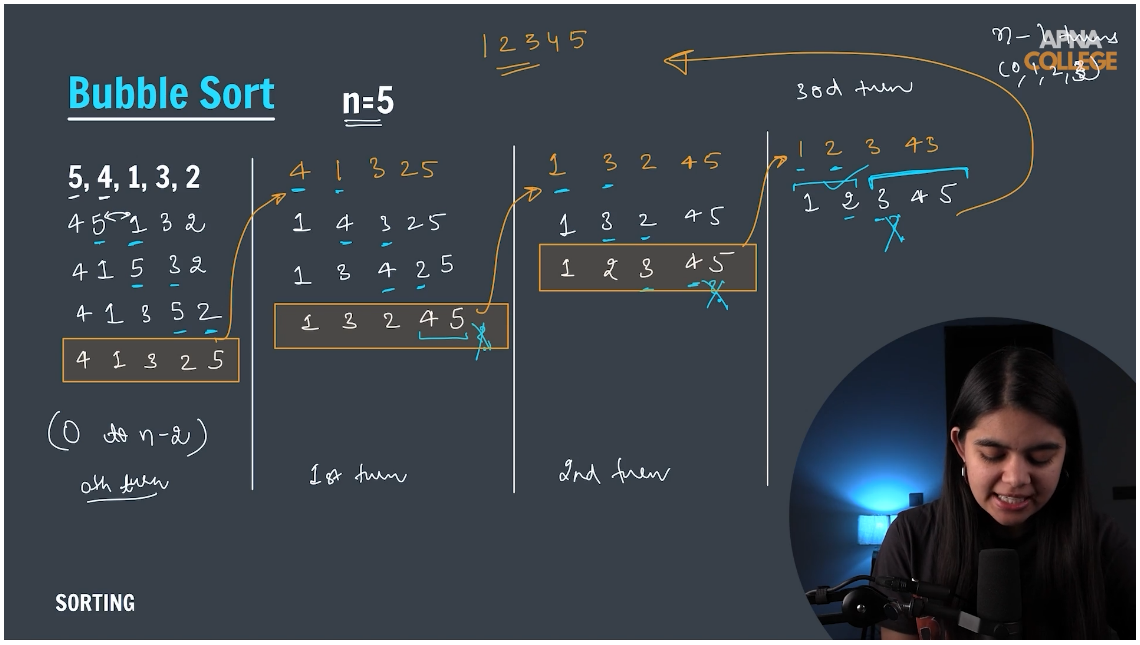
# Sorting

Arranging in order

▼ Bubble Sort :

- Logic : Is to compare and swap two indexes and push the bigger value to the end .
    - Imagine boiling a liquid. The large elements would come up to the last of the array
- In a bubble the sort the bigger elements would move to the end of the array
- Starting from the beginning of the array where the first element would be check with all other indexes one by one to find which element is bigger in size
- Total number of turns =  0 to n-2



- Time Complexity : $O(n^2)$

▼ Selection Sort :

- Pick the smallest ( form unsorted ), put it in the beginning of the array
- Create a variable Smallest and initialize it with the - INFINITY i.e. Integer.MIN_Value
- Nested Loops :

  ```
  for ( int i = 0 to n-2 )
  ```

  ```
  for ( j = i+1 to n-1 )
  ```

  - Time Complexity : $O(n^2)$

▼ Insertion Sort :

- To Sort an element such that it gets arranged accordingly into an already semi-sorted array.
- Creating a temporary memory and one by one by index wise we take one element put it inside a temporary memory and then compare the other elements of array to the temp element
- Until we could reach the sorted part of the array we compare and once the temp variable is greater than some element we assume that we have reached the sorted array part
- We than shift the values of the array such that the temp variable is accommodated to its appropriate sorting position.
- Time Complexity : $O(n^2)$

▼ Inbuilt Sort :

- Inbuilt functions used to Sort in JAVA

```
import java.util.*;     || import java.util*Arrays;
Arrays.sort(arr);
```

- Time Complexity : $O(nlogn)$ i.e. better than all above Sorting algorithm

```
Collections.reverseOrder()
```

- It is used to reverse the order of the array
    - The revere Order uses an internal comparator to compare the values

▼ Counting Sort :

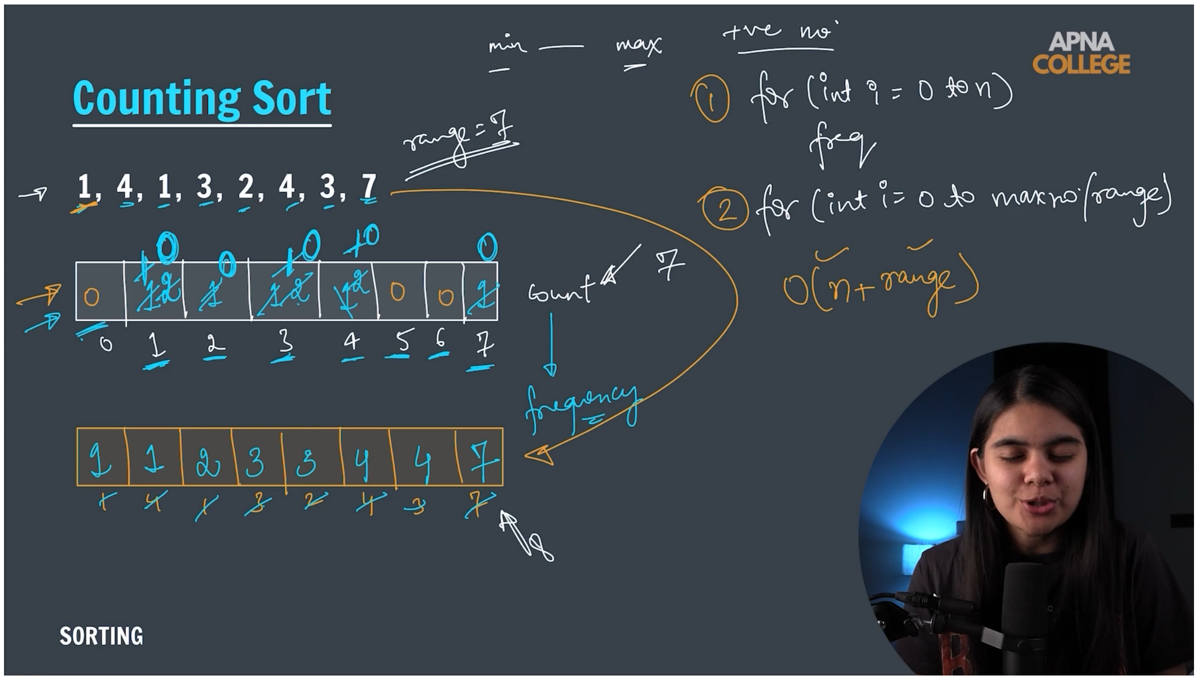
- It is used in case of smaller range of positive number
- It is very efficient with even linear time complexity
- A count array is created in which :
    - We store the frequency of numbers that how many times does an number repeats itself and add that number into the array
- Two loop would be used
    - First loop : Frequency count

    ```
    for (int i=0 to n)
    ```

    - Second loop : For sorting

    ```
    for (int i =0 to max no./range)
    ```

- Time Complexity : $O(n + range)$ i.e. time complexity would be proportional to "n" as well as "range"
    - Therefore smaller range of numbers give us lesser time complexity



## Counting Sort

1, 4, 1, 3, 2, 4, 3, 7

SORTING