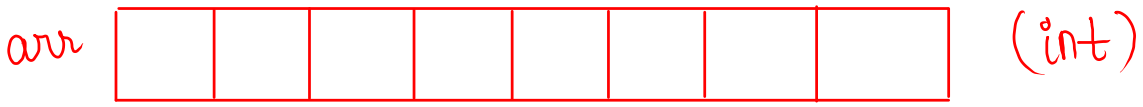
⇒ Sorting (arranging in a particular order)



Type of sorting

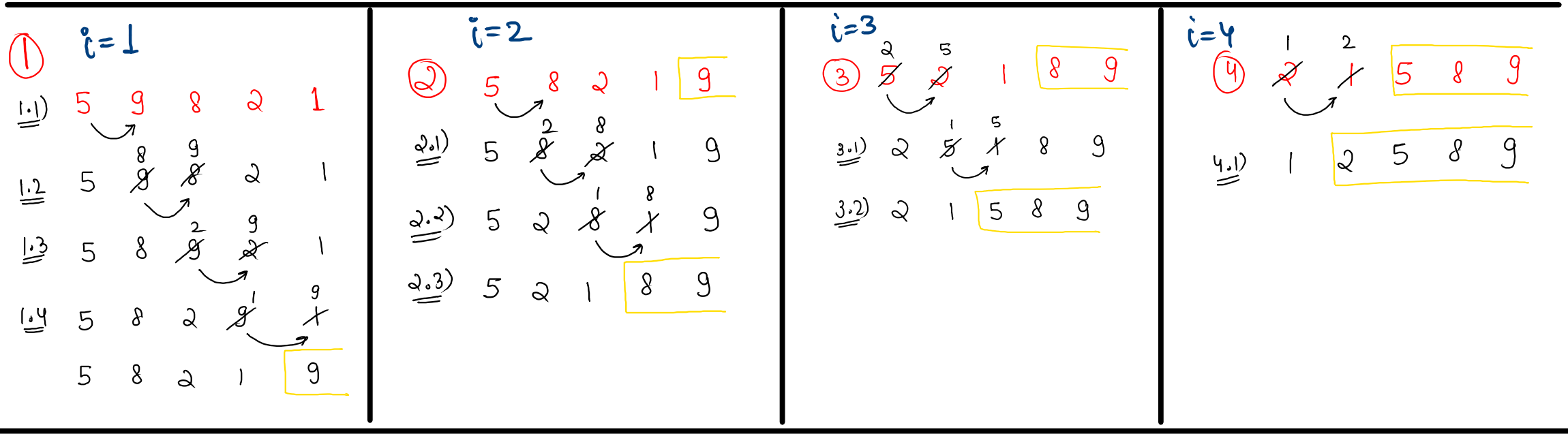
- Bubble sort
- Insertion sort
- Selection sort

Time Complexity
$O(N^2)$
where N is size of array

⇒ Bubble Sorting (trying to pick the largest element and place it to the rightmost part of unsorted array)

n=5

0	1	2	3	4
5	9	8	2	1



Exp: Each time compare 2 adjacent element and if larger element is on the left then swap both values

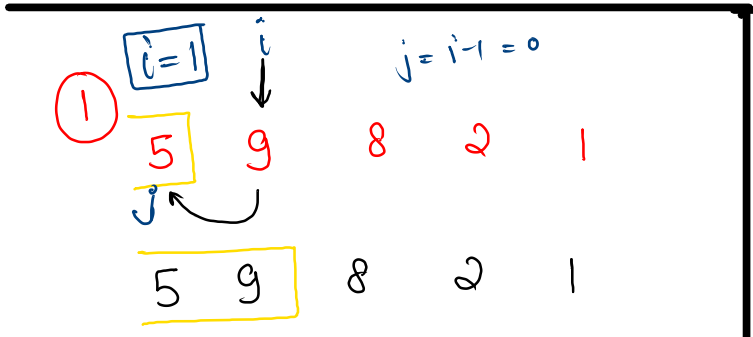
Code

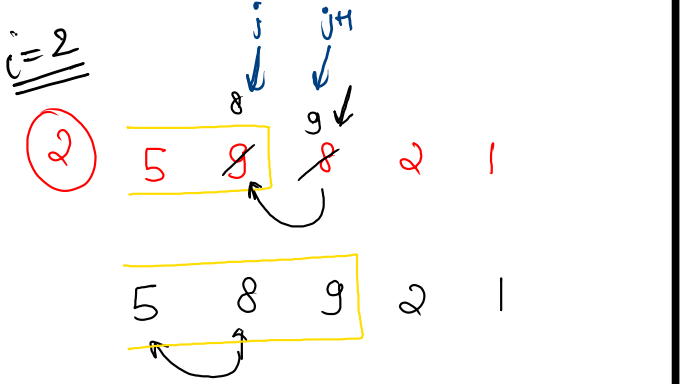
```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    bubbleSort(arr, n);
}

public static void bubbleSort(int[] arr, int n) {
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            if ( arr[j] > arr[j + 1] ) {
                swap(arr, j, j + 1);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        System.out.print(arr[i] + " ");
    }
}

public static void swap(int[] arr, int x, int y) {
    int temp = arr[x];
    arr[x] = arr[y];
    arr[y] = temp;
}
```

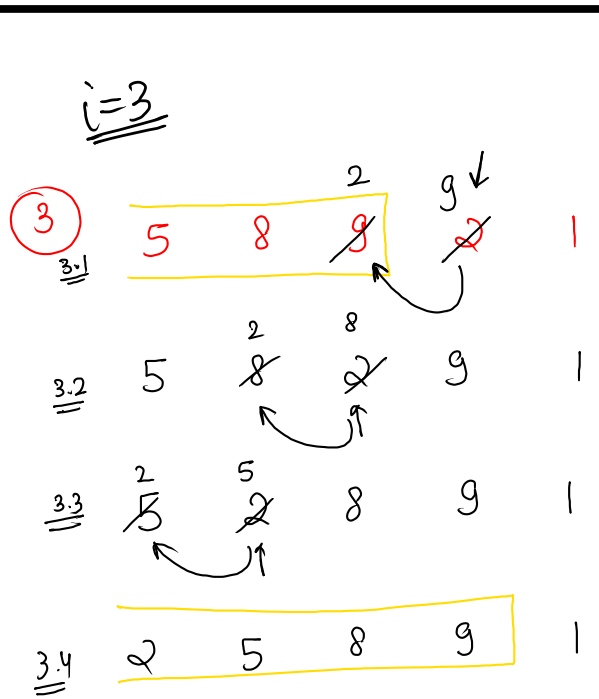
⇒ Inserting Sort (Pick the first element of unsorted array, and place it at the correct position within sorted array)

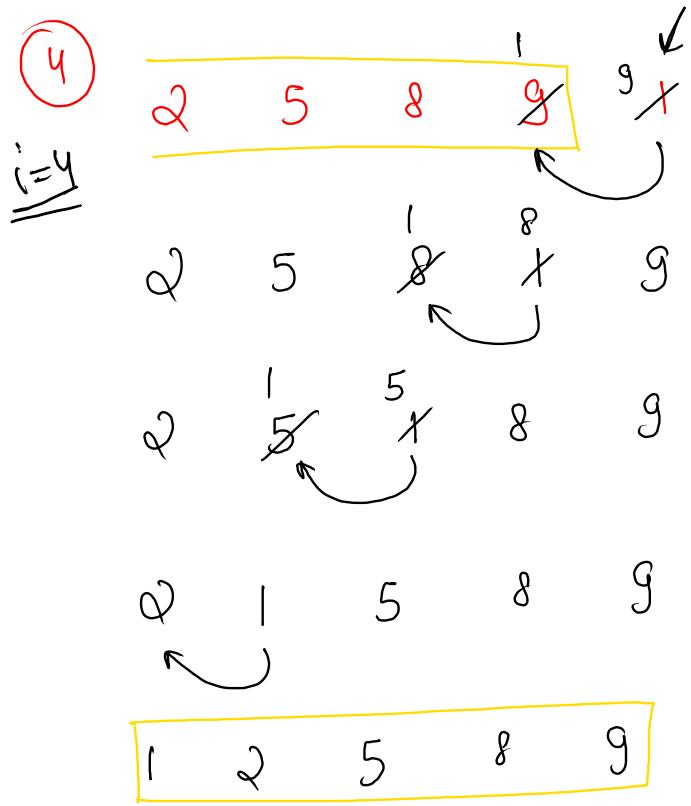
n=5

0	1	2	3	4
5	9	8	2	1



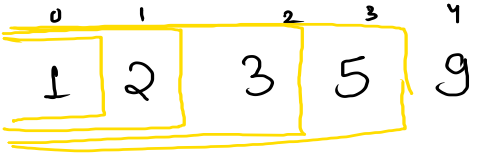






Code

```java
public static void insertionSort(int[] arr, int n) {
    // main logic
    for (int i = 1; i < n; i++) {   // n - 1 times
        for (int j = i - 1; j >= 0; j--) {
            if (arr[j] > arr[j + 1]) {
                swap(arr, j, j + 1);
            } else {
                break;
            }
        }
    }

    // print array
    for (int i = 0; i < n; i++) {
        System.out.print(arr[i] + " ");
    }
}
```



i=1 j=0 (2 > 9) false

i=2 j=1 (9 > 5) swap
j=0 (2 > 5) false

i=3 j=2 (9 > 1) swap
j=1 (5 > 1) swap
j=0 (2 > 1) swap

i=4 j=3 (9 > 3) swap
j=2 (5 > 3) swap
j=1 (2 > 3) false
j=0