Trees & Lists

class TreeNode {
    public int myValue;
    public TreeNode myLeft;
    public TreeNode myRight;
    public TreeNode(int value, TreeNode left, TreeNode right) {
        myValue = value;
        myLeft = left;
        myRight = right;
    }
}

class ListNode {
    public int myValue;
    public ListNode myNext;
    public ListNode(int value, ListNode next) {
        myValue = value;
        myNext = next;
    }
}

Recurrence Relations
In the relations below, $T(0) = T(1) = 1$.

\[
\begin{align*}
A & : T(n) = T(n/2) + O(1) & \Theta(\log n) \\
B & : T(n) = T(n/2) + O(n) & \Theta(n) \\
C & : T(n) = 2T(n/2) + O(1) & \Theta(n) \\
D & : T(n) = 2T(n/2) + O(n) & \Theta(n \log n) \\
E & : T(n) = T(n-1) + O(1) & \Theta(n) \\
F & : T(n) = T(n-1) + O(n) & \Theta(n^2) \\
G & : T(n) = 2T(n-1) + O(1) & \Theta(2^n)
\end{align*}
\]

String

- `.length()` Get the length of the String. $O(1)$.
- `.charAt(i)` Get the char at index i. $O(1)$.
- `.substring(i, j)` Get the substring between indices i and j. Index i is inclusive, and index j is exclusive. $O(1)$. For example:

```java
String x = "abcdefg";
String y = x.substring(2, 4);
// y now has the value "cd"
```
ArrayList<T>  // Where T is a type, like String or Integer
- .add(i, X) Add element X to the list at index i. If no i is provided, add an element to the end of the list. Adding to the end runs in \(O(1)\).
- .get(i) Get the element at position i. Runs in \(O(1)\).
- .set(i, X) Set the element at position i to the value X. \(O(1)\).
- .size() Get the number of elements. \(O(1)\).

HashSet<T>  // Where T is a type, like String or Integer
- .size() Compute the size. \(O(1)\).
- .add(X) Add the value X to the set. If it’s already in the set, do nothing. \(O(1)\).
- .contains(X) Return a boolean indicating if X is in the set. \(O(1)\).
- .remove(X) Remove X from the set. If X was not in the set, do nothing. \(O(1)\).

HashMap<K, V>  // Where K and V are the key and value types, respectively.
- .size() Compute the size. \(O(1)\).
- .containsKey(X) Determines if the map contains a value for the key X. To get that value, use .get(). \(O(1)\).
- .get(X) Gets the value for the key X. If X is not in the map, return null. \(O(1)\).
- .put(k, v) Map the key k to the value v. If there was already a value for k, replace it. \(O(1)\).
- .keySet() Return a Set containing the keys in the map. Useful for iterating over. \(O(1)\).

To iterate over a HashSet<T>, use

```java
for (T v : nameOfSet) {
    // v is the current element of the set.
}
```

This can be combined with HashMap’s .keySet() to iterate over a HashMap.

PriorityQueue<T>  // Where T is a type, like String or Integer
- .add() Adds the specified element to this queue.
- .remove() Retrieves and removes the head of this queue.
- .size() Returns the number of elements in this collection.
- .remove(X) Removes a single instance of X from the queue, if it is present.