DAY 13:

ASSIGNMENT 3:

Q) Implementing Heap Operations

Code a min-heap in C# with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

ANSWER:

```
package day13;

import java.util.ArrayList;

import java.util.NoSuchElementException;


public class MinHeap {

    private ArrayList<Integer> heap;


    public MinHeap() {

        heap = new ArrayList<>();

    }


    // Inserts a new value into the heap

    public void insert(int value) {

        heap.add(value);

        heapifyUp(heap.size() - 1);

    }


    // Removes and returns the minimum element from the heap

    public int deleteMin() {

        if (heap.isEmpty()) {

            throw new NoSuchElementException("Heap is empty");

        }

        int min = heap.get(0);

        int lastElement = heap.remove(heap.size() - 1);
    if (!heap.isEmpty()) {
        heap.set(0, lastElement);
        heapifyDown(0);
    }
    return min;
}

// Returns the minimum element from the heap without removing it
public int getMin() {
    if (heap.isEmpty()) {
        throw new NoSuchElementException("Heap is empty");
    }
    return heap.get(0);
}

// Maintains the heap property after insertion
private void heapifyUp(int index) {
    while (index > 0) {
        int parentIndex = (index - 1) / 2;
        if (heap.get(index) >= heap.get(parentIndex)) {
            break;
        }
        swap(index, parentIndex);
        index = parentIndex;
    }
}

// Maintains the heap property after deletion
private void heapifyDown(int index) {
    int size = heap.size();
    while (index < size) {
        int leftChildIndex = 2 * index + 1;

        int rightChildIndex = 2 * index + 2;

        int smallest = index;


        if (leftChildIndex < size && heap.get(leftChildIndex) < heap.get(smallest)) {

            smallest = leftChildIndex;

        }


        if (rightChildIndex < size && heap.get(rightChildIndex) < heap.get(smallest)) {

            smallest = rightChildIndex;

        }


        if (smallest == index) {

            break;

        }


        swap(index, smallest);

        index = smallest;

    }

}


// Swaps two elements in the heap

private void swap(int i, int j) {

    int temp = heap.get(i);

    heap.set(i, heap.get(j));

    heap.set(j, temp);

}


public static void main(String[] args) {

    MinHeap minHeap = new MinHeap();

    minHeap.insert(3);
```

```
        minHeap.insert(1);

        minHeap.insert(6);

        minHeap.insert(5);

        minHeap.insert(2);

        minHeap.insert(4);


        System.out.println("Minimum element: " + minHeap.getMin()); // Output: 1

        System.out.println("Deleted minimum element: " + minHeap.deleteMin()); // Output: 1

        System.out.println("Minimum element after deletion: " + minHeap.getMin()); // Output: 2

    }

}
```

EXPLANATION:

MinHeap Class:

- Uses an ArrayList to store the elements of the heap.
- Provides methods for insertion, deletion, and fetching the minimum element.

2. insert Method:

- Adds the new value to the end of the list.
- Calls heapifyUp to maintain the heap property by moving the new element up to its correct position.

3. deleteMin Method:

- Removes and returns the minimum element, which is the root of the heap (index 0).
- Replaces the root with the last element in the list and removes the last element.
- Calls heapifyDown to maintain the heap property by moving the new root down to its correct position.

4. getMin Method:

- Returns the minimum element without removing it. Throws an exception if the heap is empty.

5. heapifyUp Method:

- Moves an element up to its correct position by comparing it with its parent and swapping if necessary. Continues this until the heap property is restored or the element becomes the root.

6. heapifyDown Method:

- Moves an element down to its correct position by comparing it with its children and swapping with the smallest child if necessary. Continues this until the heap property is restored or the element becomes a leaf.

7. swap Method:

- Swaps two elements in the list.

8. Main Method:

- Demonstrates how to use the MinHeap class to insert elements, fetch the minimum, and delete the minimum element.

This implementation ensures that the heap property is maintained after each operation, with insertion and deletion operations having a time complexity of O(log n), where n is the number of elements in the heap.