Exercise 7: Financial Forecasting

Program | Debug | Debug Result | Java (11)

```java
import java.util.*;
class SimpleForecast {
    public static void main(String[] args) {
       Scanner s=new Scanner(System.in);
        int n=sc.nextInt();
        int[] revenue=new int[n];

        for (int i=0;i<n;i++) {
            revenue[i]=sc.nextInt();
        }

        System.out.println("Monthly Revenue:");
        for (int r:revenue) {
            System.out.print("₹"+r+" ");
        }
        System.out.println();

        int sum=0;
        for (int i=0;i<revenue.length;i++) {
            sum+=revenue[i];
        }

        int average=sum/revenue.length;
        System.out.println("Forecasted Revenue for Next Month: ₹"+average);
    }
}
```

OUTPUT:

```
5
12000 15000 20202 23023 25202
```

Compiler Message

```
Compilation successful
```

Custom Testcase

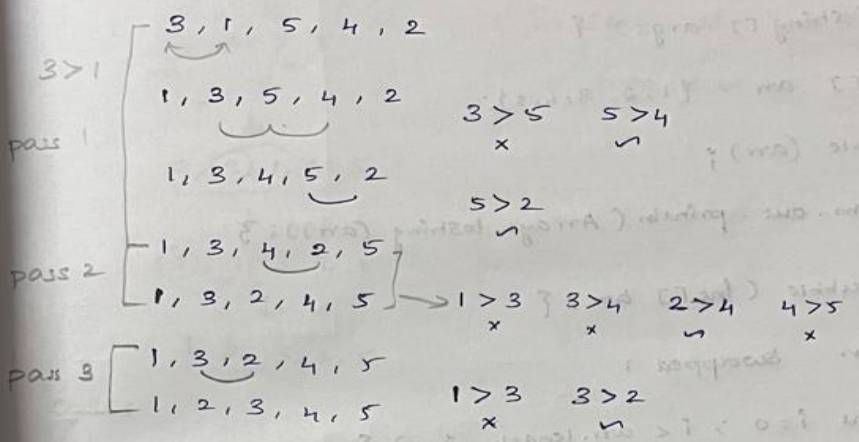
Output

```
Monthly Revenue:
₹12000 ₹15000 ₹20202 ₹23023 ₹25202
Forecasted Revenue for Next Month: ₹19085
```

NOTES:

Sorting :

Bubble Sort : ⇒ also known as Sinking Sort, exchange Sort



Why Bubble Sort ?

With the 1st pass through the array, the largest element came to the end.

With the 2nd pass the second largest element will be the last.

How it works ⇒ Refer notes book.

Space Complexity : O(1) // Constant

No extra space is required. (i.e) copying the a

This Space complexity is also known as inplace Sorting algorithm.

Best Case → Array is Sorted O(N)

Worst Case → Sorting descending order to asce order.

O(N²)

Code // Bubble Sort.

```
public class Main {
    psvm (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        bubble (arr);
        System.out.println (Arrays.toString (arr)); }

Static void bubble (int[] arr) {
    boolean swapped;
    for (int i=0; i < arr.length; i++) {
        swapped = false;
    // for each step max value will be in the last
        for (int j = 1; j < arr.length - i; j++) {
            if (arr[j] < arr[j-1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j-1];
                arr[j-1] = temp;
                swapped = true; } }
        if (!swapped) {
            break;
        } } }
```

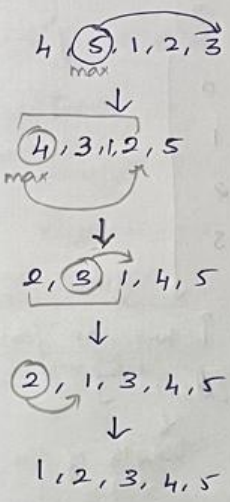
I/p : 1, 2, 3, 4, 5

O/p [1, 2, 3, 4, 5]

I/p : 5, 3, 2, 1, 4

O/p [1, 2, 3, 4, 5]

Selection Sort : ⇒ Refer Notes too.



i) Find the max Element

ii) Swap the last index with that max element.

Code :

```
public class Main { psvm (String[] args) {
    int[] arr = {3, 1, 5, 4, 2};
    selection(arr); → This method stores the sorted array
    Sop(Arrays.toString(arr)); }
```

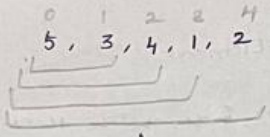
that is the original array is sorted initsey. So, we should not store it in any of the ar

```
static void selection (int[] arr) {
  for(int i=0; i< arr.length; i++) {
    int last = arr.length - i - 1;
    int maxIndex = getmaxIndex (arr, start: 0, last);
    Swap (arr, MaxIndex, last); } }
static void swap (int[] arr, int first, int second) {
    int temp = arr[1st]; arr[1st] = arr[2nd]; arr[2nd] = temp
static int getmaxIndex (int[] arr, int start; int end) {
    int max = start;
    for (int i=start; i<=end; i++) {
        if (arr[max] < arr[i]) {
            max=i; } }
        return max; }
```

I/p : {3, 1, 5, 4, 2}

O/p : [1, 2, 3, 4, 5]

Insertion Sort :



↓

3, 5, 4, 1, 2

↓

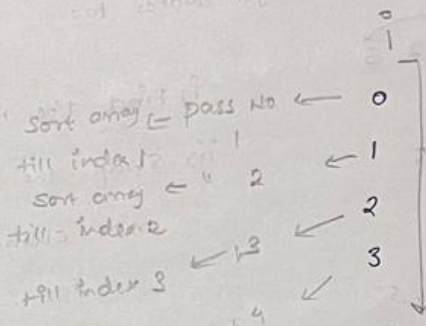
3, 4, 5, 1, 2

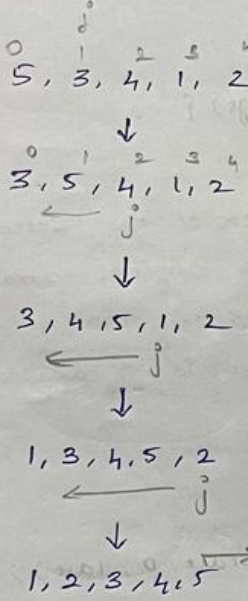
↓

1, 3, 4, 5, 2

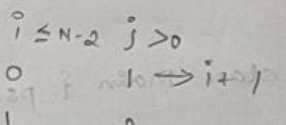
↓

1, 2, 3, 4, 5





When (j > j-1) the L.H.S is sorted.



Complexity : Worst case : $O(N^2)$

Best Case : $O(N)$ ⇒ Array is already sorted

Why we use insertion Sort ?

→ Steps gets reduced compared to Binary Search

→ Stable

→ Used for Smaller values of N

→ Used when Array is partially sorted

Code :// Insertion Sort.

```
public static void main (String args[]) {
    int[] arr = {5, 3, 4, 1, 2};
    insertion (arr);
    System.out.println (Arrays.toString(arr)); }
Static void insertion (int[] arr) {
    for (int i=0 ; i< arr.length-1 ; i++) {
        for (int j=i+1 ; j>0 ; j--) {
            if (arr[j]< arr[j-1]) {
                swap (arr, j, j-1); }
            else {
                break; } }
Static void swap (int[] arr, int first, int start) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp; }
```

I/p :

5, 3, 4, 1, 2

o/p :

[1, 2, 3, 4, 5]

In Insertion sort the number of swaps is reduced as compared to the bubble sort

Insertion sort is a stable sorting Algorithm.

Used for the smaller values of n.

Works good when array is partially sorted.

Cyclic Sort: ⇒ In all other Sorting the worst case time complexity is $O(N^2)$.

When given numbers from range 1 to N ⇒ use Cyclic Sort.

Example: 3, 5, 2, 1, 4 ⇒ 1 to N then Cyclic Sort

1, 2, 3, 4, 5 ⇒ Sorted Array so Bubble Sort.

3, 5, 2, 1, 4

After Sorting:

0	1	2	3	4
1	2	3	4	5

⟶ index = value − 1

4 = 5 − 1

0	1	2	3	4
3	5	2	1	4

Swap coz 3 − 1 = 2 index

0	1	2	3	4
2	5	3	1	4

Swap coz 2 − 1 = 1st index

0	1	2	3	4
5	2	3	1	4

Swap coz 5 − 1 = 4th index

0	1	2	3	4
4	2	3	1	5

Swap coz 4 − 1 = 3rd index

1, 2, 3, 4, 5 ⇒ Finally Sorted.

0 1 2 3 4

└ index = value − 1

4 = 5 − 1

4 = 4

Complexity: $O(N)$

↳ Worst Case.

Code // Refer Notes :

```
public class CyclicSort {
    public static void main (String[] args) {
        int[] arr = {3, 5, 2, 1, 4};
        sort (arr);
        System.out.println (Array.toString (arr)); }
    static void sort (int[] arr) {
        int i=0;
        while (i< arr.length) {
            int correct = arr[i]-1;
            if ( arr[i] != arr[correct]){
                swap ( arr, i, correct); }
            else {
                i++;
            } } }
    static void swap (int[] arr, int first, int second){
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp; } }
```

I/p :

3, 5, 2, 1, 4

⇩

1 to N

O/p :

[1, 2, 3, 4, 5]

0 1 2 3 4

↳ Index = Val

4 = 5 - 1

4 = 4