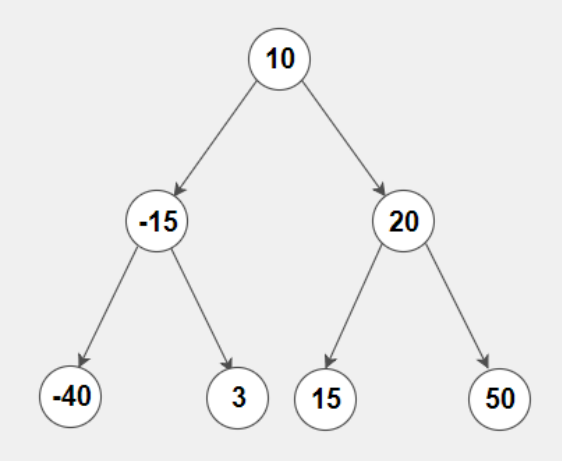
DSA - Assignment 3 (Spring 2024)

Deadline: July 01, 2024, 7:00 pm

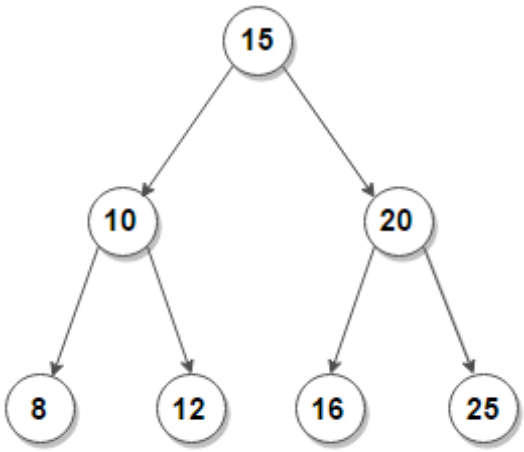
Question 1:

Given a binary search tree, find a triplet with a given sum present in it. For example, consider the following BST. If the given sum is 20, the triplet is `(-40, 10, 50)`.



Question 2:

Given a distinct sequence of keys, check if it represents a **preorder traversal** of a binary search tree (BST).
For example, the following BST can be constructed from the sequence `{15, 10, 8, 12, 20, 16, 25}` and it has the same preorder traversal:



Binary Search Tree

Question 3:

Given two binary search trees, merge them into a doubly-linked list in sorted order. For example,

```
Input: Below BSTs

      20
     /  \
   10    30
        /  \
       25  100

      50
     /  \
    5    70


Output: Below DDL

5 -> 10 -> 20 -> 25 -> 30 -> 50 -> 70 -> 100 -> nullptr
```

Question 4:

Implement the following tree:

- Red-Black Tree

Your project should have the following main menu:

```
Press 1 to insert values in the tree (one by one)
Press 2 for searching a value from the tree
Press 3 for pre-order traversal NLR
Press 4 for in-order traversal LNR
Press 5 for post-order traversal LRN
Press 6 for pre-order traversal 2 NRL
Press 7 for in-order traversal 2 RNL
Press 8 for post-order traversal 2 RLN
Press 9 for displaying parent of a node present in Tree
Press 10 to read integer values from the file "input.txt"
         to create a red-black tree
Press 11 to destroy the complete tree
Press 12 to EXIT
```

The program should exit when option 12 from the main menu is selected. There shouldn't be memory leakages or dangling pointers in your program.

Please note that in case of red-black tree, the colour of a particular node should also be displayed along with its value for options 2, 3, 4, 5, 6, 7, 8 and 9

The non-empty "input.txt" will have the data in such a way that a new value will be placed on every new line. For example, the following file (containing 7 values) is valid for creating the red-black tree (there may be less or more than 7 values):

```
10
16
2
-5
0
22
1024
```

Question 5:

Implement a function that gets a value as input from the user and deletes that value from the Red-Black tree.

Question: Improving Time Complexity for Maximum Element Retrieval (Just for you to read, no submission required. Read, discuss, think!!!)

You are given an array of (N) integers. Your task is to find the maximum element in the array using different data structures to demonstrate how time complexity can be improved step by step. The goal is to show how the choice of data structure can impact the efficiency of the solution.

Data Structures and Expected Time Complexities

1. **Naive Approach (O(N^2)):**
   - **Task:** Use a brute-force nested loop approach to find the maximum element.
   - **Explanation:** This approach involves using two nested loops where each element is compared with every other element. The outer loop runs (N) times, and for each iteration of the outer loop, the inner loop also runs (N) times, resulting in (O(N x N) = O(N^2)). This is highly inefficient and only serves as a baseline for understanding improvements.

2. **Using a Stack (O(N)):**
   - **Task:** Traverse the array once, pushing each element onto the stack and keeping track of the maximum element encountered so far.
   - **Explanation:** You only need a single pass through the array to push elements onto the stack and update the maximum value. This results in (O(N)) time complexity because each element is processed exactly once.

3. **Using a Queue (O(N)):**
   - **Task:** Traverse the array once, enqueuing each element, and keeping track of the maximum element encountered so far.
   - **Explanation:** Similar to the stack approach, this also requires a single pass through the array, resulting in (O(N)) time complexity as each element is enqueued and the maximum is updated once per element.

4. **Using a Linked List (O(N)):**
   - **Task:** Traverse the linked list once to find the maximum element.
   - **Explanation:** This involves traversing the linked list once, resulting in (O(N)) time complexity, as you simply go through each element once to find the maximum.

5. **Using a Binary Search Tree (BST) (O(N log N) for insertion, O(log N) for retrieval):**
   - **Task:** Insert all elements into a Binary Search Tree (BST) and then find the maximum element.
   - **Explanation:** Inserting an element into a balanced BST takes (O(log N)) time. Inserting (N) elements requires (O(N log N)) time because each insertion takes logarithmic time. Finding the maximum element involves traversing to the rightmost node, which takes (O(log N)) time in a balanced tree.

6. **Using a Max-Heap (O(N) for building the heap, O(log N) for removal and re-heapify):**
    - **Task:** Insert all elements into a Max-Heap and then retrieve the maximum element.
    - **Explanation:**
        - **Building the Heap:** Building a max-heap from (N) elements takes (O(N)) time using the bottom-up heapify process. This is because the cost of heapifying nodes decreases as you move up the tree, resulting in a total cost that sums to (O(N)).
        - **A question for you:** What do you think should be the Big(O) of inserting one element into the max heap?
        - **Retrieval and Re-heapify:** Retrieving the maximum element (at the root) takes (O(1)) time. However, if you remove the maximum element and need to re-heapify, it takes (O(log N)) time to restore the heap property.
        - **Combining Operations:** If you repeatedly extract the maximum element to find the Nth largest element, the process involves (O(N log N)) time. Each extraction and re-heapify operation takes (O(log N)), and performing this (N) times sums up to (O(N log N)).
7. **Optimized Approach (O(1)):**
    - **Task:** Preprocess the array to maintain the maximum element.
    - **Explanation:** By maintaining the maximum element during preprocessing, you can retrieve the maximum element in constant time, (O(1)).

Challenge: Finding the Nth Largest Element
For each of the above data structures, determine the time complexity and suggest an approach to find the Nth largest element in the array.

1. **Naive Approach:**
    - **Time Complexity:** (O(N^2))
    - **Approach:** Use nested loops to compare elements and find the Nth largest. This approach involves sorting the array in decreasing order by finding the maximum repeatedly, which takes (O(N^2)).

2. **Using a Stack:**
    - **Time Complexity:** (O(N log N))
    - **Approach:** Use a stack to sort the elements (e.g., via another sorting algorithm like quicksort) and then find the Nth largest. Sorting takes (O(N log N)) and accessing the Nth largest is (O(1)). By the way, if you use Bubble sort OR selection sort, then the complexity will change to O(N^2)

3. **Using a Queue:**
    - **Time Complexity:** (O(N log N))
    - **Approach:** Use a priority queue (min-heap of size N) to keep track of the largest elements, which requires (O(N log N)) for heap operations.

4. **Using a Linked List:**
    - **Time Complexity:** (O(N^2))

- **Approach:** Traverse the list multiple times to find and remove the largest elements until the Nth largest is found, resulting in quadratic time complexity.

5. **Using a Binary Search Tree (BST):**
   - **Time Complexity:** (O(N + N log N))
   - **Approach:** Insert all elements into the BST, then perform an in-order traversal to get elements in sorted order and find the Nth largest. In-order traversal takes (O(N)), and insertion takes (O(N log N)).

6. **Using a Max-Heap:**
   - **Time Complexity:** (O(N + N log N))
   - **Approach:** Build the max-heap in (O(N)), then perform N extract-max operations to find the Nth largest, each taking (O(log N)), totaling (O(N log N)).

7. **Optimized Approach:**
   - **Time Complexity:** (O(N log N))
   - **Approach:** Sort the array and access the Nth largest element directly. Sorting takes (O(N log N)) and accessing the element is (O(1)).

Additional Question: Finding the Maximum Element from a Min-Heap

- **Task:** Determine how to find the maximum element in a Min-Heap.
- **Challenge:** Ask the students to think about the Big(O) time complexity for finding the maximum element in a Min-Heap and suggest an approach.
- **Explanation:** In a Min-Heap, the minimum element is at the root. To find the maximum element, you need to check all leaf nodes, as they can be anywhere in the last level(s) of the heap. This would require (O(N)) time complexity because you have to potentially examine every element in the worst case.