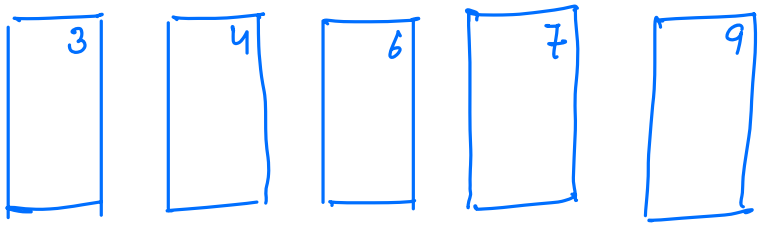
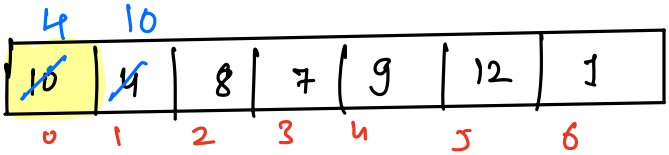
Insertion Sort

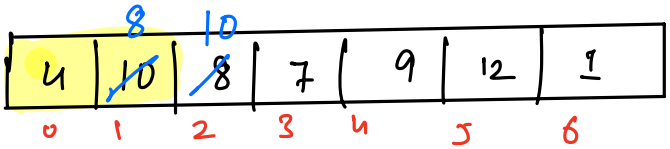
↳ Arrangement of playing cards.



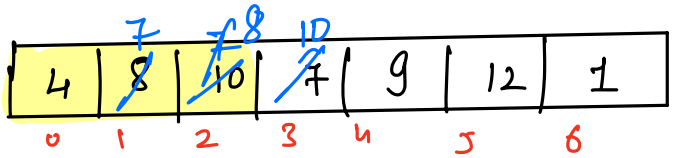
(i = 1) arr →



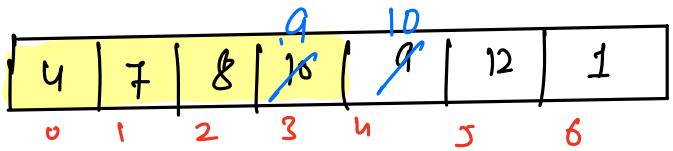
(i = 2)



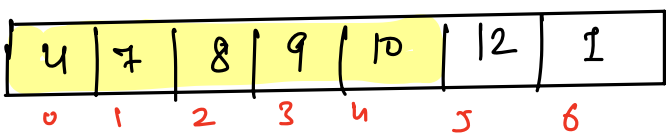
(i = 3)



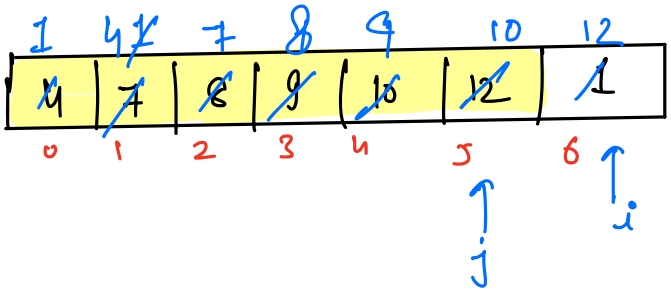
(i = 4)



(i = 5)



(i = 6)



code →

```
for( int i = 1 ; i < N ; i++){
    //find the correct position of arr[i] in sorted array
    for( j = i-1 ; j ≥ 0 ; j--){
        if (arr[j] > arr[j+1]){
            swap (arr[j] with arr[j+1])
        }
        else{
            break;
        }
    }
}
```

T.C → $O(N^2)$
S.C → $O(1)$

Stable sorting ✓

arr → [2, 5, 7, 20, 50, 100] ⇒ N iterations in best case.

→ running stream of integers.

3 → 3
3,1 → 1,3
1,3,7 → 1,3,7
1,3,7,4 → 1,3,4,7
1,3,4,7,2 → 1,2,3,4,7
1,2,3,4,7,5 → 1,2,3,4,5,7

Calling sort() function for every element
⇒ $N^2 \log N$

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

T.C → $O(N^2)$

Partition an array

Given an integer array, re-arrange the elements such that,

all elements $< x$ are on left hand side of array,

all elements $\geq x$ are on right hand side of array.

arr - [7 3 2 5 1 6 4] , X = 3.

Idea 1. Sort the array

→ [1 2 3 4 5 6 7] [T.C → O(N log N), S.C → O(1)]

idea → send smaller elements on left hand side.

arr - [2 1 ~~7~~ -2 5 3 6 4 7 ~~-2~~] , X = 3.

i, j

arr[i] < x

swap(arr[i] with arr[j])

j++

i++

arr[i] ≥ x.

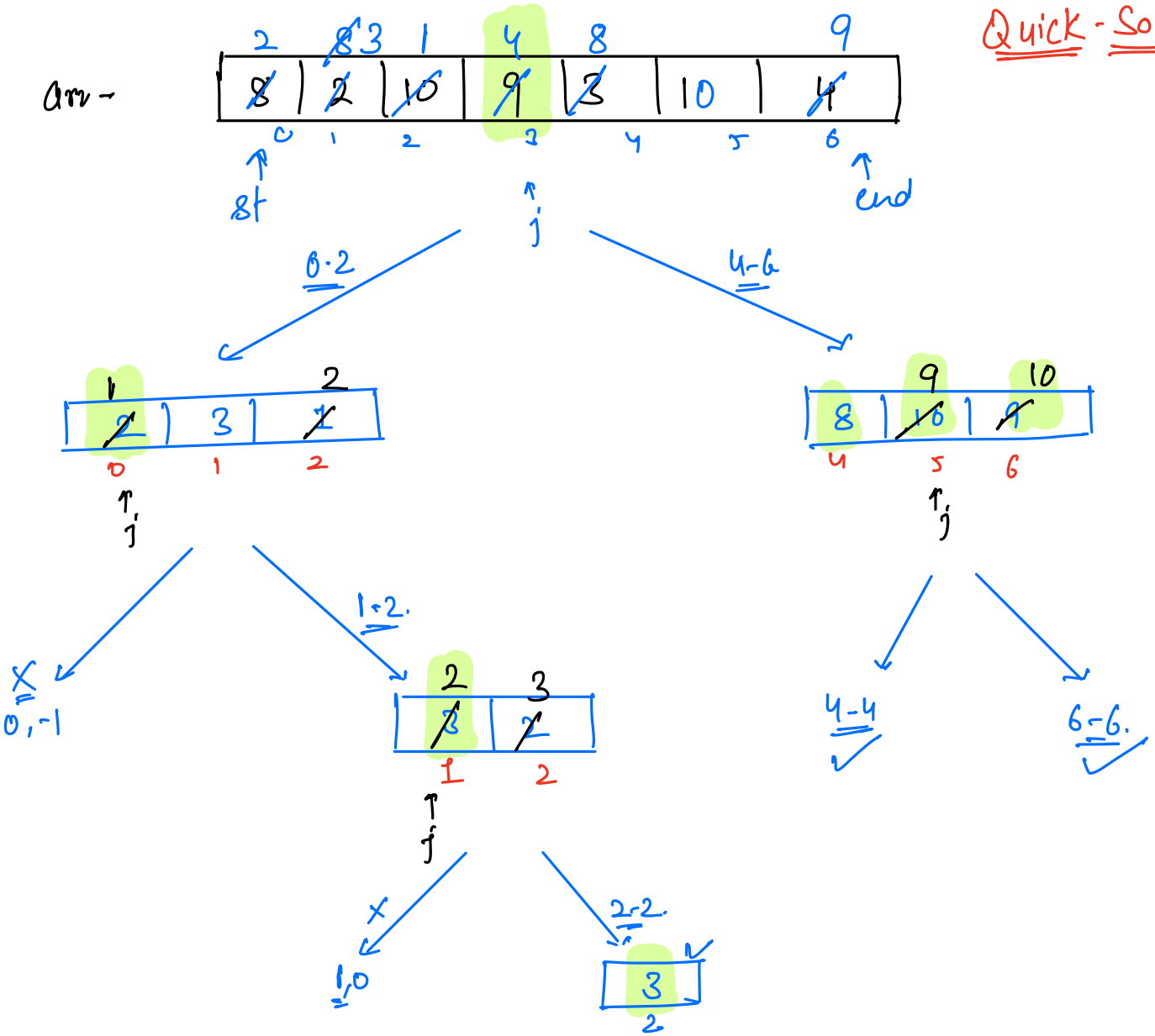
i++

code →

```
j = 0;
for( i = 0; i < N; i++){
    if( arr[i] < x){
        swap( arr[i] with arr[j] )
        j++
    }
}
```

T.C → O(N)
S.C → O(1)

Quick - Sort

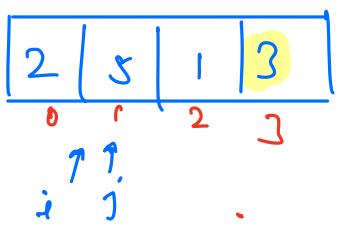

arr -
2 3 1 4 8 9
8 2 10 9 3 10 4
0 1 2 3 4 5 6
st
end
j
0-2
4-6
1 2
2 3 x
0 1 2
j
9 10
8 10 9
4 5 6
j
1-2.
x
0,-1
2
3 2
1 2
j
x
1,0
2-2.
3
2
4-4
6-6.

code →

```
void quickSort( arr, st, end){
    if( si ≥ ei) { return}
    pi = partition( arr, st, end);
    quickSort( arr, st, pi-1);
    quickSort( arr, pi+1, end);
}
```

```
int partition ( int[] arr, int st, int end){
    pivot = arr[end]
    j = st;
    for( i = st ; i ≤ end ; i++){
        if (arr[i] < pivot){
            swap( arr[i] with arr[j]);
            j++;
        }
    }
    swap( arr[j] with arr[end])   // to send pivot to its correct position
    return j;
}
```

O(N)



st → 0
end → 3

$$T(N) = 2T(N/2) + N$$

$\Rightarrow$ T.C $\rightarrow$ $O(N \log N)$
S.C $\rightarrow$ $O(\log N)$

Worst case. [when smallest / largest element is pivot]

$$T(N) = T(N-1) + N$$

$$T(N-1) = T(N-2) + (N-1)$$

$$T(N) = T(N-2) + N + (N-1)$$

$$T(N-2) = T(N-3) + N-2$$

$$T(N) = T(N-3) + N + (N-1) + (N-2)$$

$$T(N) = T(N-4) + N + (N-1) + (N-2) + (N-3)$$

$$\vdots$$

$$T(N) = T(0) + N + (N-1) + (N-2) + (N-3) + \cdots + 3 + 2 + 1$$

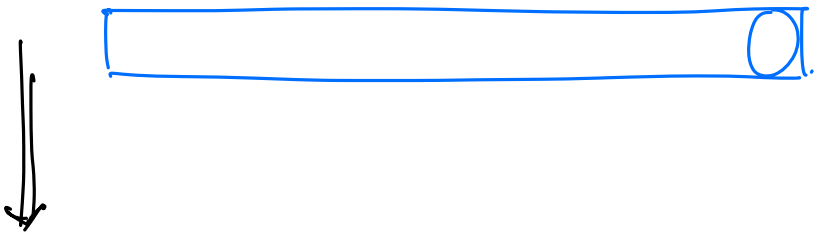
$$T(N) = \frac{N(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}$$

T.C $\rightarrow$ $O(N^2)$
S.C $\rightarrow$ $O(N)$

∴ T.C of QuickSort varies from NlogN to N^2

S.C of QuickSort varies from logN to N.

Randomised Quick Sort



[Source code] → Java Arrays.sort source code.

→ C# | C++ | Python

1. Given arr[N]. Make all elements distinct.

To do so, in one step you can increase any number by one.

Find the minimum no. of steps to make all array elements distinct.

arr → [2, 1, 3] ans = 1

arr → [1, 1→2, 2→3] ans = 2.

arr → [1, 1→2, 2→3, 3→4] ans = 3.

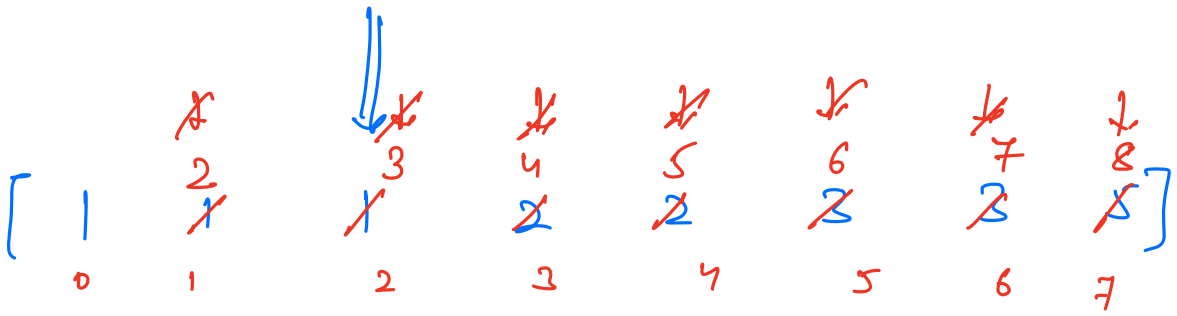
arr → [4 3 1 2 5] ans = 0

⇓

1 2 3 4 5

idea → Sort the array and check for every consecutive pair if (arr[i] > arr[i-1]) or not.

arr → [5 3 1 2 1 1 2 3]



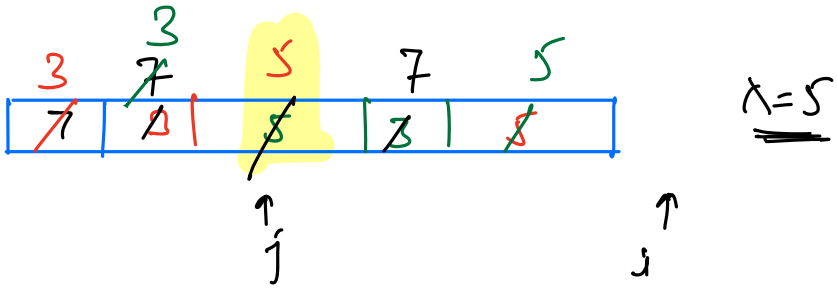
steps = 0 + 1 + 2 + 2 + 3 + 3 + 4 + 3 = 18.

code.

```
sort(arr);
steps = 0;
for( i = 1; i < N; i++){
    if (arr[i] <= arr[i-1]){
        steps += ( arr[i-1]+1 - arr[i]);
        arr[i] = arr[i-1]+1
    }
}
return steps;
```

TC → O(N log N)
SC → O(1)

Partitioning.



Quick → unstable sorting