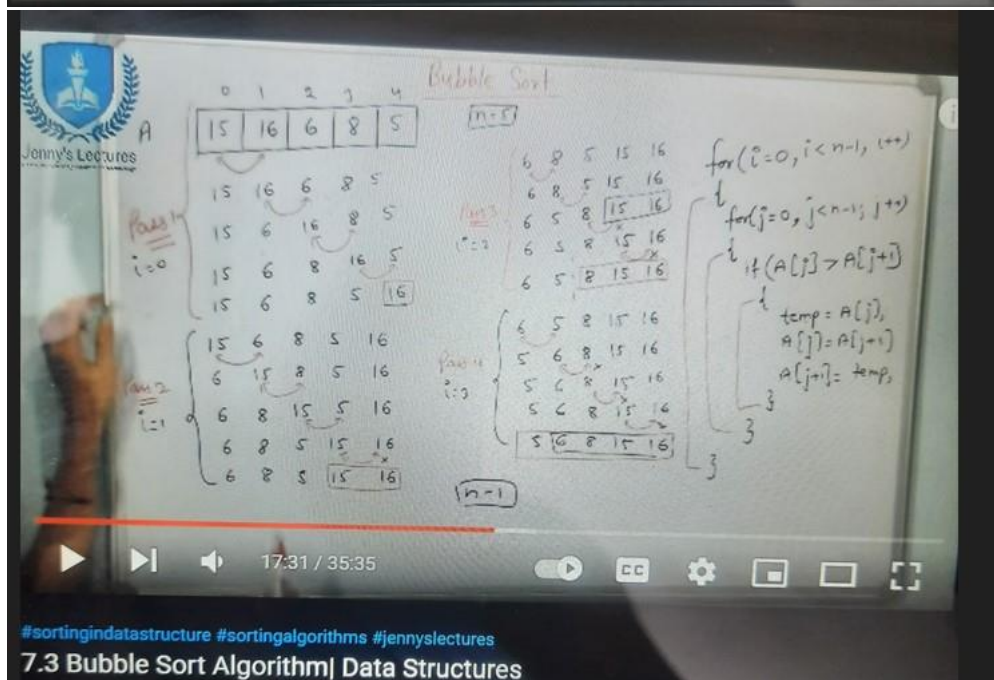
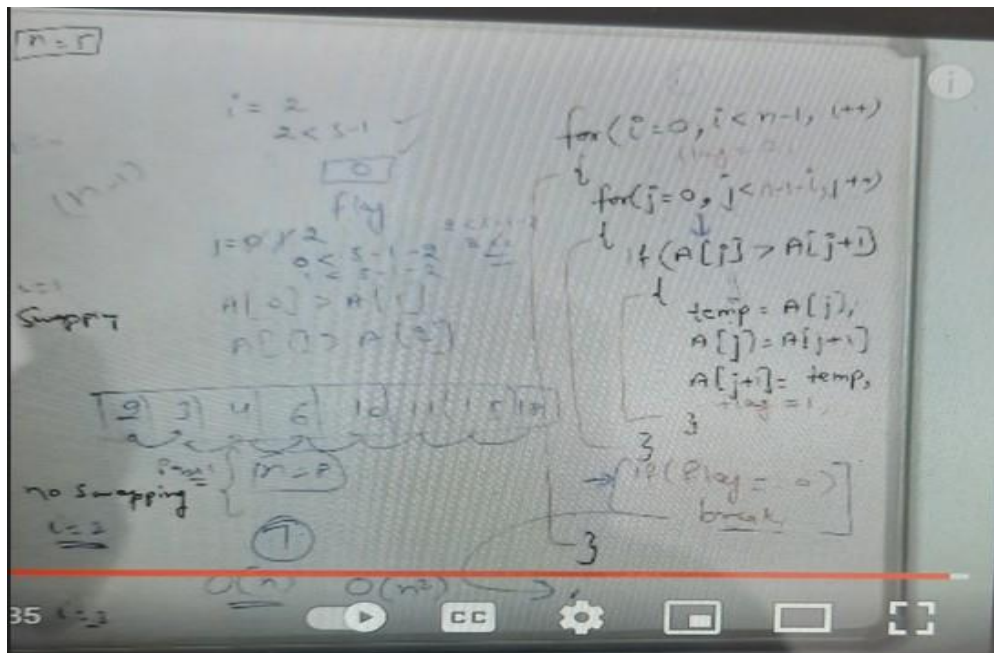


### Algorithm Overview:

### Bubble sort:

[https://www.youtube.com/watch?v=o4bAoo\\_gFBU&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxUL8LU&index=97](https://www.youtube.com/watch?v=o4bAoo_gFBU&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxUL8LU&index=97)



## Insertion sort:

[https://www.youtube.com/watch?v=yCxV0kBPpA6M&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLl8LU&index=99](https://www.youtube.com/watch?v=yCxV0kBPpA6M&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=99)

Given array is divided into two parts. Sorted array and Unsorted array.

Insertion Sort Algorithm | Data Structure

Array:  $[5, 4, 10, 1, 6, 2]$ ,  $n=6$

Sorted part:  $[4, 5, 10]$ , Unsorted part:  $[1, 6, 2]$

Code:

```
for (i = 1; i < n; i++)  
{  
    temp = a[i];  
    j = i - 1;  
    while (j >= 0 && a[j] > temp)  
    {  
        a[j+1] = a[j];  
        j--;  
    }  
    a[j+1] = temp;  
}
```

Example steps:

- $i=1$ ,  $j=0$ ,  $a[1]=4$ ,  $a[0]=5$ ,  $a[1] < a[0]$ , swap  $a[0]$  and  $a[1]$ . Array:  $[4, 5, 10, 1, 6, 2]$
- $i=2$ ,  $j=1$ ,  $a[2]=10$ ,  $a[1]=5$ ,  $a[2] > a[1]$ , no swap. Array:  $[4, 5, 10, 1, 6, 2]$
- $i=3$ ,  $j=2$ ,  $a[3]=1$ ,  $a[2]=10$ ,  $a[3] < a[2]$ , swap  $a[2]$  and  $a[3]$ . Array:  $[4, 1, 5, 10, 6, 2]$
- $i=3$ ,  $j=1$ ,  $a[3]=10$ ,  $a[1]=5$ ,  $a[3] > a[1]$ , no swap. Array:  $[4, 1, 5, 10, 6, 2]$
- $i=3$ ,  $j=0$ ,  $a[3]=10$ ,  $a[0]=4$ ,  $a[3] > a[0]$ , no swap. Array:  $[4, 1, 5, 10, 6, 2]$
- $i=4$ ,  $j=3$ ,  $a[4]=6$ ,  $a[3]=10$ ,  $a[4] < a[3]$ , swap  $a[3]$  and  $a[4]$ . Array:  $[4, 1, 5, 6, 10, 2]$
- $i=4$ ,  $j=2$ ,  $a[4]=10$ ,  $a[2]=5$ ,  $a[4] > a[2]$ , no swap. Array:  $[4, 1, 5, 6, 10, 2]$
- $i=4$ ,  $j=1$ ,  $a[4]=10$ ,  $a[1]=5$ ,  $a[4] > a[1]$ , no swap. Array:  $[4, 1, 5, 6, 10, 2]$
- $i=4$ ,  $j=0$ ,  $a[4]=10$ ,  $a[0]=4$ ,  $a[4] > a[0]$ , no swap. Array:  $[4, 1, 5, 6, 10, 2]$
- $i=5$ ,  $j=4$ ,  $a[5]=2$ ,  $a[4]=10$ ,  $a[5] < a[4]$ , swap  $a[4]$  and  $a[5]$ . Array:  $[4, 1, 5, 6, 2, 10]$
- $i=5$ ,  $j=3$ ,  $a[5]=10$ ,  $a[3]=6$ ,  $a[5] > a[3]$ , no swap. Array:  $[4, 1, 5, 6, 2, 10]$
- $i=5$ ,  $j=2$ ,  $a[5]=10$ ,  $a[2]=5$ ,  $a[5] > a[2]$ , no swap. Array:  $[4, 1, 5, 6, 2, 10]$
- $i=5$ ,  $j=1$ ,  $a[5]=10$ ,  $a[1]=5$ ,  $a[5] > a[1]$ , no swap. Array:  $[4, 1, 5, 6, 2, 10]$
- $i=5$ ,  $j=0$ ,  $a[5]=10$ ,  $a[0]=4$ ,  $a[5] > a[0]$ , no swap. Array:  $[4, 1, 5, 6, 2, 10]$

## Selection sort:

[https://www.youtube.com/watch?v=9oWd4VJOwr0&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLl8LU&index=100](https://www.youtube.com/watch?v=9oWd4VJOwr0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=100)

Selection Sort

Array:  $[7, 4, 10, 8, 3, 1]$ ,  $n=6$

Sorted part:  $[1, 3, 4, 7, 8, 10]$ , Unsorted part:  $[7, 4, 10, 8, 3, 1]$

Code:

```
for (i = 0; i < n - 1; i++)  
{  
    int min = i;  
    for (j = i + 1; j < n; j++)  
    {  
        if (a[j] < a[min])  
            min = j;  
    }  
    swap(a[i], a[min]);  
}
```

Example steps:

- $i=0$ ,  $j=1$ ,  $a[0]=7$ ,  $a[1]=4$ ,  $a[0] > a[1]$ , swap  $a[0]$  and  $a[1]$ . Array:  $[4, 7, 10, 8, 3, 1]$
- $i=0$ ,  $j=2$ ,  $a[0]=4$ ,  $a[2]=10$ ,  $a[0] < a[2]$ , no swap. Array:  $[4, 7, 10, 8, 3, 1]$
- $i=0$ ,  $j=3$ ,  $a[0]=4$ ,  $a[3]=8$ ,  $a[0] < a[3]$ , no swap. Array:  $[4, 7, 10, 8, 3, 1]$
- $i=0$ ,  $j=4$ ,  $a[0]=4$ ,  $a[4]=3$ ,  $a[0] > a[4]$ , swap  $a[0]$  and  $a[4]$ . Array:  $[3, 7, 10, 8, 4, 1]$
- $i=0$ ,  $j=5$ ,  $a[0]=3$ ,  $a[5]=1$ ,  $a[0] > a[5]$ , swap  $a[0]$  and  $a[5]$ . Array:  $[1, 7, 10, 8, 4, 3]$
- $i=1$ ,  $j=2$ ,  $a[1]=7$ ,  $a[2]=10$ ,  $a[1] < a[2]$ , no swap. Array:  $[1, 7, 10, 8, 4, 3]$
- $i=1$ ,  $j=3$ ,  $a[1]=7$ ,  $a[3]=8$ ,  $a[1] < a[3]$ , no swap. Array:  $[1, 7, 10, 8, 4, 3]$
- $i=1$ ,  $j=4$ ,  $a[1]=7$ ,  $a[4]=4$ ,  $a[1] > a[4]$ , swap  $a[1]$  and  $a[4]$ . Array:  $[1, 4, 10, 8, 7, 3]$
- $i=1$ ,  $j=5$ ,  $a[1]=4$ ,  $a[5]=3$ ,  $a[1] > a[5]$ , swap  $a[1]$  and  $a[5]$ . Array:  $[1, 3, 10, 8, 7, 4]$
- $i=2$ ,  $j=3$ ,  $a[2]=10$ ,  $a[3]=8$ ,  $a[2] > a[3]$ , swap  $a[2]$  and  $a[3]$ . Array:  $[1, 3, 8, 10, 7, 4]$
- $i=2$ ,  $j=4$ ,  $a[2]=8$ ,  $a[4]=7$ ,  $a[2] > a[4]$ , swap  $a[2]$  and  $a[4]$ . Array:  $[1, 3, 7, 10, 8, 4]$
- $i=2$ ,  $j=5$ ,  $a[2]=7$ ,  $a[5]=4$ ,  $a[2] > a[5]$ , swap  $a[2]$  and  $a[5]$ . Array:  $[1, 3, 4, 10, 8, 7]$
- $i=3$ ,  $j=4$ ,  $a[3]=10$ ,  $a[4]=8$ ,  $a[3] > a[4]$ , swap  $a[3]$  and  $a[4]$ . Array:  $[1, 3, 4, 8, 10, 7]$
- $i=3$ ,  $j=5$ ,  $a[3]=8$ ,  $a[5]=7$ ,  $a[3] > a[5]$ , swap  $a[3]$  and  $a[5]$ . Array:  $[1, 3, 4, 7, 10, 8]$
- $i=4$ ,  $j=5$ ,  $a[4]=10$ ,  $a[5]=8$ ,  $a[4] > a[5]$ , swap  $a[4]$  and  $a[5]$ . Array:  $[1, 3, 4, 7, 8, 10]$

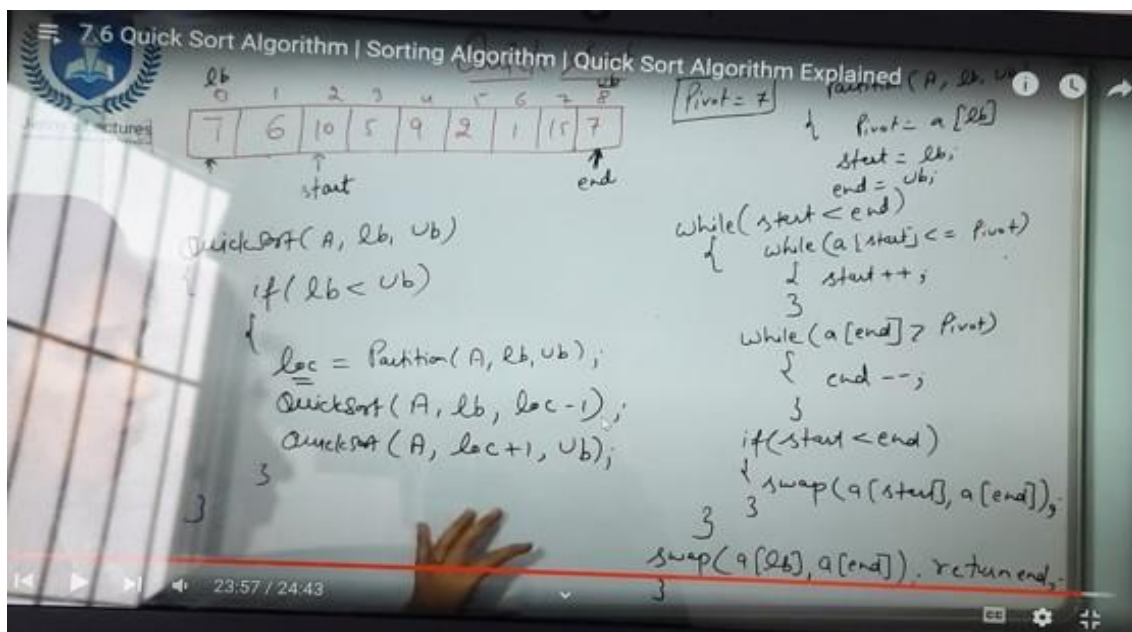
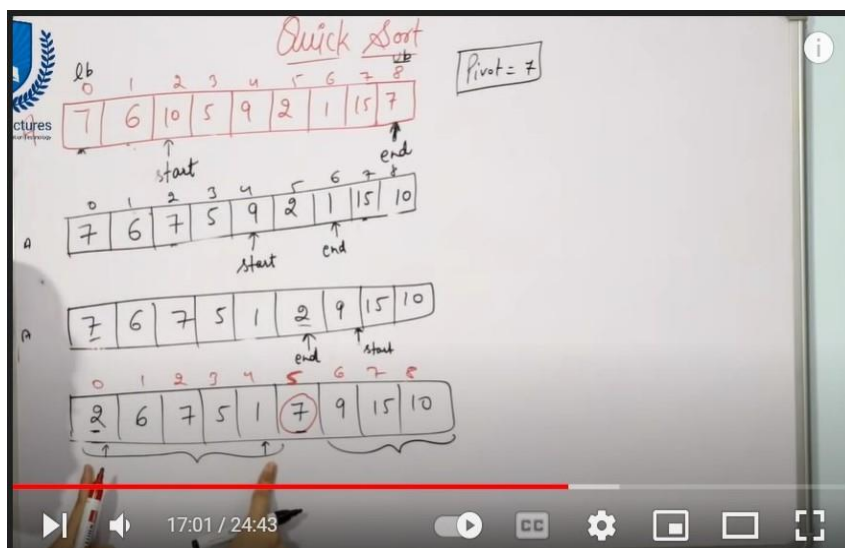
## Quicksort:

[https://www.youtube.com/watch?v=QN9hnmAgmOc&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLL8LU&index=100](https://www.youtube.com/watch?v=QN9hnmAgmOc&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLL8LU&index=100)

Quicksort algorithm is based on the divide and conquer approach where an array is divided into subarrays by selecting a pivot element.

While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side.

The same process is continued for both left and right subarrays. Finally, sorted elements are combined to form a sorted array.





## Merge sort:

[https://www.youtube.com/watch?v=jlHkDBEumP0&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLI8LU&iAĀĀĀĀĀindex=101](https://www.youtube.com/watch?v=jlHkDBEumP0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLI8LU&iAĀĀĀĀĀindex=101)

[illegible]

The whiteboard content includes a diagram of the Merge Sort algorithm and handwritten code.

**Diagram:**

- Initial array: `[15, 5, 24, 8, 1, 3, 16, 10, 20]`
- Splitting phase (left to right):
  - `[15, 5, 24]` and `[8, 1]`
  - `[15, 5]` and `[24]`; `[8]` and `[1]`
  - `[15]` and `[5]`; `[24]`; `[8]` and `[1]`
  - `[15]`; `[5]`; `[24]`; `[1]` and `[8]`
  - `[5]` and `[15]`; `[24]`; `[1]` and `[8]`
  - `[5, 15]`; `[24]`; `[1]` and `[8]`
  - `[5, 15, 24]`; `[1]` and `[8]`
  - `[1, 5, 8, 15, 24]`
- Merging phase (right to left):
  - `[3, 16]` and `[10, 20]`
  - `[3, 16]` and `[10]`; `[16]` and `[20]`
  - `[3, 10, 16]` and `[20]`
  - `[3, 10, 16, 20]`
  - `[5, 8]` and `[10, 20]`
  - `[5, 10, 16, 20]`
  - `[1, 3, 5, 8, 10, 15, 20, 24]`

**Handwritten Code:**

```

i = lb;
j = mid + 1;
k = lb;
while (i <= mid && j <= ub)
{
    if (a[i] <= a[j])
    {
        b[k] = a[i];
        i++;
    }
    else
    {
        b[k] = a[j];
        j++;
    }
    k++;
}
if (i > mid)
{
    while (j <= ub)
    {
        b[k] = a[j];
        j++; k++;
    }
}
else
{
    while (i <= mid)
    {
        b[k] = a[i];
        i++; k++;
    }
}

```

At the bottom, a final sorted array is shown: `[1, 3, 5, 8, 10, 15, 20, 24]`.

## Shell sort:

[https://www.youtube.com/watch?v=9crZRd8GPWM&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLl8LU&index=105](https://www.youtube.com/watch?v=9crZRd8GPWM&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=105)

7.11 Shell Sort algorithm | sorting algorithms | Full explanation with

23	29	15	19	31	7	9	5	2
----	----	----	----	----	---	---	---	---

Lectures

Pass 1

23	7	15	19	31	29	9	5	2
23	7	9	19	31	29	15	5	2
23	7	9	5	31	29	15	19	2
23	7	9	5	2	29	15	19	31
2	7	9	5	23	29	15	19	31

Pass 2

2	5	9	7	23	29	15	19	31
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19

Pass 3

2	5	7	9	15	19	23	29	31
---	---	---	---	----	----	----	----	----

gap sequence  $\rightarrow 4, 2$

$\lfloor \frac{N}{2} \rfloor$

22:00 / 34:07

7.11 Shell Sort algorithm | sorting algorithms | Full explanation with code | data struc...

23	29	15	19	31	7	9	5	2
----	----	----	----	----	---	---	---	---

Lectures

Pass 1

23	7	15	19	31	29	9	5	2
23	7	9	19	31	29	15	5	2
23	7	9	5	31	29	15	19	2
23	7	9	5	2	29	15	19	31
2	7	9	5	23	29	15	19	31

Pass 2

2	5	9	7	23	29	15	19	31
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19
2	5	9	7	15	2	29	23	19

Pass 3

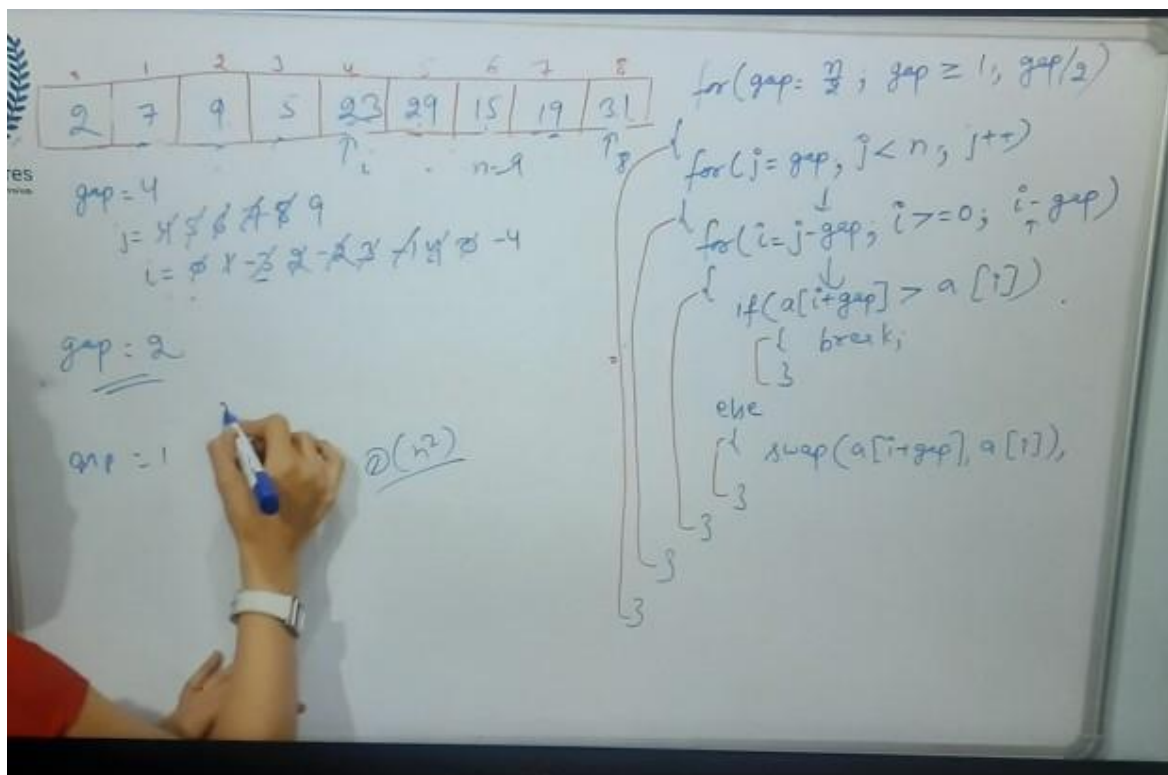
2	5	7	9	15	19	23	29	31
---	---	---	---	----	----	----	----	----

gap sequence  $\rightarrow 4, 2, 1$

$\lfloor \frac{N}{2} \rfloor$

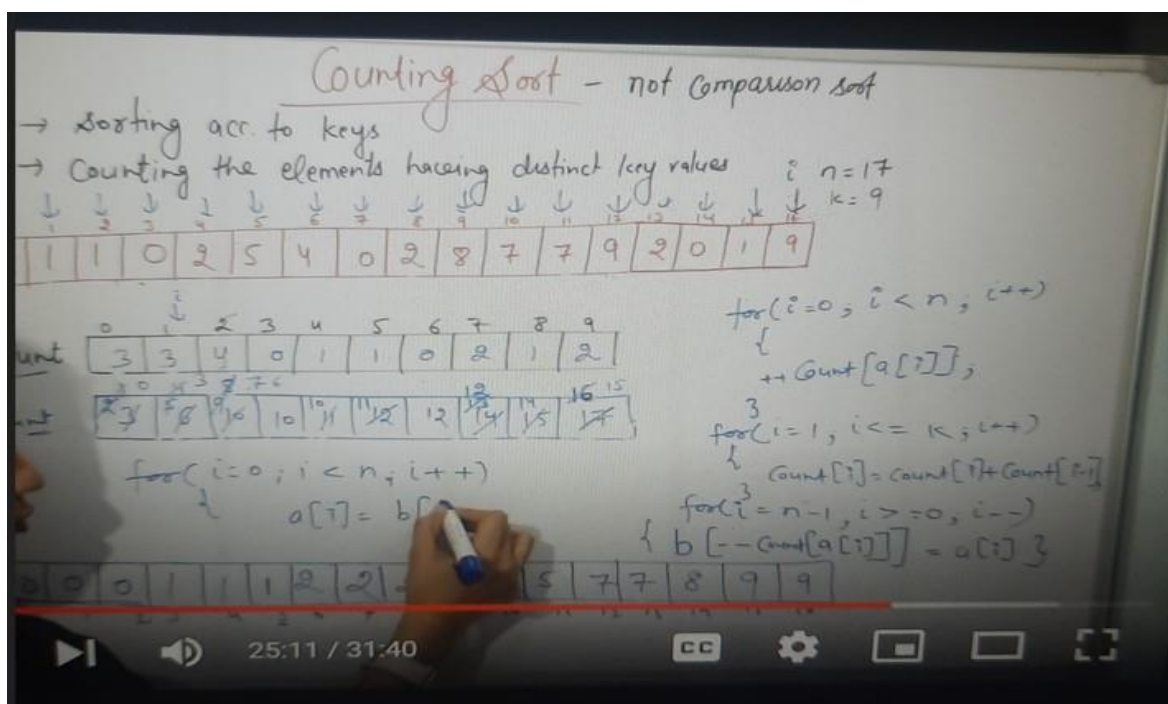
```
for (gap = N/2; gap > 0; gap = gap/2)
{
    for (j = gap; j < n; j++)
    {
        for (i = j - gap; i >= 0; i = i - gap)
        {
            if (a[i + gap] > a[i])
            {
                break;
            }
            else
            {
                swap(a[i + gap], a[i]);
            }
        }
    }
}
```

26:09 / 34:07



Counting sort:

[https://www.youtube.com/watch?v=pEJiGC-ObQE&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuLI8LU&index=106](https://www.youtube.com/watch?v=pEJiGC-ObQE&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLI8LU&index=106)





Sort - not Comparison sort  $n + k + n + n$

using distinct key values  $O(n+k)$

$n = 17$   $k = 9$

CountSort( $a, n, k$ )

int Count[ $k+1$ ]

b[ $n$ ]

```

for (i = 0; i < n; i++)
{
    ++Count[a[i]];
}

for (i = 1; i <= k; i++)
{
    Count[i] = Count[i] + Count[i-1];
}

for (i = n-1; i >= 0; i--)
{
    b[--Count[a[i]]] = a[i];
}

```

Initial array: 8, 7, 7, 9, 2, 0, 1, 9

Count array: 6, 7, 8, 9

Sorted array: 0, 2, 1, 2

Final array: 0, 2, 1, 2, 12, 13, 14, 15, 16, 17

Radix/Bucket sort:

[https://www.youtube.com/watch?v=Il45xNUHGp0&list=PLdo5W4Nhv31bbKJzrsKfMpo\\_grxuL8LU&index=107](https://www.youtube.com/watch?v=Il45xNUHGp0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuL8LU&index=107)

13 Radix Sort - Easiest explanation with code | sorting algorithm

Initial array: 432, 008, 530, 090, 088, 231, 011, 045, 677, 199

Pass 1: Count: 2, 2, 1, 0, 0, 1, 0, 1, 2, 1

Pass 2: Count: 1, 1, 0, 3, 1, 0, 0, 1, 1, 2

Pass 3: Count: 5, 1, 1, 0, 1, 1, 1, 0, 0, 0

Pass 4: Count: 8, 5, 5, 7, 7, 8, 9, 10, 10, 10

Final array: 008, 011, 045, 088, 090, 199, 231, 432, 530, 677

$n=10$

7	199
---	-----

↑

1
---

↑

199
-----

↑

1	2
---	---

↑

8	10
---	----

↑

199
-----

↑

0	10
---	----

↑

30	677
----	-----

↑

```

radixSort(a, n)
{
    int max = getMax(A, n)
    for (pos = 1; max/pos > 0; pos *= 10)
    {
        CountSort(A, n, pos)
    }
}

CountSort(int A[], int n, int pos)
{
    int Count[10] = {0};
    for (i = 0; i < n; i++)
        ++Count[(a[i]/pos)%10];
    for (i = 1; i <= 9; i++)
        Count[i] = Count[i] + Count[i-1];
    for (i = n-1; i >= 0; i--)
        b[Count[(a[i]/pos)%10]] = a[i];
}
  
```

$n=10$

5	677	199
---	-----	-----

↑

1	2	1
---	---	---

↑

5	6	7	199
---	---	---	-----

↑

0	1	1	2
---	---	---	---

↑

6	7	8	10
---	---	---	----

↑

0	0	0
---	---	---

↑

5	10	10	10
---	----	----	----

↑

31	432	530	677
----	-----	-----	-----

↑

```

radixSort(a, n)
{
    int max = getMax(A, n)
    for (pos = 1; max/pos > 0; pos *= 10)
    {
        CountSort(A, n, pos)
    }
}

CountSort(int A[], int n, int pos)
{
    int Count[10] = {0};
    for (i = 0; i < n; i++)
        ++Count[(a[i]/pos)%10];
    for (i = 1; i <= 9; i++)
        Count[i] = Count[i] + Count[i-1];
    for (i = n-1; i >= 0; i--)
        b[Count[(a[i]/pos)%10]] = a[i];
}

for (i = 0; i < n; i++)
    a[i] = b[i];
  
```

$O(n+k)$

$O(d * (n+k))$

$O(n+k)$