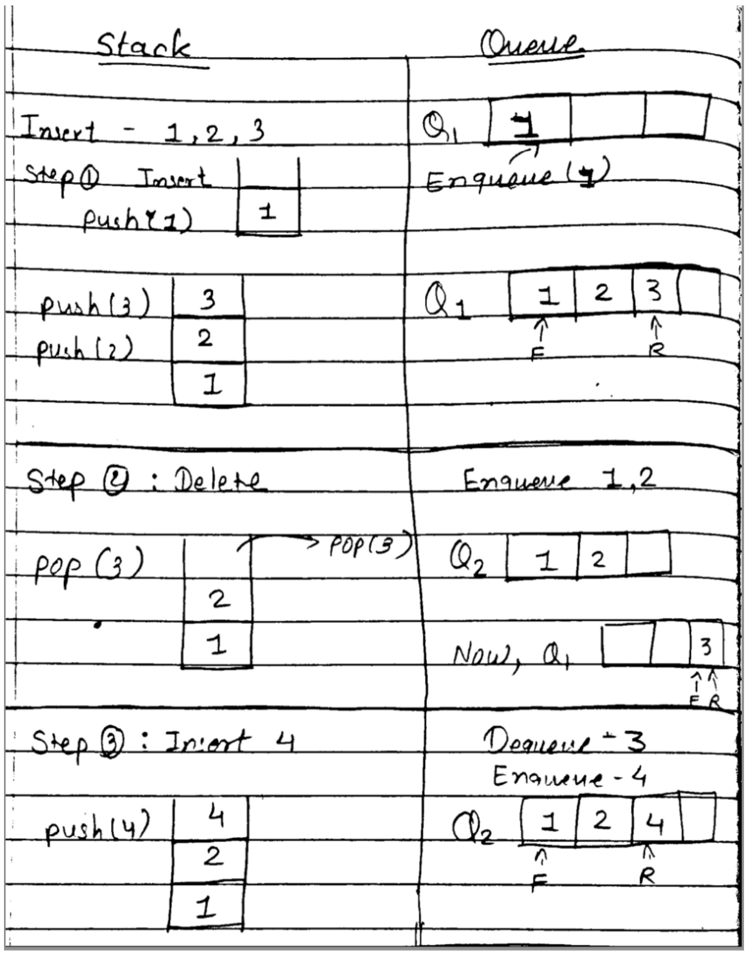
CS 600 HW 2
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

- **2.5.13**

Describe how to implement a stack using two queues. What is the running time of the `push()` and `pop()` methods in this case?

Utilizing this method, we may implement a stack using two queues.
Take two queues $Q1$ and $Q2$ for $push()$ operation, check that Q1 is empty before beginning to enqueue elements (e.g., 1, 2, 3).
It's simple to insert an element. Now when we $pop()$ the element, the example indicates that element 3 will be pop first. For this, we'll use Q2 and begin enqueuing elements from Q1. We'll keep doing this until there is just one element in Q1 left.
We dequeue it now as there are just 3 items left in Q1. To enqueue element 4 in the Q2, we must first insert element 4. Similarly, we can perform $push()$ and $pop()$ operations for stack.

All enqueue operations need $O(1)$ time, as do all dequeue operations. Therefore, the complexity of the $push()$ and $pop()$ operations will be $O(1)$.



- **2.5.20**

  Give an $O(n)$-time algorithm for computing the depth of all the nodes of a tree T, where n is the number of nodes of T.

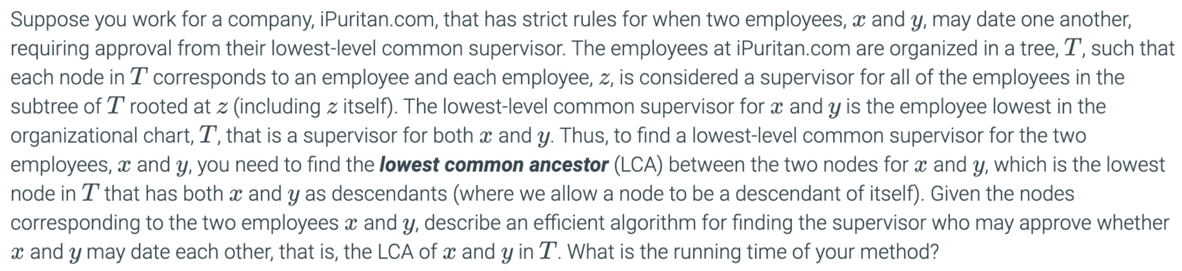
We can determine the depth of each node in a tree T using the algorithm depth described in the textbook. The depth of the node is zero if it is the root node. Following that, we can use the theory that the depth of a child is equal to the 1 plus depth of its parent to determine the depth of each node.

```
$\boldsymbol{Algorithm\ depthAllNodes(T, v)}$:

 $\boldsymbol{if}$ $T.isRoot(v)$ $\boldsymbol{then}$
    $\boldsymbol{return}$ 0
 $\boldsymbol{else}$
     $d = 0$
     $\boldsymbol{for}$ $each\ child\ w\ of\ v$ $\boldsymbol{do}$
     $d = 1 + depth(T, T.parent(v))$
     $\boldsymbol{return}$ $d$
```

The Time Complexity of this algorithm is O(n). As, there are n nodes in a tree, the for loop will run n times for n nodes, with the if-else statement running in constant time.

- **2.5.32**



Suppose you work for a company, iPuritan.com, that has strict rules for when two employees, x and y, may date one another, requiring approval from their lowest-level common supervisor. The employees at iPuritan.com are organized in a tree, T, such that each node in T corresponds to an employee and each employee, z, is considered a supervisor for all of the employees in the subtree of T rooted at z (including z itself). The lowest-level common supervisor for x and y is the employee lowest in the organizational chart, T, that is a supervisor for both x and y. Thus, to find a lowest-level common supervisor for the two employees, x and y, you need to find the ***lowest common ancestor*** (LCA) between the two nodes for x and y, which is the lowest node in T that has both x and y as descendants (where we allow a node to be a descendant of itself). Given the nodes corresponding to the two employees x and y, describe an efficient algorithm for finding the supervisor who may approve whether x and y may date each other, that is, the LCA of x and y in T. What is the running time of your method?

Consider the root node and begin traversing it to get the lowest common ancestor between two nodes x, y of a tree T. Root is the lowest common ancestor of all nodes whose specified values for x and y match.

Call the lowest common ancestor algorithm recursively for the left subtree and right subtree when x and y don't match. If x, y present as the left child and right child then the parent node is lca, if x, y present in the left subtree then lca is from left subtree vice versa with right subtree.

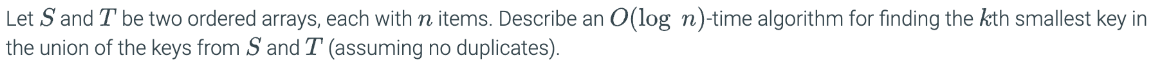
```
$\boldsymbol{Algorithm\ lca(T, x, y)}$
    $\textbf{if}\ (isRoot\ null)$
        $\textbf{return}\ null$
    $\textbf{if}\ (isRoot\ =\ x\ or\ isRoot\ =y)$
        $\textbf{return}\ root$

    $left \leftarrow lca(root.leftChild, x, y)$
    $right \leftarrow lca(root.rightChild, x, y)$

    $\textbf{if}(left\ and\ right)$
        $\textbf{return}\ root$
    $\textbf{else if}(left)$
        $\textbf{return}\ left$
    $\textbf{else}$
        $\textbf{return}\ right$
```

Time Complexity: Since this algorithm traverses the tree from root to all nodes, the running time is O(n). Where n is the number of nodes present in the tree.

- **3.6.15**


Let S and T be two ordered arrays, each with n items. Describe an $O(\log\ n)$-time algorithm for finding the kth smallest key in the union of the keys from S and T (assuming no duplicates).

To find kth smallest key in the union of keys from S and T where S and T are ordered arrays, each with n items.

Firstly, examine the $k/2$ element in the array list S. Now analyze largest element in the T which is less than $k/2$ by binary search. Now adding the indices of these 2 elements:

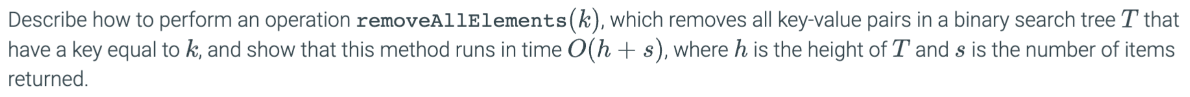
- if sum of them is equal $i.e.\ k$ then take maximum of two elements.
- If sum > k, binary search is performed to the right of S.
- If sum < k, binary search is performed to the left of S.

The same operations are now carried out on T based on the largest element in S that is less than the element being used right now.

To calculate the process' overall time complexity, the binary search operations for the two arrays S and T will require $O(log\ n)$ and $O(log\ n)$, respectively.

That would be $O(\log^2 n)$. After being resolved, the entire process will have an $O(log\ n)$ running time complexity.

- **3.6.19**


Describe how to perform an operation `removeAllElements`(k), which removes all key-value pairs in a binary search tree T that have a key equal to k, and show that this method runs in time $O(h+s)$, where h is the height of T and s is the number of items returned.

To remove all the nodes from a binary search tree. First, execute a post order traversal on the tree, calling the left and right subtrees recursively and freeing the corresponding nodes.

Algorithm $removeAllElements(T, k)$
Input: A search key k for node of a binary search tree T.
Output: Empty binary search tree

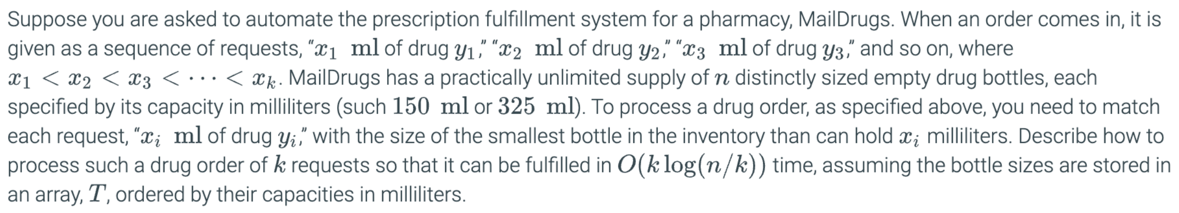
```
    **if** $T\ (k, T.root())$ $is\ null$
      **return** $null$
    **else**
      $removeAllElements(\ binaryPostorder(T, T.leftChild(k)))$
      $removeAllElements(binaryPostorder(T, T.rightChild(k)))$

      $perform\ the\ “free”\ action\ for\ key\ (k)\ node$

      $//\ (binaryPostorder\ algorithm\ is\ defined\ in\ our\ textbook\ in$ 2.4.12)
```

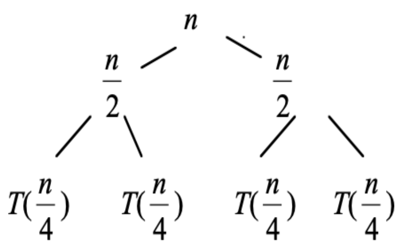
Time Complexity:
The height $h\ of\ T$ and the number of nodes visited in a tree are proportional. The remove method in a binary search tree runs in $O(h)$ time. The procedure will require $O\ (h\ +\ s)$ time in order to remove s elements from the binary search tree.

- **3.6.26**



Suppose you are asked to automate the prescription fulfillment system for a pharmacy, MailDrugs. When an order comes in, it is given as a sequence of requests, "x_1 ml of drug y_1," "x_2 ml of drug y_2," "x_3 ml of drug y_3," and so on, where $x_1 < x_2 < x_3 < \cdots < x_k$. MailDrugs has a practically unlimited supply of n distinctly sized empty drug bottles, each specified by its capacity in milliliters (such 150 ml or 325 ml). To process a drug order, as specified above, you need to match each request, "x_i ml of drug y_i," with the size of the smallest bottle in the inventory than can hold x_i milliliters. Describe how to process such a drug order of k requests so that it can be fulfilled in $O(k \log(n/k))$ time, assuming the bottle sizes are stored in an array, T, ordered by their capacities in milliliters.

Using the divide and conquer method (Binary search) to store a medicine in the smallest bottle in the inventory, which holds xi millimeters. The smallest bottle will be on the left and the largest on the right, as sorted by capacities in an array T.



The above approach will give the recurrence relation:

$$T(n) = T(n/2) + c$$

Solving this recurrence relation using iteration method

$$T(n) = \left(c + c + T\left(\frac{n}{4}\right)\right)$$

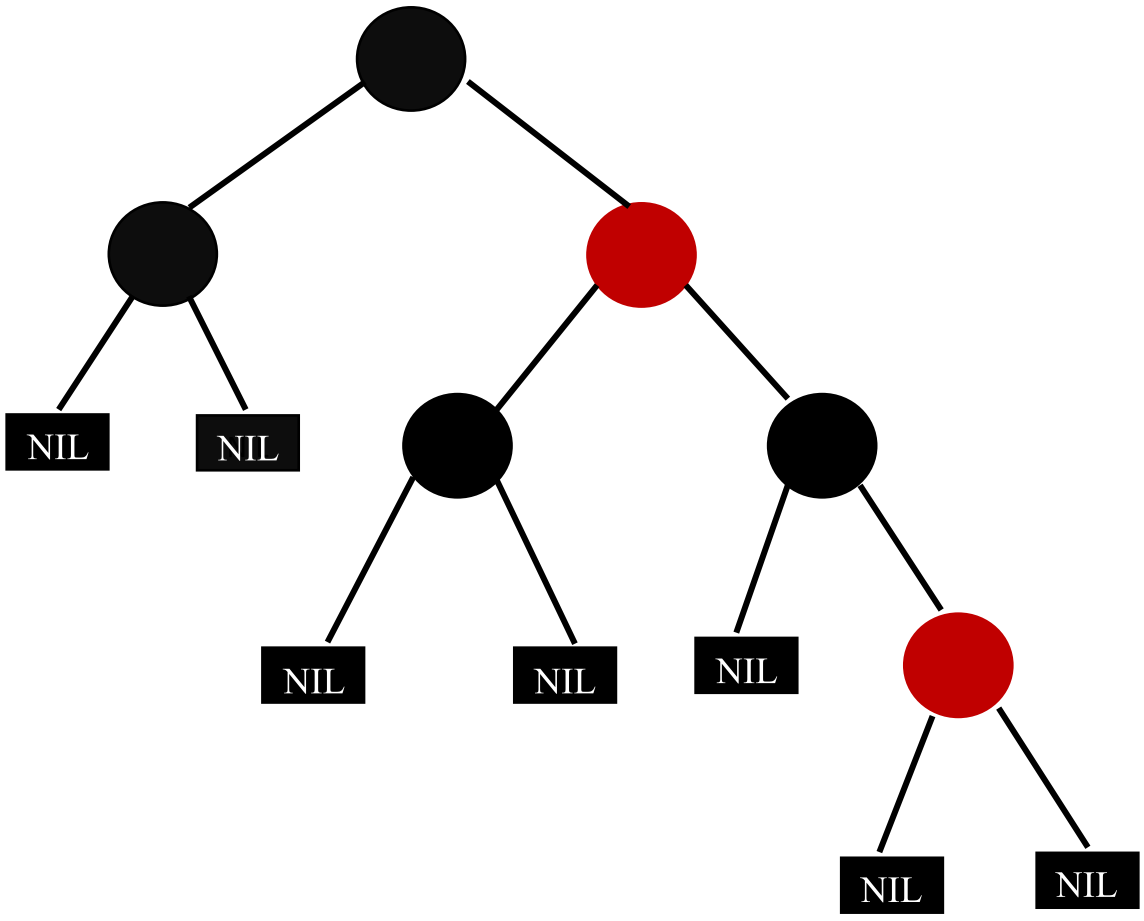
After solving will give

$$T(n) = k * c + T\left(\frac{n}{2^k}\right)$$
$$T(n) = c \log n$$

The time complexity of the above method is *c log n* for n requests, but if we process the medicine order for k requests, the time complexity will change to $O(k \log n)$. Searching for k requests will take less time because the order is already sorted for x_i. Following each x_i, the complexity changes to $O(k \log n/k)$.

- **4.7.15**

  Draw an example red-black tree that is not an AVL tree. Your tree should have at least 6 nodes, but no more than 16.



This tree is a red-black tree consists of 6 nodes and it is not an AVL tree.

As, the AVL tree's property states that, for every internal node, v, in T, the heights of the children of v may differ by at most 1

- **4.7.22**

The ***Fibonacci sequence*** is the sequence of numbers,

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots,$

defined by the base cases, $F_0 = 0$ and $F_1 = 1$, and the general-case recursive definition, $F_k = F_{k-1} + F_{k-2}$, for $k \geq 2$. Show, by induction, that, for $k \geq 3$,

$F_k \geq \phi^{k-2},$

where $\phi = (1+\sqrt{5})/2 \approx 1.618$, which is the well-known ***golden ratio*** that traces its history to the ancient Greeks.
Hint: Note that $\phi^2 = \phi + 1$; hence, $\phi^k = \phi^{k-1} + \phi^{k-2}$, for $k \geq 3$.

Given: $F_0 = 0$ and $F_1 = 1$

And $F_k = F_{k-1} + F_{k-2}$, for $k \geq 2$

Proof: $F_k \geq \varphi^{k-2}$,

Base Case: for k=2

$F_2 = F_1 + F_0 = 1$

$F_k \geq \varphi^{k-2}$ => $\varphi^0 = 1$

Therefore, true for k = 2

Base Case: for k=2

$F_3 = F_2 + F_1 = 2$

$F_k \geq \varphi^{k-2}$ => φ

$F_k > \varphi$

Therefore, true for k = 3

Now check for k-1

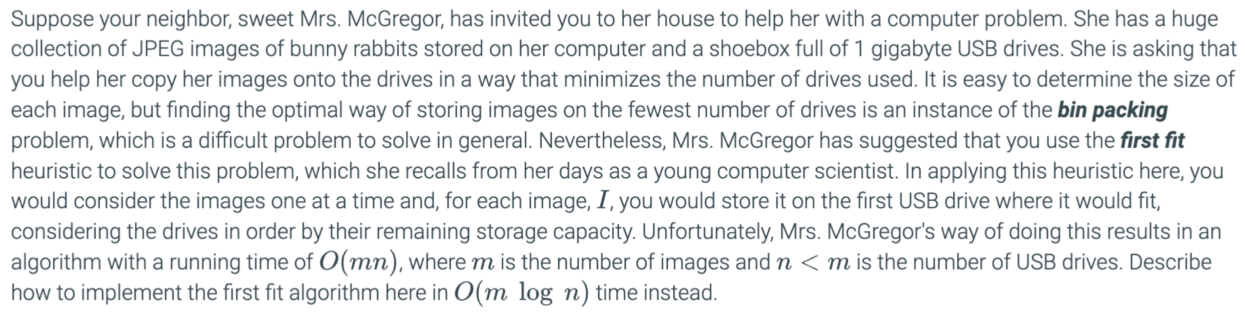
$F_{k-1} = F_{k-2} + F_{k-3} > \varphi^{k-4} + \varphi^{k-5} = \varphi^{k-4}\left(1 + \frac{1}{\varphi}\right) = \varphi^{k-4}\left(\frac{\varphi+1}{\varphi}\right) = \varphi^{k-3}$ (Given: $\varphi^2 = \varphi + 1$)

$F_{k-1} \geq \varphi^{k-3}$ (Assumption true for k-1)

So, it will be true for $F_k \geq \varphi^{k-2}$

Hence Proved

- **4.7.47**



Suppose your neighbor, sweet Mrs. McGregor, has invited you to her house to help her with a computer problem. She has a huge collection of JPEG images of bunny rabbits stored on her computer and a shoebox full of 1 gigabyte USB drives. She is asking that you help her copy her images onto the drives in a way that minimizes the number of drives used. It is easy to determine the size of each image, but finding the optimal way of storing images on the fewest number of drives is an instance of the ***bin packing*** problem, which is a difficult problem to solve in general. Nevertheless, Mrs. McGregor has suggested that you use the ***first fit*** heuristic to solve this problem, which she recalls from her days as a young computer scientist. In applying this heuristic here, you would consider the images one at a time and, for each image, I, you would store it on the first USB drive where it would fit, considering the drives in order by their remaining storage capacity. Unfortunately, Mrs. McGregor's way of doing this results in an algorithm with a running time of $O(mn)$, where m is the number of images and $n < m$ is the number of USB drives. Describe how to implement the first fit algorithm here in $O(m \log n)$ time instead.

Bin Packing Problem: A finite number of bins or containers, each of volume V, must be packed with a certain number of objects from different volumes in a way that uses the fewest possible bins.

First fit heuristic: This algorithm attempts to place the item in the first bin that can accommodate the items, and this process occurs for each of the existing items. If no bin is found, it opens a new bin and puts the item within the new bin. To encapsulate, this algorithm processes the items in arbitrary order.

We can minimize the running time complexity of storing the images into the hard drives from $O(mn)\ to\ O(m\ log\ n)$ with the help of balancing search trees (AVL tree),
where m is the number of images and $n < m$ is the number of USB drives.

AVL tree takes $O(log\ n)$ while performing insertion operation for n items. While it takes m running time for checking inserting images in an order to check all the m drives, whether any space is left in the previous bins (according to First fit) or not. So, the total running time complexity for first fit is $O(m\ log\ n)$.