**🔷 Heap কী?**

Heap হলো এক ধরনের **Complete Binary Tree**, যেখানে সবগুলো লেভেল পূর্ণ থাকে এবং শেষ লেভেল বাম থেকে ডানে পূরণ হয়।

দুই ধরনের Heap হয়ঃ

### ✅ 1. Max Heap

👉 প্রতিটি নোডের মান তার চাইল্ডদের চেয়ে **বড় বা সমান** হয়।  
👉 অর্থাৎ: **Parent ≥ Left Child, Right Child**

### ✅ 2. Min Heap

👉 প্রতিটি নোডের মান তার চাইল্ডদের চেয়ে **ছোট বা সমান** হয়।  
👉 অর্থাৎ: **Parent ≤ Left Child, Right Child**

**MAX HEAP:**

50

/ \

30 40

/ \ / \

10 5 20 25

package Heap;

import java.util.PriorityQueue;

import java.util.Collections;

public class maxHeap {

    public static void main(String[] args) {

        PriorityQueue<Integer> maxH=new PriorityQueue<>(Collections.reverseOrder());

        maxH.add(10);

        maxH.add(30);

        maxH.add(20);

        maxH.add(5);

        while (!maxH.isEmpty()) {

            System.out.println(maxH.poll()+" ");

        }

    }

}

**MIN HEAP:**

5

/ \

10 15

/ \ / \

20 30 25 50

package Heap;

import java.util.PriorityQueue;

public class MinHeap {

    public static void main(String[] args) {

        PriorityQueue<Integer> minHeap=new PriorityQueue<>();

        minHeap.add(10);

        minHeap.add(30);

        minHeap.add(20);

        minHeap.add(5);

        while (!minHeap.isEmpty()) {

            System.out.println(minHeap.poll()+" ");

        }

    }

}

MAX HEAP ADD(ARRAY)

package Heap;

import java.util.Scanner;

public class maxHeapArray {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter number of elemnts: ");

        int n=sc.nextInt();

        int[] A=new int[n+1];

        A[0]=0;

        for(int i=1;i<=n;i++){

            int temp=sc.nextInt();

            int k=i;

            while (k>1 && A[k/2]<temp) {

                A[k]=A[k/2];

                k=k/2;

            }

            A[k]=temp;

        }

        System.out.println("Max Heap");

        for(int i=1;i<=n;i++){

            System.out.println(A[i]+" ");

        }

    }

}

    // public static void main(String[] args) {

    //     int[] A = {0, 10, 20, 5, 6, 1, 8}; // index 0 unused for easier parent-child calculation

    //     int n = A.length - 1; // Because we are not using index 0

    //     // Heap construction begins

    //     for (int i = 2; i <= n; ++i) {

    //         int temp = A[i]; // Step 1: temp holds current element

    //         int k = i;

    //         // Step 2: Bubble up if parent is smaller (for max heap)

    //         while (k > 1 && A[k / 2] < temp) {

    //             A[k] = A[k / 2]; // Move parent's value down

    //             k = k / 2;       // Move up to parent's index

    //         }

    //         A[k] = temp; // Step 3: place temp in correct position

    //     }

    //     // Output the heap

    //     System.out.println("Heap Array:");

    //     for (int i = 1; i <= n; i++) {

    //         System.out.print(A[i] + " ");

    //     }

    // }

//DELETE

package Heap;

public class maxDelete {

    public static void heapifyDown(int[] heap, int n, int i) {

        int largest = i;

        int left = 2 \* i + 1; // left child

        int right = 2 \* i + 2; // right child

        if (left < n && heap[left] > heap[largest])

            largest = left;

        if (right < n && heap[right] > heap[largest])

            largest = right;

        if (largest != i) {

            int temp = heap[i];

            heap[i] = heap[largest];

            heap[largest] = temp;

            heapifyDown(heap, n, largest);

        }

    }

    public static int deleteMax(int[] heap, int n) {

        if (n <= 0) return -1;

        int max = heap[0];

        heap[0] = heap[n - 1]; // Last element goes to root

        heapifyDown(heap, n - 1, 0);

        return max;

    }

    public static void main(String[] args) {

        int[] heap = {50, 30, 40, 10, 20, 35, 25};

        int n = heap.length;

        int deleted = deleteMax(heap, n);

        System.out.println("Deleted Max: " + deleted);

        System.out.print("Updated Heap: ");

        for (int i = 0; i < n - 1; i++) {

            System.out.print(heap[i] + " ");

        }

    }

}

In Queue

package Heap;

import java.util.Collections;

import java.util.PriorityQueue;

public class maxHeapDeleteQueue {

    public static void main(String[] args) {

        // Max Heap using PriorityQueue

        PriorityQueue<Integer> maxH = new PriorityQueue<>(Collections.reverseOrder());

        // Adding elements to heap

        maxH.add(50);

        maxH.add(30);

        maxH.add(40);

        maxH.add(10);

        maxH.add(20);

        maxH.add(35);

        maxH.add(25);

        System.out.println("Max Heap: " + maxH);

        // Deleting the max (root)

        int deleted = maxH.poll(); // Removes and returns max

        System.out.println("Deleted Max: " + deleted);

        System.out.println("Heap after deletion: " + maxH);

    }

}