**Name: SHAFIN-UZ-ZAMAN**

**Reg no: SP22-BCS-063**

**Class: DSA LAB**

**Sec: C**

**Code  
Driver class:**public class Driver {

    public static void main(String[] args) {

    // Creating an object of MyArray class with size 10.

    MyArray mr=new MyArray(10);

    // Adding elements to the array.

    System.out.println("Adding");

    boolean a0=mr.addAtEnd(10);

    boolean a1=mr.addAtEnd(20);

    boolean a2=mr.addAtEnd(30);

    boolean a5=mr.addAtEnd(30);

    boolean a3=mr.addAtEnd(40);

    boolean a4=mr.addAtEnd(50);

    // Printing the array.

    mr.printArray();

    System.out.println("---------------");

    // Deleting an element from a specific index.

    System.out.println("Deleting at index 2");

    int deleted\_val=mr.deleteAtIndex(2);

    mr.printArray();

    System.out.println("---------------");

    // Adding an element at a specific index.

    System.out.println("Add at index 2");

    boolean b0=mr.addAtIndex(30,2);

    mr.printArray();

    System.out.println("---------------");

    // Deleting an element from the end of the array.

    System.out.println("Delete at end");

    int dv2=mr.deleteAtEnd();

    mr.printArray();

    System.out.println("---------------");

    // Searching for the index of a specific value in the array.

    System.out.println("Searching index of value 30 ");

    System.out.println(mr.linearSearchOfOne(30));

    // Searching for all the indexes of a specific value in the array.

    System.out.println("Searching all indexes of value 30 ");

    int arr[]=mr.linearSearchOfAll(30);

    System.out.print("At index ");

    for(int i=0;i<arr.length;i++){

        if(arr[0]!=-1){

            if(arr[0]==0){

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

            }

            if(arr[i]!=0){

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

            }

        }

    }

    System.out.println();

    System.out.println("---------------");

    // Updating an element at a specific index.

    System.out.println("Update at index 2");

    int deleted\_val2=mr.updateAtIndex(40, 2);

    mr.printArray();

    // Updating the first matching element with a specific value.

    System.out.println("Update val by finding first match");

    int deleted\_val3=mr.updateOneVal(10, 40);

    mr.printArray();

    // Updating all the matching elements with a specific value.

    System.out.println("Update all finding val");

    int deleted\_val4=mr.updateAllMatchingVal(40, 100);

    mr.printArray();

    System.out.println("---------------");

    // Sorting the array.

    System.out.println("Sorting");

    boolean a7=mr.addAtEnd(70);

    mr.sorting();

    mr.printArray();

    System.out.println("---------------");

    // Performing binary search on the array.

    System.out.println("Binary Search");

    System.out.println(mr.binarySearch(100));

}

}

**Array Class:**  
public class MyArray {

    int A[]; // array to store values

    int N; // size of the array

    int k; // current number of elements in the array

    // constructor to initialize the array with a given size

    MyArray(int N) {

        A = new int[N];

        this.N = N;

    }

    // method to add a value at the end of the array

    public boolean addAtEnd(int val) {

        if (k < N) { // check if there's enough space in the array

            A[k] = val;

            k++;

            return true;

        }

        return false;

    }

    // method to add a value at a specific index in the array

    public boolean addAtIndex(int val, int index) {

        if (k < N) { // check if there's enough space in the array

            if (index >= 0 && index < k) { // check if the index is valid

                for (int i = k; i > index; i--) {

                    A[i] = A[i - 1]; // shift elements to the right to make room for the new value

                }

                A[index] = val;

                k++;

                return true;

            }

        }

        return false;

    }

    // method to delete a value at a specific index in the array

    public int deleteAtIndex(int index) {

        int temp = 0;

        if (index >= 0 && index < k) { // check if the index is valid

            temp = A[index]; // save the value that is being deleted

            for (int i = index; i < k; i++) {

                A[i] = A[i + 1]; // shift elements to the left to fill the gap left by the deleted value

            }

            k--;

        }

        return temp;

    }

    // method to delete the value at the end of the array

    public int deleteAtEnd() {

        int temp = 0;

        if (k > 0) { // check if there are values in the array

            temp = A[k - 1]; // save the value that is being deleted

            A[k - 1] = 0; // set the last element to 0 to "delete" it

            k--;

        }

        return temp;

    }

    // method to search for the first occurrence of a value in the array

    public int linearSearchOfOne(int val) {

        int index = -1;

        for (int i = 0; i < k; i++) {

            if (A[i] == val) {

                index = i;

                break; // stop searching once the value is found

            }

        }

        return index;

    }

    // method to search for all occurrences of a value in the array

    public int[] linearSearchOfAll(int val) {

        int index[] = new int[k];

        int j = 0;

        for (int i = 0; i < k; i++) {

            if (A[i] == val) {

                index[j] = i;

                j++;

            }

        }

        if (j == 0) { // check if the value was not found in the array

            index[0] = -1;

            return index;

        }

        return index;

    }

    // method to update a value at a specific index in the array

    public int updateAtIndex(int val, int index) {

        int temp = 0;

        if (index >= 0 && index < k) { // check if the index is valid

            temp = A[index]; // save the old value

            A[index] = val; // set the new value

        }

        return temp;

    }

    // Update a single occurrence of val with updated\_val in array A

    public int updateOneVal(int val, int updated\_val) {

        int temp = 0;

        // Use linear search to find the index of val in A

        int search = linearSearchOfOne(val);

        if (search != -1) {

            // Update the value at the index with updated\_val

            A[search] = updated\_val;

        }

        // Return temp (which is always 0 in this method)

        return temp;

    }

    // Update all occurrences of val with updated\_val in array A

    public int updateAllMatchingVal(int val, int updated\_val) {

        int temp = 0;

        // Use linear search to find the indices of all occurrences of val in A

        int search[] = linearSearchOfAll(val);

        if (search[0] != -1) {

            // If there is at least one occurrence of val in A

            if (search[0] == 0) {

                // If the first occurrence of val is at index 0, set temp to val

                temp = val;

                // Update the value at index 0 with updated\_val

                A[0] = updated\_val;

            }

            // Loop through the rest of the indices with occurrences of val and update their

            // values

            for (int i = 1; i < search.length; i++) {

                if (search[i] != 0) {

                    A[search[i]] = updated\_val;

                }

            }

        }

        // Return temp (which is either 0 or val depending on the position of the first

        // occurrence of val)

        return temp;

    }

    // Sort array A in ascending order using selection sort

    public void sorting() {

        for (int i = 0; i < k; i++) {

            int smallestVal = A[i];

            // Find the index of the smallest value in the unsorted portion of the array

            for (int j = i; j < k; j++) {

                if (A[j] < smallestVal) {

                    int temp = smallestVal;

                    smallestVal = A[j];

                    A[j] = temp;

                }

            }

            // Swap the smallest value with the first value in the unsorted portion of the

            // array

            A[i] = smallestVal;

        }

    }

    // Use binary search to find the index of val in the sorted array A

    public int binarySearch(int val) {

        // Sort array A first

        sorting();

        int start = 0;

        int end = k;

        // Keep searching until start and end meet

        while (start <= end) {

            int mid = (start + end) / 2;

            if (A[mid] >= val) {

                // If the middle element is equal to val, return its index

                if (A[mid] == val) {

                    return mid;

                }

                // If the middle element is greater than val, search the left half of the array

                end = mid - 1;

            }

            // If the middle element is less than val, search the right half of the array

            if (A[mid] < val) {

                start = mid + 1;

            }

        }

        // If val is not found in A, return -1

        return -1;

    }

    // Print array A to the console

public void printArray(){

    for(int a:A){

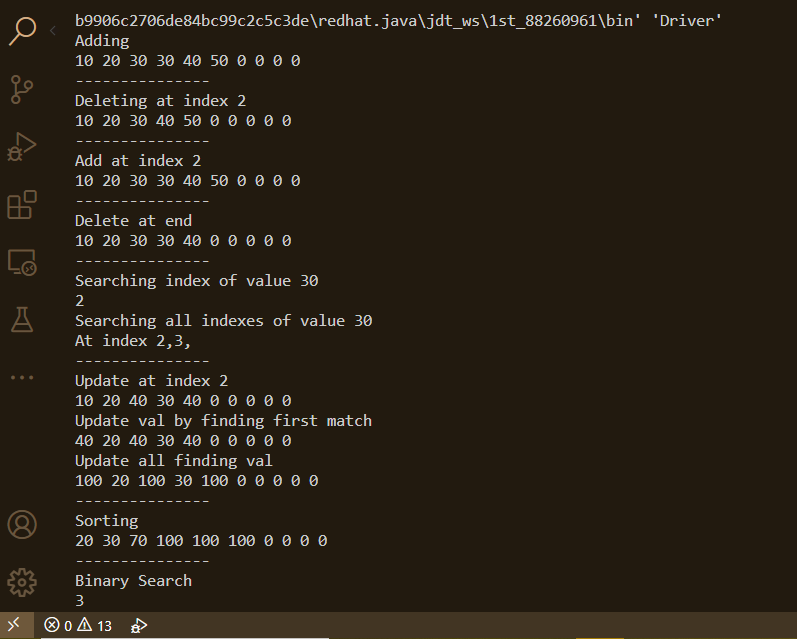
        System.out.print(a+" ");

    }

    System.out.println("");

}

}

**Output:**