Problem Statement

Write a program in java that sorts element in ascending order using insertion sort algorithm.

Algorithm

Algorithm InsertionSort
Input: An array of n number of elements from the user.
Output: Sorted array elements in ascending order of their values.
Step 1: Start
Step 2: For (i=1 to n-1) do
Step 2.1: k ← a[i]
Step 2.2: j← i-1
Step 2.3: While (j>=0) and (k < a[j])
Step 2.3.1: a[j+1] ← a[j]
Step 2.3.2: j ← j-1
Step 2.4: End While
Step 2.5: a[j+1] ← k
Step 3: End For
Step 4: For (i=0 to n-1) do
Step 4.1: Print (a[i])
Step 5: End For
Step 6: Stop

Source Code

```
import java.util.Scanner;
class InsertionSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);			// Create a Scanner object for user input
		System.out.print("Enter the number of elements: "); // Prompt the user to enter the number of elements
		int n = input.nextInt();
		if(n>0)					// Check if n is a positive integer
		{
			int[] arr = new int[n];				// Create an array to store the elements
			System.out.println("Enter the elements:");	// Prompt the user to enter the elements
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
```

```
                        if (isSorted)
                        {
                                System.out.println("The input array is already sorted.");
                        }
                        else
                        {
                                // Perform Insertion Sort
                                for (int i = 1; i < n; i++)
                                {
                                        int k = arr[i];
                                        int j = i - 1;
                                        while (j >= 0 && arr[j] > k)
                                        {
                                                arr[j + 1] = arr[j];
                                                j--;
                                        }
                                        arr[j + 1] = k;
                                }

                                // Display the sorted array in ascending order
                                System.out.println("Sorted Array in Ascending Order:");
                                for (int i = 0; i < n; i++)
                                {
                                        System.out.print(arr[i] + " ");
                                }
                        }
                }
                else                    // Display an error message if n is not positive
                {
                        System.out.println(" !! Enter a positive number of elements !!");
                }
        }
}
```

Result

C:\JAVA\College>*javac A20.java*

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted Array in Ascending Order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 6
Enter the elements:
-9 -6 -4 -10 -8 0
Sorted Array in Ascending Order:
-10 -9 -8 -6 -4 0

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: -5
!! Enter a positive number of elements !!

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 5
Enter the elements:
5 8 1 0 5
Sorted Array in Ascending Order:
0 1 5 5 8

C:\JAVA\College>*java InsertionSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Insertion Sort:*** If the array is not sorted, the program proceeds to perform the Insertion Sort algorithm. It iterates through the array, placing each element in its correct position by comparing it with elements on its left side.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The best-case time complexity of insertion sort is ***O(n)*** when the array is already sorted.
- The average-case time complexity of insertion sort is also ***$O(n^2)$.***
- The worst-case scenario (when the array is not sorted), the time complexity of the program is ***$O(n^2)$.***

Problem Statement

Write a program in java that sorts element in ascending order using merge sort algorithm.

Algorithm

Algorithm MergeSort
Input: An array of n number of elements from the user.
Output: Sorted array elements in ascending order of their values.
Step 1: Start
Step 2: i ← start, j ← mid+1, k ← 0
Step 3: While (I ≤ mid AND j ≤ end)
    Step 3.1: If (Arr[i] ≤ Arr[j]) then
        Step 3.1.1: temp[k++] ← Arr[i++]
        Step 3.1.2: k =k+1
        Step 3.1.3: i = i+1
    Step 3.2: Else
        Step 3.2.1: temp[k++] ← Arr[j++]
        Step 3.2.2: k =k+1
        Step 3.2.3: j = j+1
    Step 3.3: End If
Step 4: End While
Step 5: While (i ≤ mid) do
    Step5.1: temp[k++] ← Arr[i++]
    Step5.2: k =k+1
    Step5.3: i = i+1
Step 6: End While
Step 7: While (j ≤ end) do
    Step7.1: temp[k++] ← Arr[j++]
    Step7.2: k =k+1
    Step7.3: j = j+1
Step 8: End While
Step 9: For (i = start to end) do
    Step 9.1: Arr[i] ← temp[i-start]
Step 10: End For
Step 11: End

Source Code

```
import java.util.Scanner;
class MergeSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the array size and elements from the user
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();
		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
			if (isSorted)
			{
				System.out.println("The input array is already sorted.");
			}
			else
			{
				// Perform merge sort
				mergeSort(arr, 0, n - 1);

				// Output: Display the sorted array
				System.out.println("Sorted array in ascending order:");
				for (int i = 0; i < n; i++)
				{
					System.out.print(arr[i] + " ");
				}
			}
		}
		else
		{
			System.out.println(" !! Enter a positive number of elements !!");
		}
	}

	// Merge Sort function
	static void mergeSort(int[] arr, int l, int r)
	{
```

```
        if (l < r)
        {
            int mid = (l + r) / 2;
            mergeSort(arr, l, mid);
            mergeSort(arr, mid + 1, r);
            merge(arr, l, mid, r);
        }
    }

    // Merge function to combine two sorted subarrays
    static void merge(int[] arr, int l, int mid, int r)
    {
        int n1 = mid - l + 1;
        int n2 = r - mid;

        int[] lArr = new int[n1];
        int[] rArr = new int[n2];

        for (int i = 0; i < n1; i++)
        {
            lArr[i] = arr[l + i];
        }
        for (int j = 0; j < n2; j++)
        {
            rArr[j] = arr[mid + 1 + j];
        }

        int i = 0, j = 0;
        int k = l;
        while (i < n1 && j < n2)
        {
            if (lArr[i] <= rArr[j])
            {
                arr[k] = lArr[i];
                i++;
            }
            else
            {
                arr[k] = rArr[j];
                j++;
            }
            k++;
        }
        while (i < n1)
        {
            arr[k] = lArr[i];
            i++;
            k++;
        }
        while (j < n2)
        {
            arr[k] = rArr[j];
            j++;
            k++;
        }
    }
}
```

Result

C:\JAVA\College>*javac A21.java*

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array in ascending order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 7
Enter the elements:
-5 -7 -3 -8 0 -4 -2
Sorted array in ascending order:
-8 -7 -5 -4 -3 -2 0

C:\JAVA\College>*java MergeSort*
Enter the number of elements: -5
!! Enter a positive number of elements !!

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 5
Enter the elements:
8 6 1 8 2
Sorted array in ascending order:
1 2 6 8 8

C:\JAVA\College>*java MergeSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Merge Sort:*** If the array is not sorted, the program proceeds to perform the Merge Sort algorithm. It recursively divides the array into halves, sorts them separately, and then merges them back together to create a sorted array.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The Time Complexity of merge sort for Best case, average case and worst case is **O(n* logn).**

Problem Statement

Write a program in java that sorts element in ascending order using quick sort algorithm.

Algorithm

Algorithm partition(a, l, h)

Input: An unsorted array of elements of a fixed length, its lower and upper bounds are received as formal arguments from partition( ), called from quicksort( ).

Output: Initializes the "pivot" element, place it at its appropriate position and at the end returns the last index of the element from the array.

Step 1: Start
Step 2: pivot ← a[l], start ← l, end ← h
Step 3: While (start < end) do
  Step 3.1: While (a[start] <= pivot and start < h) do
    Step 3.1.1: start ← start + 1
  Step 3.2: End While
  Step 3.3: While (a[end] > pivot and end > l) do
    Step 3.3.1: end ← end -1
  Step 3.4: End While
  Step 3.5: If (start < end) then
    Step 3.5.1: swap(a[start], a[end]) *//swap( ) swaps the values of the given elements*
  Step 3.6: End If
Step 4: End While
Step 5: swap(a[l], a[end])
Step 6: Return end
Step 7: Stop

Algorithm quickSort(a[ ], l, h)

Input: An unsorted array of elements of a fixed length, its lower and upper bounds are taken as the input from the user.

Output: Recursively call the quickSort( ) and partition( ) time and again to disrupt the original array into smaller sub-arrays and returns the last location of the "pivot" element in the array at the end of partition( ).

Step 1: Start
Step 2: If (l < h) then
  Step 2.1: piv ← partition(a, l, h)
  Step 2.2: quicksort(a, l, piv-1)
  Step 2.3: quicksort(a, piv+1, h)
Step 3: End If
Step 4: Stop

Source Code

```
import java.util.Scanner;
class QuickSort
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the array size and elements from the user
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();

		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Check if the array is already sorted
			boolean isSorted = true;
			for (int i = 1; i < n; i++)
			{
				if (arr[i] < arr[i - 1])
				{
					isSorted = false;
					break;
				}
			}
			if (isSorted)
			{
				System.out.println("The input array is already sorted.");
			}
			else
			{
				// Perform quick sort
				quickSort(arr, 0, n - 1);

				// Output: Display the sorted array
				System.out.println("Sorted array in ascending order:");
				for (int i = 0; i < n; i++)
				{
					System.out.print(arr[i] + " ");
				}
			}
		}
		else
		{
			System.out.println(" !! Enter a positive number of elements !!");
		}
	}
```

```
        // Quick Sort function
        static void quickSort(int[] arr, int l, int h)
        {
                if (l < h)
                {
                        int piv = partition(arr, l, h);
                        quickSort(arr, l, piv - 1);
                        quickSort(arr, piv + 1, h);
                }
        }

        // Partition function to select a pivot and rearrange elements
        public static int partition(int[] arr, int l, int h)
        {
                int pivot = arr[h];
                int i = l - 1;
                for (int j = l; j < h; j++)
                {
                        if (arr[j] <= pivot)
                        {
                                i++;
                                int temp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = temp;
                        }
                }
                int temp = arr[i + 1];
                arr[i + 1] = arr[h];
                arr[h] = temp;
                return i + 1;
        }
}
```

Result

C:\JAVA\College>*javac A22.java*

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array in ascending order:
-8 -7 0 3 4 5 6 9

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 6
Enter the elements:
-8 -6 -1 -2 0 -4
Sorted array in ascending order:
-8 -6 -4 -2 -1 0

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 7
Enter the elements:
9 12 3 48 62 0 7

Sorted array in ascending order:
0 3 7 9 12 48 62

C:\JAVA\College>*java QuickSort*
Enter the number of elements: -6
!! Enter a positive number of elements !!

C:\JAVA\College>*java QuickSort*
Enter the number of elements: 6
Enter the elements:
-10 -9 -8 -6 -4 0
The input array is already sorted.

Discussion

- ***Input Handling:*** The program starts by taking input from the user, including the number of elements in the array and the actual elements themselves.
- ***Input Validation:*** It checks if the input array size is greater than 0 to ensure it's a valid input.
- ***Array Sorting Check:*** The program includes a check to determine if the input array is already sorted. It does so by comparing each element with its adjacent element. If the array is found to be sorted in ascending order, it prints a message indicating that it's already sorted.
- ***Quick Sort:*** If the array is not sorted, the program proceeds to perform the Quick Sort algorithm. Quick Sort is known for its efficiency in sorting and works by selecting a pivot element and partitioning the array into two subarrays. The subarrays are then sorted recursively.
- ***Partition Function:*** The partition function selects a pivot and rearranges elements so that elements less than the pivot are on one side, and elements greater than the pivot are on the other side.
- ***Displaying Sorted Array:*** After sorting, the program displays the sorted array in ascending order.
- ***Input Validation for Non-Positive Size:*** It includes a condition to handle cases where the user enters a non-positive number for the array size, prompting the user to enter a positive number.

Time Complexity

- The best-case time complexity of quick sort is ***O(n*logn)*** occurs when the pivot element is the middle element or near to the middle element.
- The average-case time complexity of quick sort is ***O(n*logn)*** occurs when the array elements are in jumbled order that is not properly ascending and not properly descending.
- In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is **$O(n^2)$**.

Problem Statement

Write a program in java that sorts half of element in ascending and rest half of the elements in descending order.

Algorithm

Algorithm SortAseDes

Input: An unsorted array elements of a fixed length is taken as the input from the user

Output: Divide the array into two halves, such that the first half is sorted in ascending order of their values and the second half in descending order of their values.

Step 1: Start
Step 2: n ← size of array
Step 3: For (i=0 to i=n/2) do
    Step 3.1: For (j=i+1 to n-1) do
        Step 3.1.1: If (a[i] > a[j]) then
            Step 3.1.1.1: temp ← a[j]
            Step 3.1.1.2: a[i] ← a[j]
            Step 3.1.1.3: a[j] ← temp
        Step 3.1.2: End If
    Step 3.2: End For
Step 4: End For
Step 5: For (i=n/2 to i=n - 1) do
    Step 5.1: For (j=i+1 to n-1) do
        Step 5.1.1: If (a[i] < a[j]) then
            Step 5.1.1.1: temp ← a[i]
            Step 5.1.1.2: a[i] ← a[j]
            Step 5.1.1.3: a[j] ← temp
        Step 5.1.2: End If
    Step 5.2: End For
Step 5: End For
Step 6: Print ("Desired Array: ", a[i])
Step 7: Stop

Source Code

```
import java.util.Scanner;
class SortAseDes
{
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);		// Create a Scanner object for user input

		// Input: Read the number of elements
		System.out.print("Enter the number of elements: ");
		int n = input.nextInt();

		if(n>0)
		{
			int[] arr = new int[n];
			System.out.println("Enter the elements:");
			for (int i = 0; i < n; i++)
			{
				arr[i] = input.nextInt();
			}

			// Sort the first half in ascending order
			for (int i = 0; i < n / 2; i++)
			{
				for (int j = i + 1; j < n / 2; j++)
				{
					if (arr[i] > arr[j])
					{
						int temp = arr[i];
						arr[i] = arr[j];
						arr[j] = temp;
					}
				}
			}

			// Sort the second half in descending order
			for (int i = n / 2; i < n - 1; i++)
			{
				for (int j = i + 1; j < n; j++)
				{
					if (arr[i] < arr[j])
					{
						int temp = arr[i];
						arr[i] = arr[j];
						arr[j] = temp;
					}
				}
			}
			// Output: Display the sorted array
			System.out.println("Sorted array with the first half in ascending and the second half in
descending order:");
			for (int i = 0; i < n; i++)
			{
				System.out.print(arr[i] + " ");
			}
```

```
                }
                else
                {
                        System.out.println(" !! Enter a positive number of elements !!");
                }
        }
}
```

Result

C:\JAVA\College>*javac A23.java*

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 8
Enter the elements:
5 6 -7 9 0 4 3 -8
Sorted array with the first half in ascending and the second half in descending order:
-7 5 6 9 4 3 0 -8

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 7
Enter the elements:
-5 -7 -3 -8 0 -4 -2
Sorted array with the first half in ascending and the second half in descending order:
-7 -5 -3 0 -2 -4 -8

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: -6
!! Enter a positive number of elements !!

C:\JAVA\College>*java SortAseDes*
Enter the number of elements: 5
Enter the elements:
8 6 1 8 2
Sorted array with the first half in ascending and the second half in descending order:
6 8 8 2 1

Discussion

- This Java program takes user input for the number of elements and then proceeds to take input for each of those elements, storing them in an integer array.
- The program sorts the first half of the array in ascending order and the second half in descending order using nested loops and a simple bubble sort algorithm. This means that the array is effectively split into two parts: the first half ascending and the second half descending.
- It provides an error message if the user enters a non-positive number of elements, reminding them to input a positive value.
- This program demonstrates basic array manipulation and sorting techniques. However, it uses a less efficient sorting algorithm (bubble sort), which may not be practical for larger datasets.
- Depending on the user's input, the program can create an interesting pattern in the sorted output, where the first half of the array is in ascending order, and the second half is in descending order.