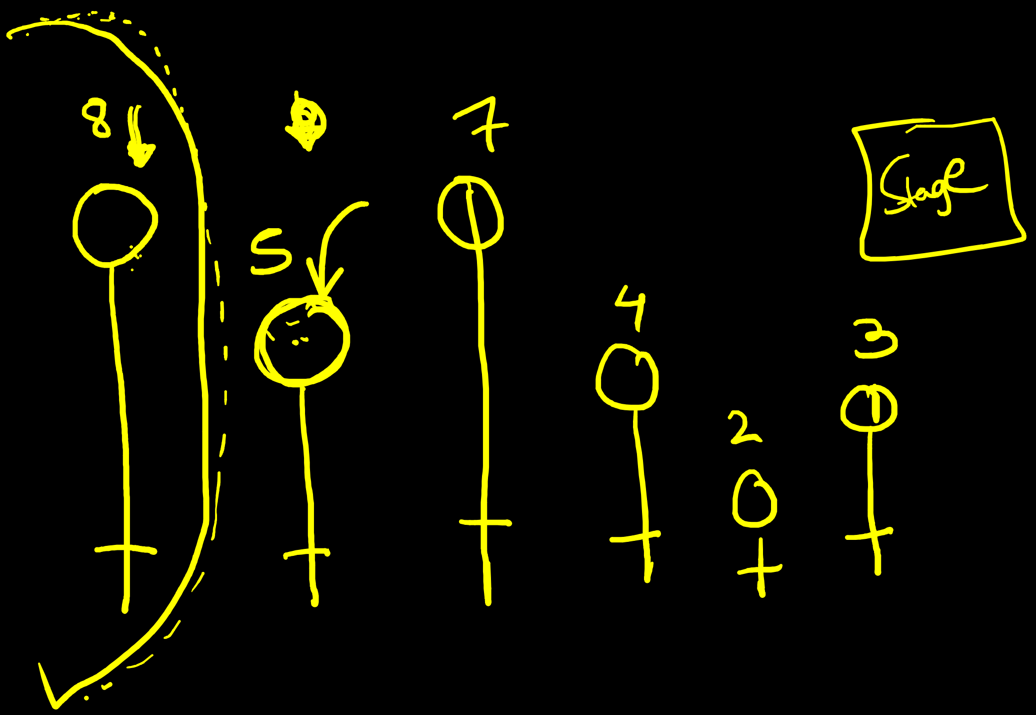
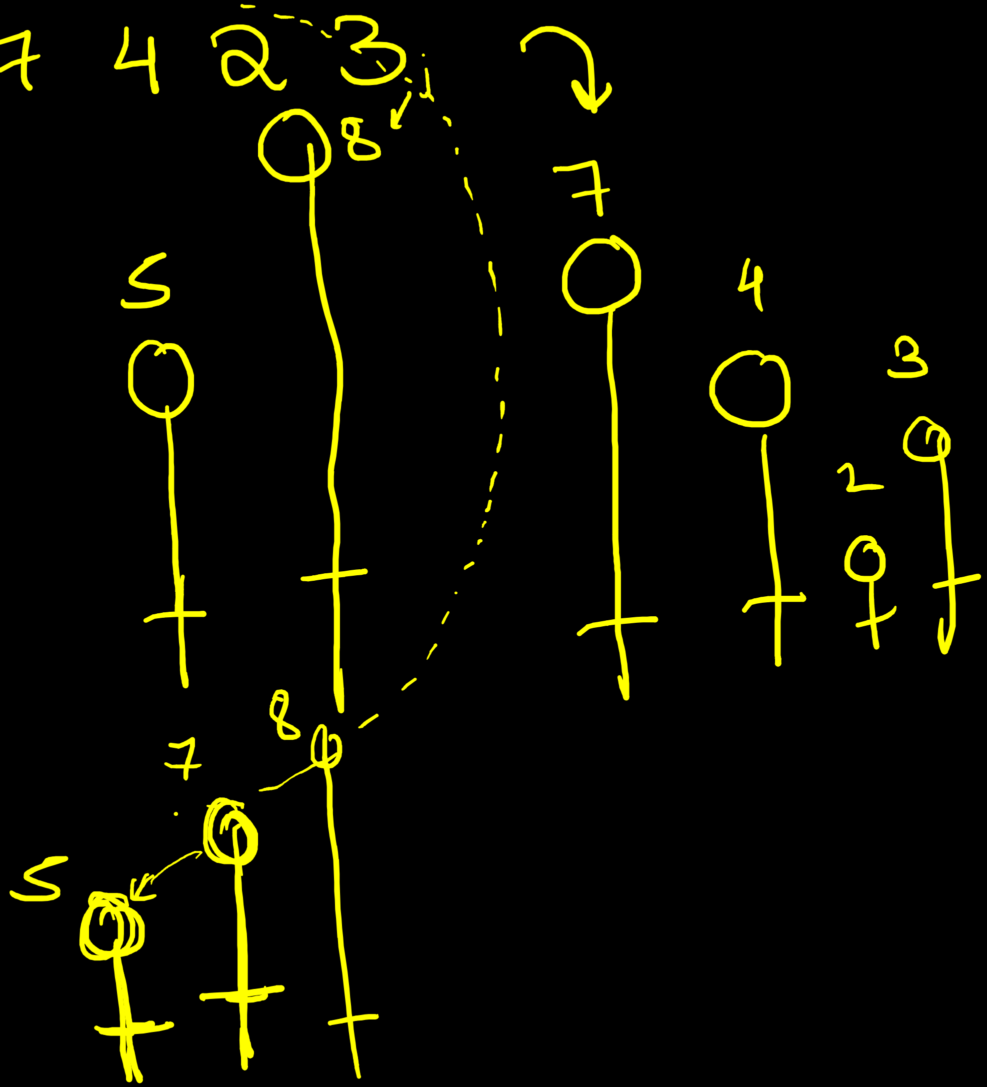
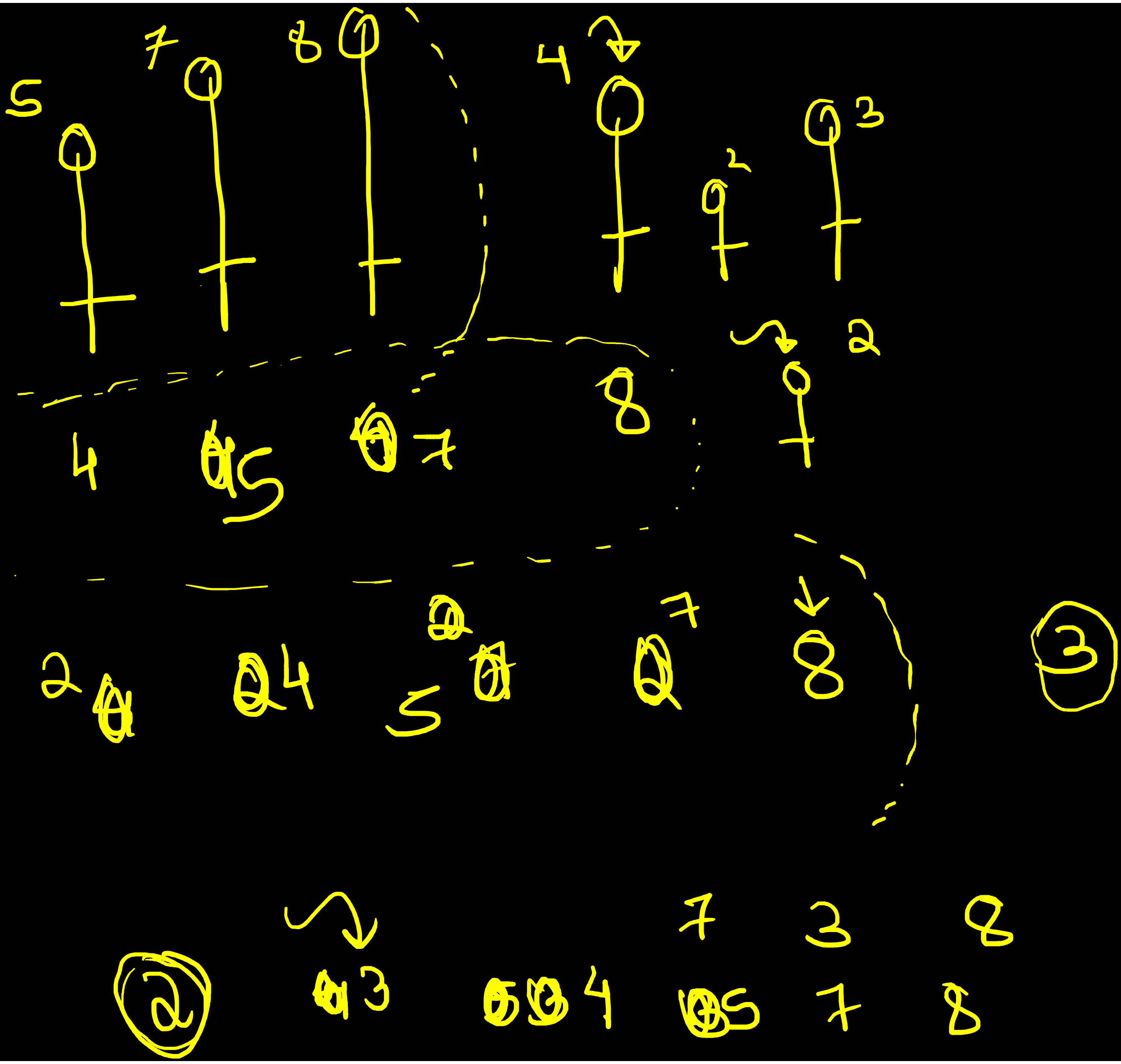
Insertion Sort

8 5 7 4 2 3

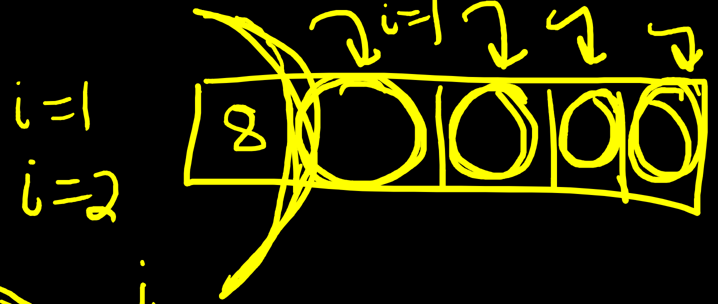


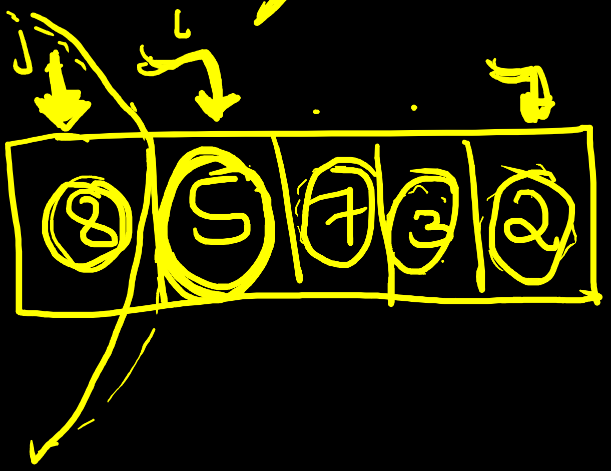



5 7 8 4 2 3
4 5 7 8 2
2 4 5 7 8 3
2 3 4 5 7 8
7 3 8

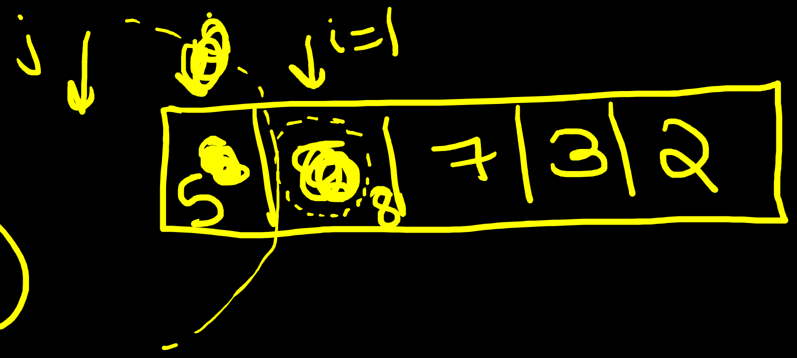
$\textcircled{n} \longrightarrow (n-1)$ elements

```
for( int i=1; i<n; i++)
{ // arr[i]
    int j= i-1;
    while (arr[j] > arr[j+1])
    { swap (arr[j], arr[j+1]);
      j--;
    }
}
```

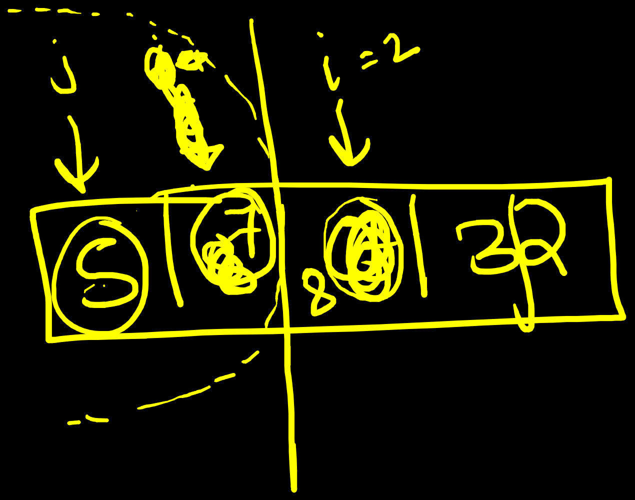


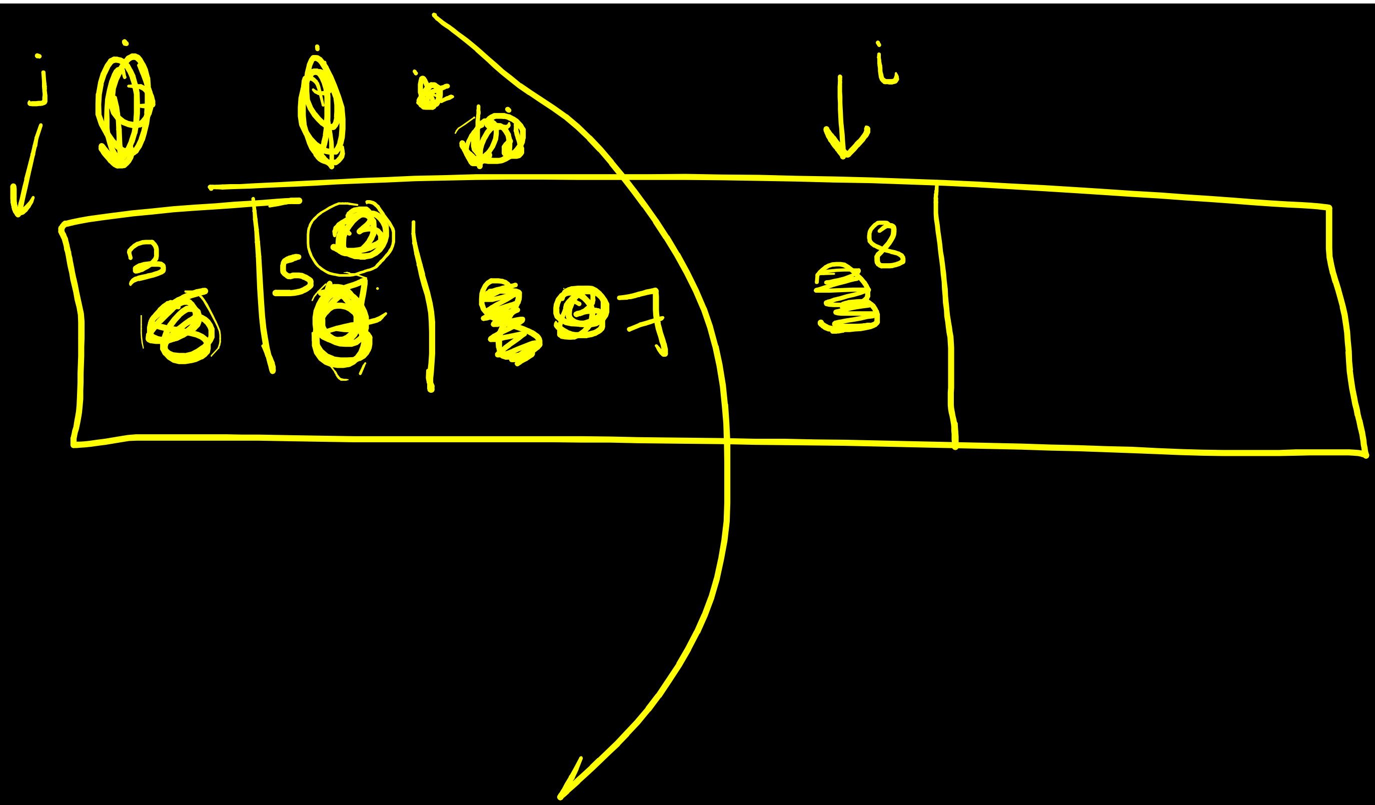


```
for ( int i = 1; i<n; i++)
{  int j = i-1;
   while (j>-1 && arr[j] > arr[j+1])
   {  swap(arr[j], arr[j+1])
        j--;
   }
}
```



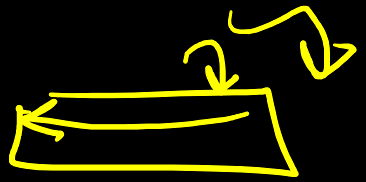
8 > 5





j
i
3
5
8
7
3
8

$1 \longrightarrow 1$ cmp



$2 \longrightarrow 2$

$\vdots$

$(n-1) \longrightarrow (n-1)$

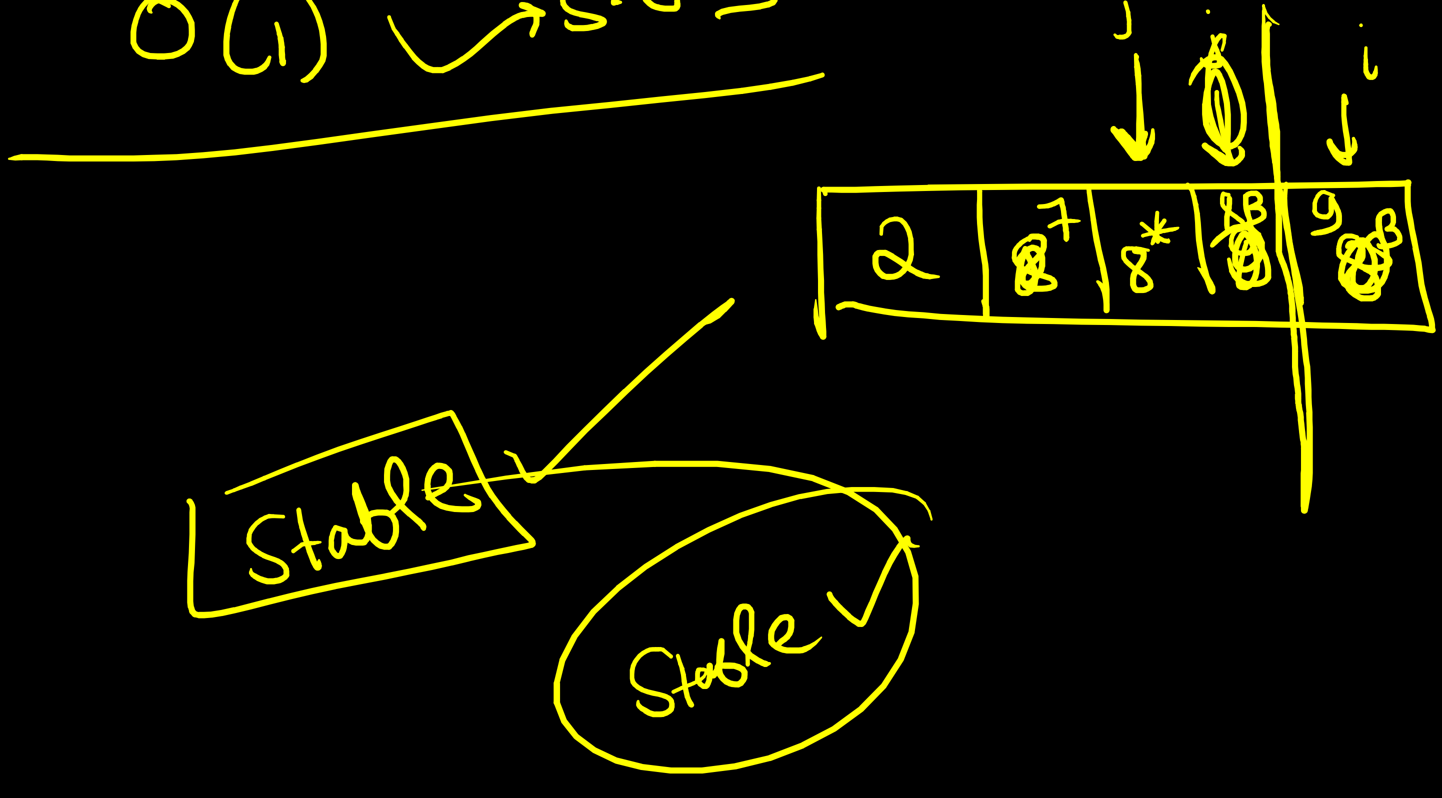
$$\frac{n(n-1)}{2} \quad \boxed{O(n^2)}$$

$$\left.\begin{array}{l} O(n^2) \rightarrow T.C \\ O(1) \rightarrow S.C \end{array}\right\}$$

Stable

Adaptive

arr[j] >

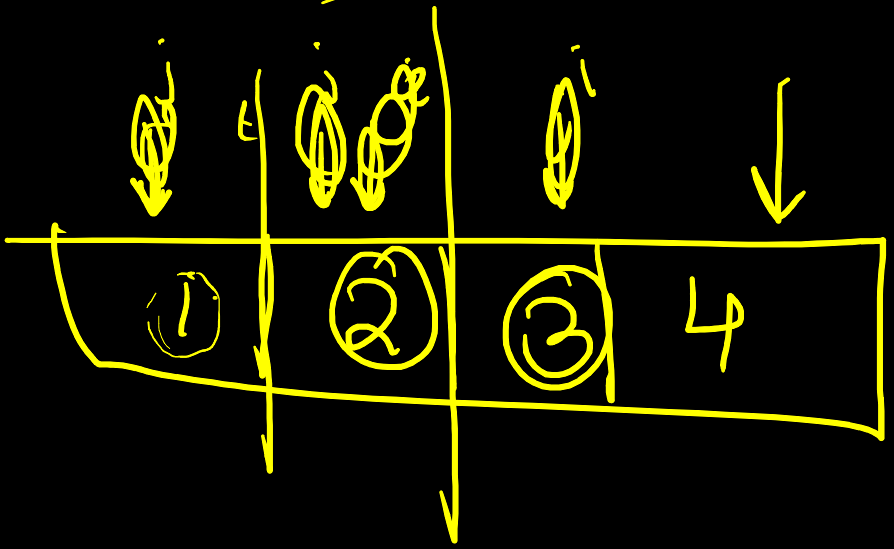


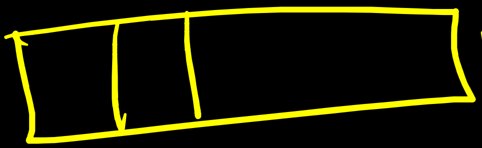
Adaptive

✓ Adaptive

$i = 1$

$\Omega(n)$

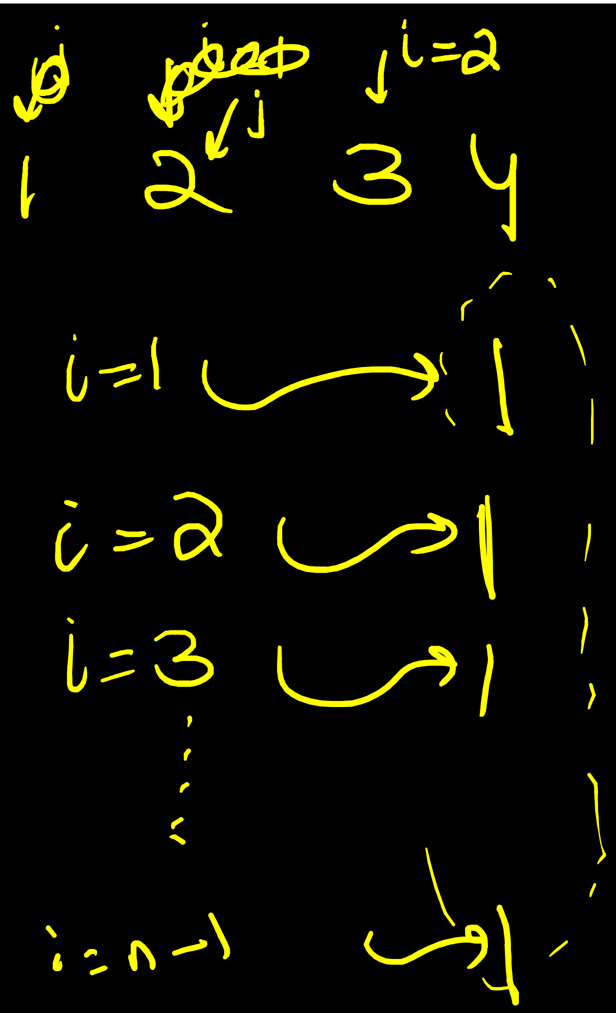


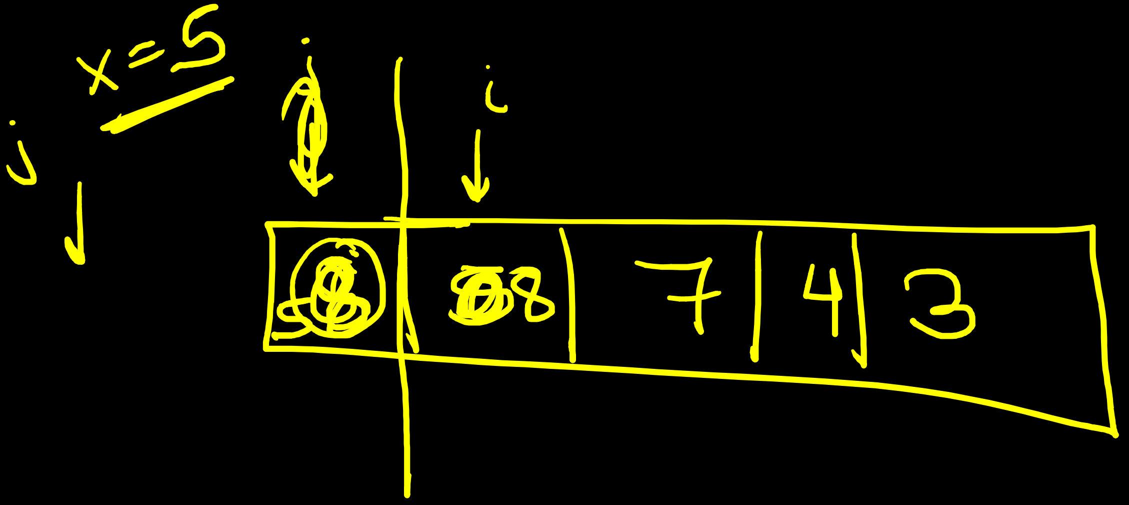


$\Omega(n)$ $\rightsquigarrow$ Adaptive

```
for( int i =1; i<n ;i++)
{
    for(int j=i-1 ; j>=0;j--)
    {
      →if(arr[j] > arr[j+1])
        ( swap)
       else
        {break;}
    }
}
```

$(n-1)$





Insertion

$O(n^2)$

$O(1)$

$\Omega(n)$

Stable ✓

Adaptive ✓

Bubble Sort

1 pass $\longrightarrow$

Insertion Sort

Bubble Sort ✓

Insertion Sort ✓

import java.util.*;

Arrays

Arrays.sort(arr);

O(nlogn)

→ Selection Sort

→ Arrays. sort

→ Switch case

→ Marc Cakewalk.

→ and largest