|  |  |
| --- | --- |
| Ex.No.**1**  **01.10.2022** | Bubble Sort |

|  |
| --- |
| **AIM:** |

To write and execute Java program to perform Bubble Sort.

|  |
| --- |
| **Pseudocode:** |

**1** Bubble sort(A)

**2** for i=1 to A.length-1

**3** for j=A.length to i+1

**4** if A[j]<A[j-1]

**5** Swap(A[j],A[j-1]

|  |
| --- |
| **Explanation:** |

In Line No. 1: -There is a Linear array A .

In Line No. 2: -Then there is a for loop starting from zero to (n-1) that is a.length which

will find size of an array.

In Line No. 3: -Then again there is a for loop starting from (n-1) to (i+1).

In Line No. 4: -We have to check condition A of j is less than A of j minus 1.

In Line No. 5: - If Condition will true then swap the elements of A of j to A of J -1.

|  |
| --- |
| **Example:** |

We take an unsorted array for our example. Bubble sort takes Ο(n2) time so we're keeping it short and precise.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 33 | 27 | 35 | 10 |

Bubble sort starts with very first two elements, comparing them to check which one is greater.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 33 | 27 | 35 | 10 |

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 33 | 27 | 35 | 10 |

We find that 27 is smaller than 33 and these two values must be swapped.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 33 | 27 | 35 | 10 |

The new array should look like this −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 35 | 10 |

Next we compare 33 and 35. We find that both are in already sorted positions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 35 | 10 |

Then we move to the next two values, 35 and 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 35 | 10 |

We know then that 10 is smaller 35. Hence they are not sorted.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 35 | 10 |

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 33 | 10 | 35 |

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 27 | 10 | 33 | 35 |

Notice that after each iteration, at least one value moves at the end.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | 10 | 27 | 33 | 35 |

And when there's no swap require, bubble sorts learns that an array is completely sorted.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 14 | 27 | 33 | 35 | |
| **Program Code:** | | | | |

import java.util.Scanner;

class Bubble {

    static int max=5;

    private int[] a=new int[5];

    public void read() {

        Scanner sc=new Scanner(System.in);

        for(int i=0;i<max;i++) {

            System.out.print("Enter Element of an Array :-");

            a[i]=sc.nextInt();

        }

    }

    public void bubble() {

        int temp = 0;

        for(int i=0; i<max-1; i++){

            for(int j=1; j <max; j++){

                if(a[j-1] > a[j]){

                    temp = a[j-1];

                    a[j-1] = a[j];

                    a[j] = temp;

                }

            }

        }

    }

    public void print() {

        for(int i=0;i<max;i++) {

            System.out.println(a[i]);

        }

    }

}

public class BubbleSort {

    public static void main(String[] args) {

        Bubble b=new Bubble();

        b.read();

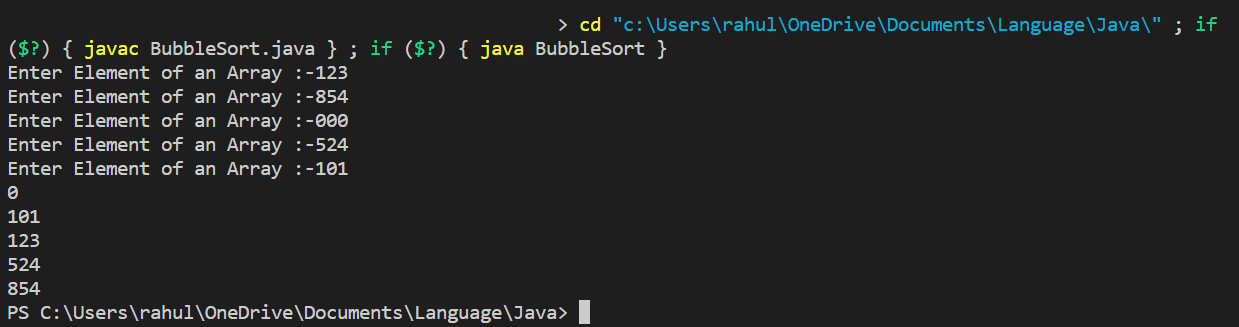
        b.bubble();

        b.print();

    }

}

|  |
| --- |
| **Output Screenshots:** |



|  |
| --- |
| **RESULT:** |

Thus, the programs for the given problem statements has been executed and the results are verified successfully.