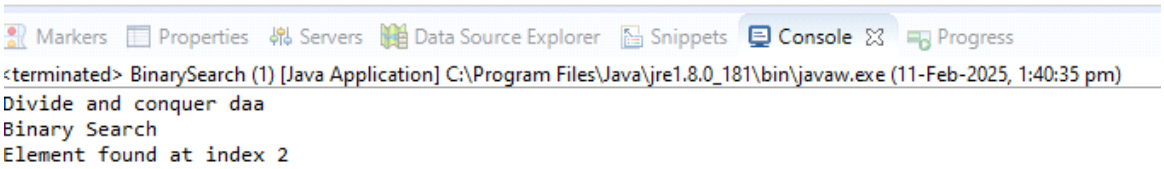
Unit -2 Divide and conquer
Lab Notes

Program No. 6- Given an array of integers, find an element from it using Binary Search.

```
package unit2;

class BinarySearch {
   int binarySearch(int a[], int l, int r, int x)
   {
      while (l <= r) {
         int m = (l + r) / 2;

         // Index of Element Returned
         if (a[m] == x) {
            return m;

         // If element is smaller than mid, then
         // it can only be present in left subarray
         // so we decrease our r pointer to mid - 1
         } else if (a[m] > x) {
            r = m - 1;

         // Else the element can only be present
         // in right subarray
         // so we increase our l pointer to mid + 1
         } else {
           l = m + 1;
         }
      }

      // No Element Found
      return -1;
   }

   public static void main(String args[])
   {
      BinarySearch ob = new BinarySearch();

      int a[] = { 2, 3, 4, 12, 44 };
      int n = a.length;
      int x = 4;
      System.out.println("Divide and conquer daa");
      System.out.println("Binary Search");
      int res = ob.binarySearch(a, 0, n - 1, x);

      if (res == -1)
         System.out.println("Element not present");
      else
```

```
        System.out.println("Element found at index " + res);
    }
}
```

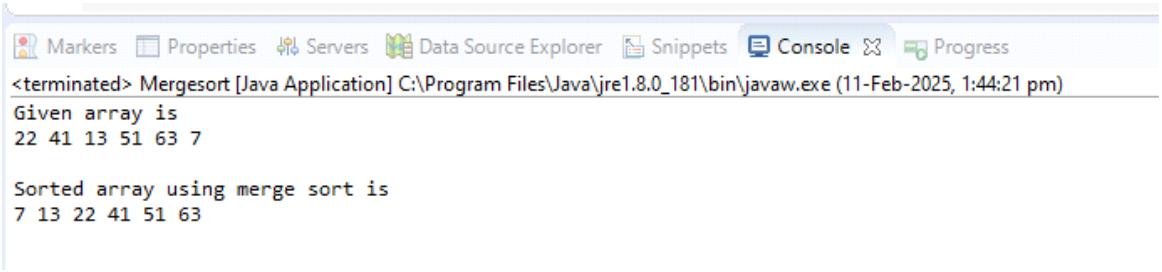


Program No. 7- Given an array of integers, sort it using the merge sort technique using the Divide and Conquer Approach.

```
package unit2;
//Java program for Merge Sort
import java.io.*;
public class Mergesort {



    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    static void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        // Create temp arrays
        int L[] = new int[n1];
        int R[] = new int[n2];

        // Copy data to temp arrays
        for (int i = 0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[m + 1 + j];

        // Merge the temp arrays

        // Initial indices of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarray array
        int k = l;
```

```
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements of L[] if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy remaining elements of R[] if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Main function that sorts arr[l..r] using
// merge()
static void sort(int arr[], int l, int r)
{
    if (l < r) {

        // Find the middle point
        int m = l + (r - l) / 2;

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// A utility function to print array of size n
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
```

```
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 22, 41, 13, 51, 63, 7 };

        System.out.println("Given array is");
        printArray(arr);

        sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array using merge sort is");
        printArray(arr);
    }
}
```



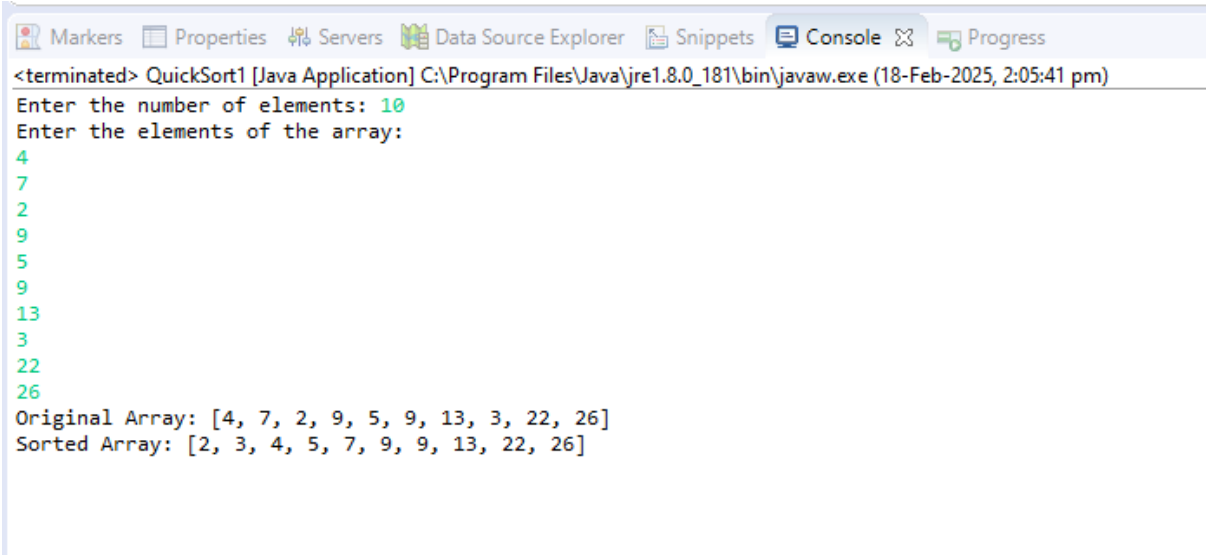
Program – 8 Given an array of integers, sort it by using Quick Sort using Divide and Conquer Approach.

Pivot first element

```
package sort;

import java.util.Arrays;
import java.util.Scanner;

public class QuickSort1 {

    // Function to perform QuickSort
    public static void quickSort(int[] arr, int x, int y) {
        if (x < y) {
            int r = partition(arr, x, y);
            quickSort(arr, x, r); // Sorting the left partition
            quickSort(arr, r + 1, y); // Sorting the right partition
        }
```

```
}

// Hoare's Partition Scheme
public static int partition(int[] A, int x, int y) {
    int pe = A[x]; // Choosing the first element as pivot
    int p = x - 1;
    int q = y + 1;

    while (true) {
        // Move q left until finding an element ≤ pivot
        do {
            q--;
        } while (A[q] > pe);

        // Move p right until finding an element ≥ pivot
        do {
            p++;
        } while (A[p] < pe);

        if (p < q) {
            swap(A, p, q); // Swap elements
        } else {
            return q; // Return partition index
        }
    }
}

// Utility function to swap two elements
public static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

// Main function to take user input and test QuickSort implementation
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    // Taking input from user
    System.out.print("Enter the number of elements: ");
    int n = sc.nextInt();

    int[] arr = new int[n];

    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();
    }

    System.out.println("Original Array: " + Arrays.toString(arr));
```

```
        // Sorting the array using QuickSort
        quickSort(arr, 0, n - 1);

        System.out.println("Sorted Array: " + Arrays.toString(arr));

        sc.close();
    }
}
```



Program- 9 Sort an array of integers by building a max or min heap using the Divide and Conquer Approach. (Assignment- self-study)

Program- 10 Write a program to implement the Multiplication of Large Integers using Divide and Conquer Approach.

```
import java.util.Scanner;
public class Largeintermulti {

    /**
     * Recursively multiplies two numbers using the divide and conquer approach.
     * The algorithm assumes that if the number of digits is odd, we increment it
     * so that we can split the numbers evenly.
     *
     * @param a the first number
     * @param b the second number
     * @return the product of a and b
     */
    public static long multiply(long a, long b) {
        // Base case: if both numbers are single-digit, multiply them directly.
        if (a < 10 && b < 10) {
            return a * b;
        }

        // Determine the maximum number of digits between a and b.
        int n = Math.max(numDigits(a), numDigits(b));
        // If n is odd, increment it so that n becomes even.
        if (n % 2 != 0) {
            n++;
        }

        // p is 10^(n/2)
        long p = (long) Math.pow(10, n / 2);

        // Partition the numbers into high and low parts.
        long a1 = a / p;
        long a2 = a % p;
        long b1 = b / p;
        long b2 = b % p;

        // Recursively compute the four products.
        long A = multiply(a1, b1); // Multiply high parts
        long B = multiply(a2, b1); // Multiply low part of a with high part of b
        long C = multiply(a1, b2); // Multiply high part of a with low part of b
        long D = multiply(a2, b2); // Multiply low parts

        // Combine the results:
        // A is multiplied by 10^n, and (B + C) is multiplied by 10^(n/2)
        long result = A * (long) Math.pow(10, n) + (B + C) * p + D;
        return result;
    }
```

```
    /**
     * Returns the number of digits in a nonnegative number.
     *
     * @param x the number
     * @return the number of digits in x
     */
    public static int numDigits(long x) {
        // Special case for 0.
        if (x == 0) return 1;
        return (int) Math.log10(x) + 1;
    }

    public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);

         // Prompt the user for two numbers.
         System.out.print("Enter the first number: ");
         long a = scanner.nextLong();

         System.out.print("Enter the second number: ");
         long b = scanner.nextLong();
        System.out.println("Multiplying " + a + " and " + b);
        long product = multiply(a, b);
        System.out.println("Product: " + product);
    }
}
```