } else if (tree.right != null && tree.right.key <= tree.key) {
        return false;
    } else {
        return brokenIsBST(tree.left) && brokenIsBST(tree.right);
    }
}
```

(b) Now, write `isBST` that fixes the error encountered in part (a).

Hint: You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

Hint 2: You want to somehow store information about the keys from previous layers, not just the direct parent and children. How do you use the parameters given to do this?

```
public static boolean isBST(BST T) {
    return isBSTHelper(_____________________________________________________________);
}

public static boolean isBSTHelper(BST T, int min, int max) {

    if (_____________________________________________________________) {
        _____________________________________________________________;
    } else if (_____________________________________________________________) {
        _____________________________________________________________;
    } else {
        _____________________________________________________________;
    }
}
```

3 Transpose

The transpose of a 2D array is a new 2D array whose rows are the columns of the original (and vice versa). In an upper-triangular 2D array, you are guaranteed that every row has exactly **one** less element than the row above it. The last row is guaranteed to have one element. For example, let `A` be the 2D array below on the left. The transpose of `A` is the resulting 2D array below on the right.

Extra: What if we were only guaranteed that the length of each row is less than or equal to the size of the row above it (not always 1 smaller)? What would you change about the algorithm?

$$A = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 \\ \cline{1-3} 8 & 9 \\ \cline{1-2} 10 \\ \cline{1-1}\end{array} \implies \begin{array}{|c|c|c|c|}\hline 1 & 5 & 8 & 10 \\ \hline 2 & 6 & 9 \\ \cline{1-3} 3 & 7 \\ \cline{1-2} 4 \\ \cline{1-1}\end{array} = transposeTriangular(A)$$

Given an upper-triangular 2D array `A`, non-destructively transpose `A`.

```
// Non-destructively transposes A. Assume that A is upper-triangular.
public static int[][] transposeTriangular(int[][] A) {

    if (_____) {
        return _____;
    }

    _____;

    for (_____) {
        _____;
        _____;

        for (_____) {
            _____;
        }
    }

    return _____;
}
```

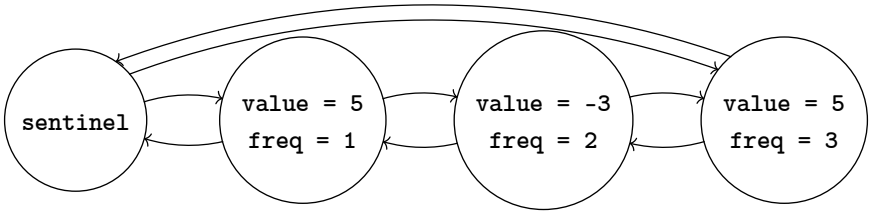
4 Compressed Doubly-Linked List

from Spring 2025 Midterm 1

Creating new nodes for consecutive identical elements in a doubly-linked list is inefficient! To combat this, 61B Inc. has created a new data structure called the Compressed Doubly-Linked List (`CDLL`).

Each node in a `CDLL` has a next and prev pointer, along with parameters value and freq, and represents a sequence of freq consecutive elements equal to value.

For example, a `CDLL` representing the list `[5, -3, -3, 5, 5, 5]` would look like:



Except for the sentinel, a `CDLL` may not have two consecutive nodes with the same value, and may not have nodes with freq less than 1. The class is defined as follows:

```
public class CDLL implements Iterable<Integer> {
  private Node sentinel;
  private class Node {
    Node prev;
    Node next;
    int value;
    int freq;
    public Node (Node prev, Node next, int value, int freq) {
      this.prev = prev;
      this.next = next;
      this.value = value;
      this.freq = freq;
    }

    public CDLL(List<Integer> original) {
    // part a
    }

    @Override
    public Iterator<Integer> iterator() {
      return new CDLLIterator();
    }
    private class CDLLIterator implements Iterator<Integer> {
    // part b
    }
  }
```

(a) Fill out the constructor of the CDLL class, which takes in a List as an argument and converts it into its corresponding CDLL representation. You may not need all lines.

```
public CDLL(List<Integer> original) {

  sentinel = new Node(null, null, -1, -1);
  Node curr = sentinel;

  for (_____:_____) {
    if (_____&&_____) {
      _____;
    } else {
      _____;
      _____;
      _____;
    }
  }
  _____;
  _____;
}
```

(b) Complete the CDLLIterator class, whose iteration returns the values of the CDLL in the order they would appear in a regular List. Your iterator may not modify the CDLL. You may not need all lines.

```
public CDLLIterator implements Iterator<Integer> {
  Node curr = _____;
  int count = 0;

  @Override
  public boolean hasNext() {
    return _____;
  }

  @Override
  public Integer next() {
    _____;
    _____;
    if (_____) {
      _____;
      _____;
    }
    return _____;
  }
}
```

5 Flip Flop

Suppose we have the flip function as defined below. Assume the method unknown returns a random integer between 1 and N, exclusive, and runs in constant time. For each definition of the flop method below, give the best and worst case runtime of flip in big Theta notation as a function of N.

```
public static void flip(int N) {
  if (N <= 100) {
    return;
  }
  int stop = unknown(N);
  for (int i = 1; i < N; i++) {
    if (i == stop) {
      flop(i, N);
      return;
    }
  }
}
```

(a)
```
public static void flop(int i, int N) {
  flip(N - i);
}
```

Best case: _____	Worst case: _____

(b)
```
public static void flop(int i, int N) {
  int minimum = Math.min(i, N - i);
  flip(minimum);
  flip(minimum);
}
```

Best case: _____	Worst case: _____

(c)
```
public static void flop(int i, int N) {
  flip(i);
  flip(N - i);
}
```

Best case: _____	Worst case: _____