**Assignment 1**

Name : Sri Sai Preethi Munnaluri

Class : CSAI-B

Roll No : 50

**Title:**

Write a program to implement Bubble Sort to sort an array of integers in ascending order. Find out Time and space complexity.

**Objective:**

* To understand the mechanism of comparison-based sorting.
* To implement Bubble Sort in C++ or Java.
* To analyze time and space complexity of Bubble Sort.

**Software Requirements:**

* Operating System: Windows/Linux
* Language: C++ or Java
* Compiler: g++/javac

**Hardware Requirements:**

* Processor: 2 GHz or above
* RAM: 4 GB or more
* Disk Space: Minimum 500 MB

**Theory:**

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order**.**

Algorithm:

**Step 1**− Check if the first element in the input array is greater than the next element in the array.

**Step 2**− If it is greater, swap the two elements; otherwise move the pointer forward in the array.

**Step 3**− Repeat Step 2 until we reach the end of the array.

**Step 4**− Check if the elements are sorted; if not, repeat the same process (Step 1 to Step 3) from the last element of the array to the first.

**Step 5**− The final output achieved is the sorted array.

**Pseudocode of bubble sort:**

Start

Repeat for i = 0 to n-1

a. Repeat for j = 0 to n-i-1

- If arr[j] > arr[j+1], swap them

End

**Time Complexity:**

| **Best Case** | **O(n)** |
| --- | --- |
| Average Case | O(n²) |
| Worst Case | O(n²) |

**Space Complexity:**

It sorts data directly within array without additional memory apart from few variables (counter and temp). The memory usage does not grow with the size of input. Regardless of whether you are sorting 10 elements or 10,000, fixed amount of memory is used for variables.

Hence **Space Complexity of bubble sort is O(1).**

**CODE:**

package LAB1;

import java.util.Scanner;

public class BubbleSort

{

    public static void bubbleSort(int[] arr)

    {

        int len=arr.length;

        for(int i=0;i<len;i++)

        {

            for(int j=0;j<len-i-1;j++)

            {

                if(arr[j]>arr[j+1])

                {

                    int temp=arr[j];

                    arr[j]=arr[j+1];

                    arr[j+1]=temp;

                }

            }

        }

    }

    public static void main(String[] args)

    {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array:");

        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();

        }

        bubbleSort(arr);

        System.out.println("Sorted array");

        for (int i = 0; i < arr.length; i++)

        {

            System.out.println(arr[i]);

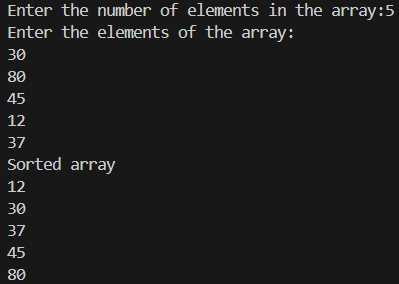
        }

        sc.close();

    }

}

**OUTPUT:**



**Time Complexity Analysis:**

* Best Case (when array is already sorted): O(n)
* Average Case: O(n²)
* Worst Case (when array is in reverse order): O(n²)

**Space Complexity Analysis:**

* Uses no extra space other than a few variables: O(1)

**Conclusion:**

Bubble sort is easy to understand and implement. However, it is inefficient on large lists and is rarely used in practice for performance-critical applications.