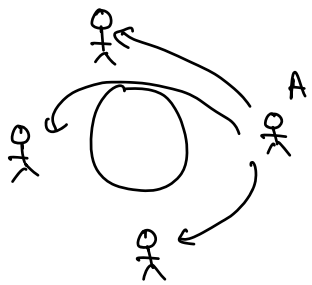
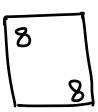
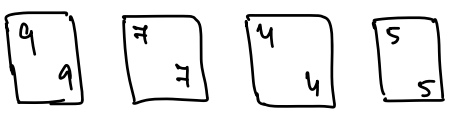
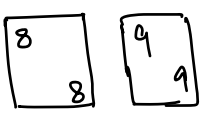
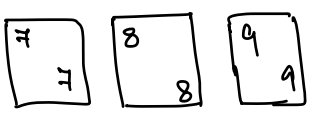

A

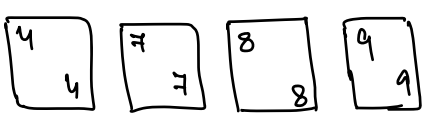

Arun



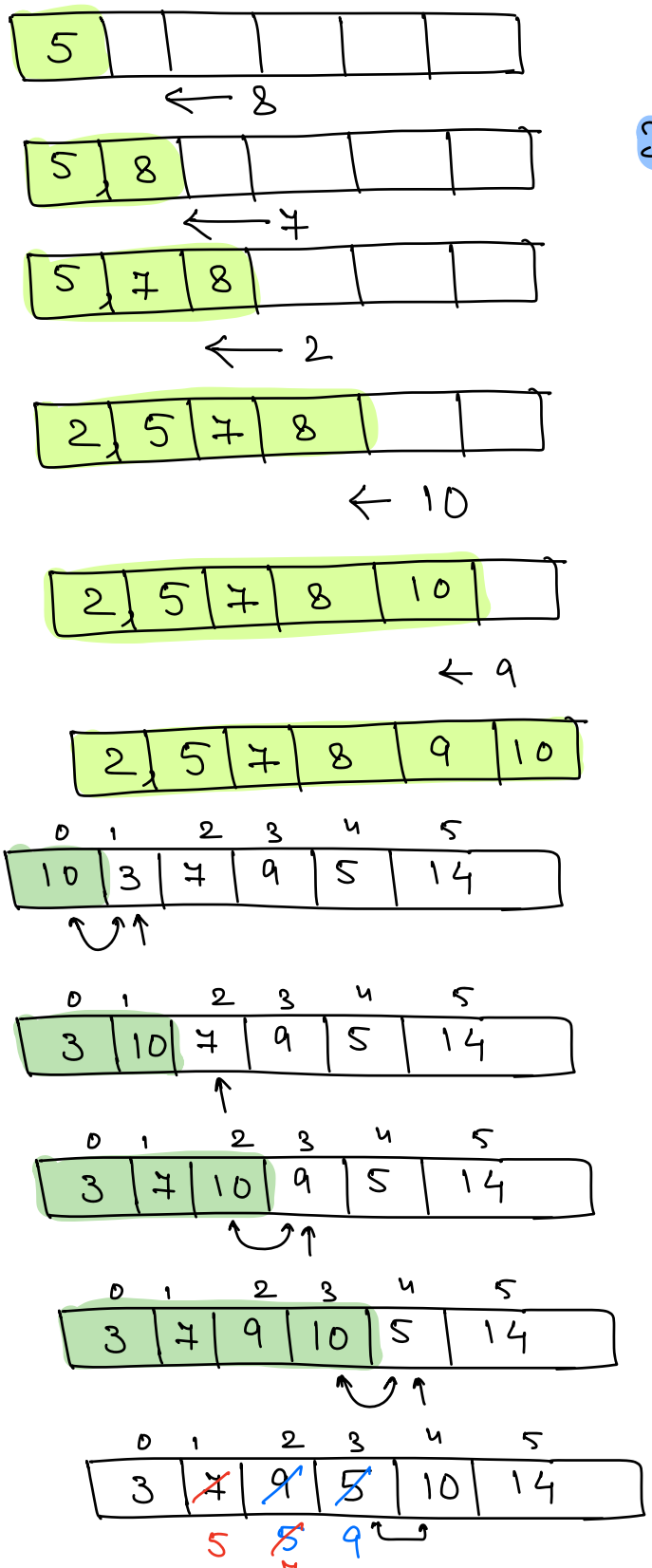








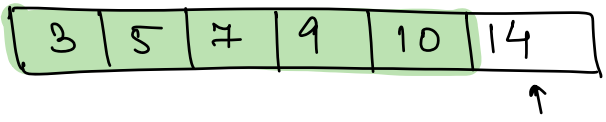
Stream of integers.

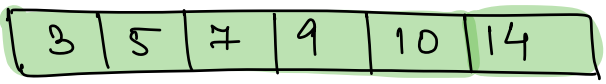


Insert the element at its correct pos^

Insertion Sort

Ex





Sorted

```
void insertionSort(int A[], int N) {
    for(i=1; i<N; i++){
        j = i-1;
        index = i;

        while(j>=0) {
            if(A[index] < A[j]){
                swap(A[j], A[index]);
                index = j
                j--;
            }
            else{
                break;
            }
        }
    }
}
```

TC: $O(N^2)$:

Stable ? ✓

Inplace ? ✓ SC: O(1)

No. of swaps (W.C) ?

1 2 3 4 7 8 N = 6

1 + 2 + 3 + 4 + 5

No. of swaps = 1 + 2 + 3 + 4 + ------ + N-1

$$= \frac{N(N-1)}{2}$$

Best case T.C

1, 2, 3, 4, 5 ⇒ O(N) No swaps

TC: O(N)

If the Array is almost sorted then we can use Insertion sort.

1, 2, 3, 5, 4

1 swaps.

Q: Given an Array of size N, rearrange the array s.t

1. All elements <= A[0] ⇒ Go to left
2. All elements > A[0] ⇒ Go to right

{10, 3, 8, 15, 6, 12, 2, 18, 7, 15, 14}

Partition

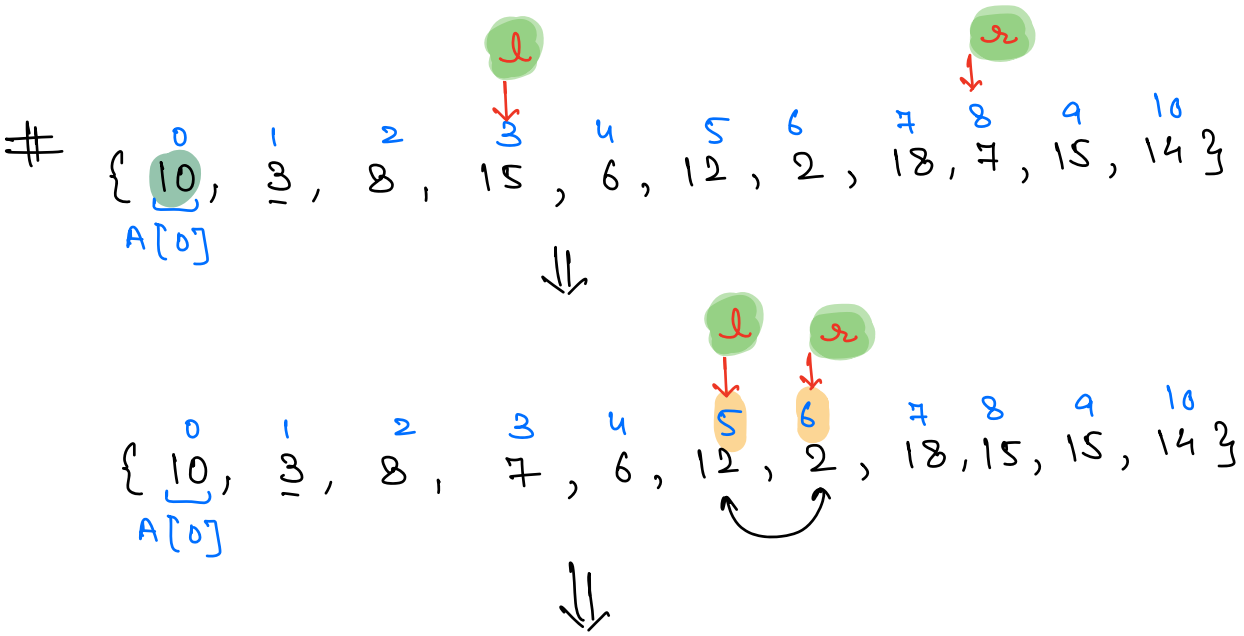
{3, 8, 6, 2, 7, 10, 15, 12, 18, 15, 14}

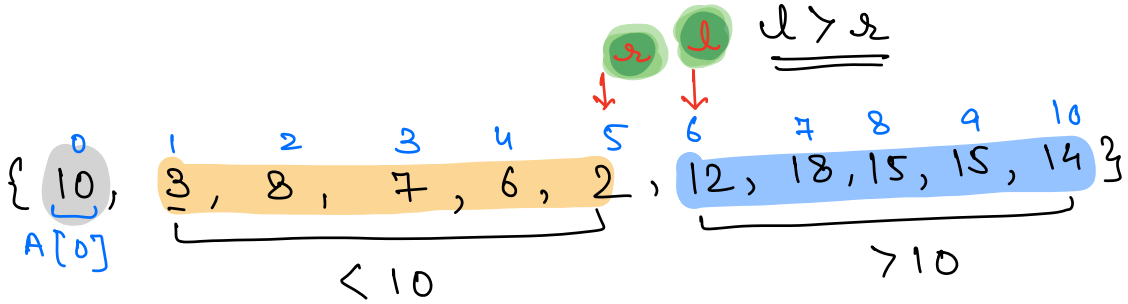
<=10 >10

TC: O(N)

SC: O(1)

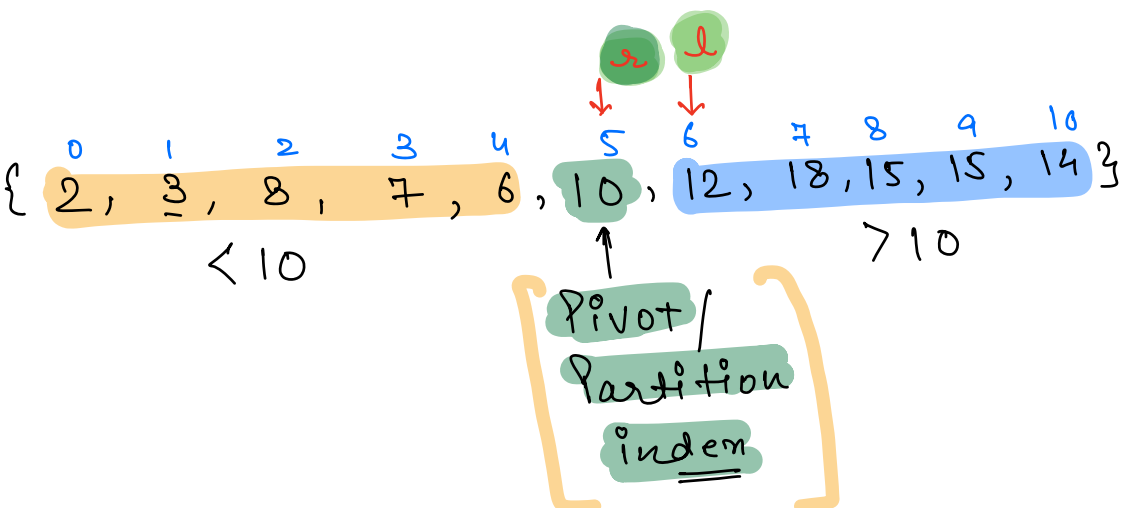
#





swap(A[0], A[l]) ✗

swap(A[0], A[r])
swap(A[0], A[l-1]) } ✓



```
void partition ( int A[], int N) {
    l = 1
    r = N-1 ;
    while ( l <= r ) {
        if( A[l] <= A[0])
            l++;
        else if ( A[r] > A[0])
            r--;
        else {
            swap( A[l], A[r] );
            l++;
            r--;
        }
    }
    swap ( A[0], A[r] );
}
```

Pivot index => r

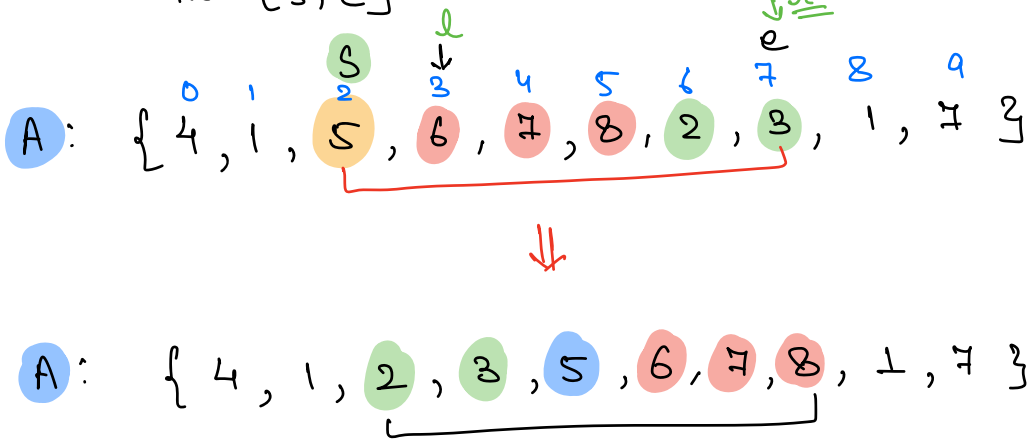
TC : O(N)

SC : O(1)

Q: Given an Array of size N, rearrange the subarray from s to e s.t

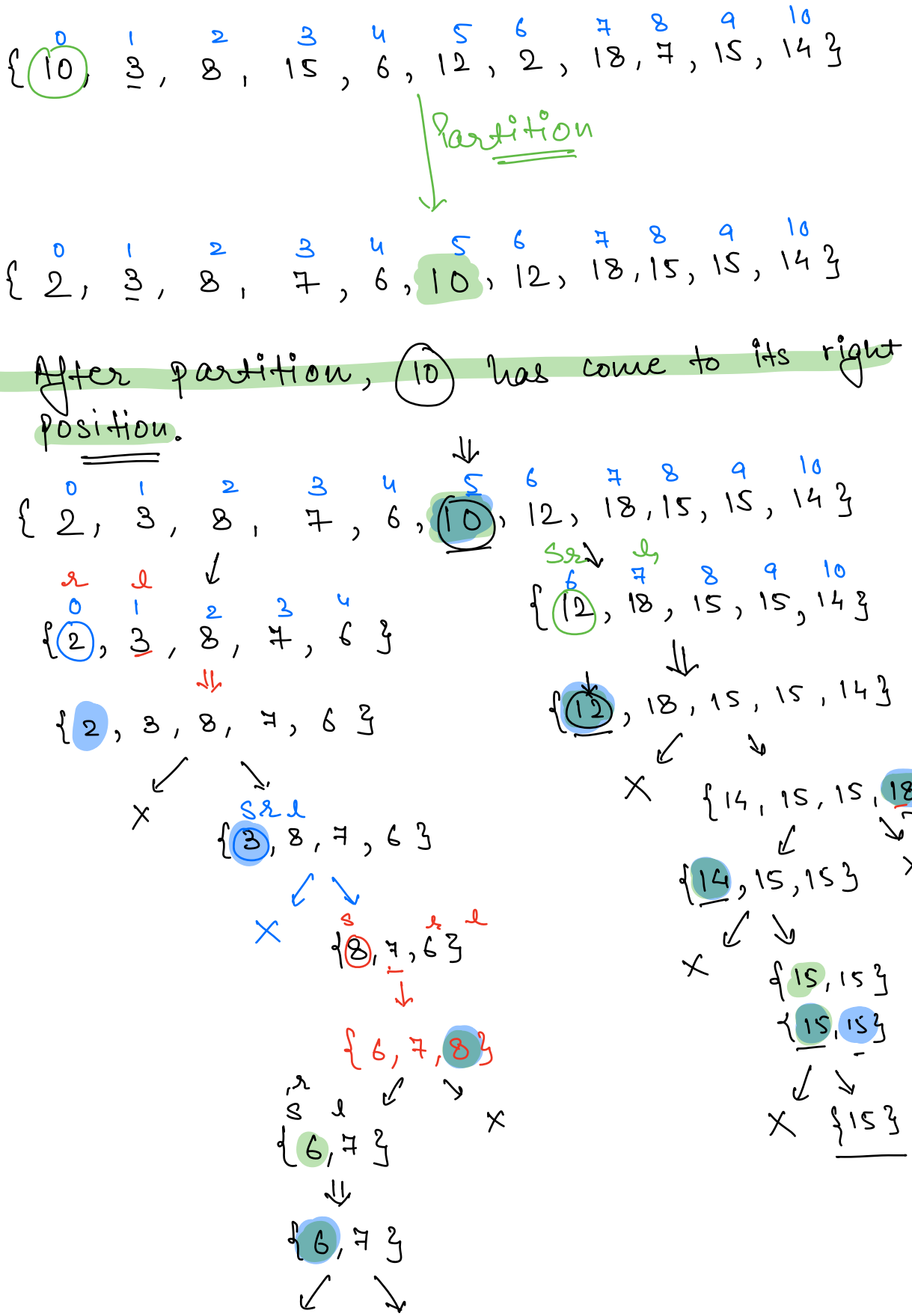
① All elements <= A[s] ⇒ Go to left in [s, e]

② All elements > A[s] ⇒ Go to right in [s, e]



```
int partition ( int A[], int s, int e)
    l = s+1
    r = e ;
    while ( l <= r ) {
        if( A[l] <= A[s])
            l++;
        else if ( A[r] > A[s])
            r--;
        else {
            swap( A[l], A[r] );
            l++;
            r--;
        }
    }
    swap ( A[s], A[r] );
    return (r) ;
}
```

↳ Pivot index


{10, 3, 8, 15, 6, 12, 2, 18, 7, 15, 14}
Partition
{2, 3, 8, 7, 6, 10, 12, 18, 15, 15, 14}
After partition, 10 has come to its right position.
{2, 3, 8, 7, 6, 10, 12, 18, 15, 15, 14}
{2, 3, 8, 7, 6}
{2, 3, 8, 7, 6}
{3, 8, 7, 6}
{8, 7, 6}
{6, 7, 8}
{6, 7}
{6, 7}
{12, 18, 15, 15, 14}
{12, 18, 15, 15, 14}
{14, 15, 15, 18}
{14, 15, 15}
{15, 15}
{15, 15}
{15}

× {7}

{2, 3, 6, 7, 8, 10, 12, 14, 15, 15, 18}

⇒ Sorted.

Quick Sort

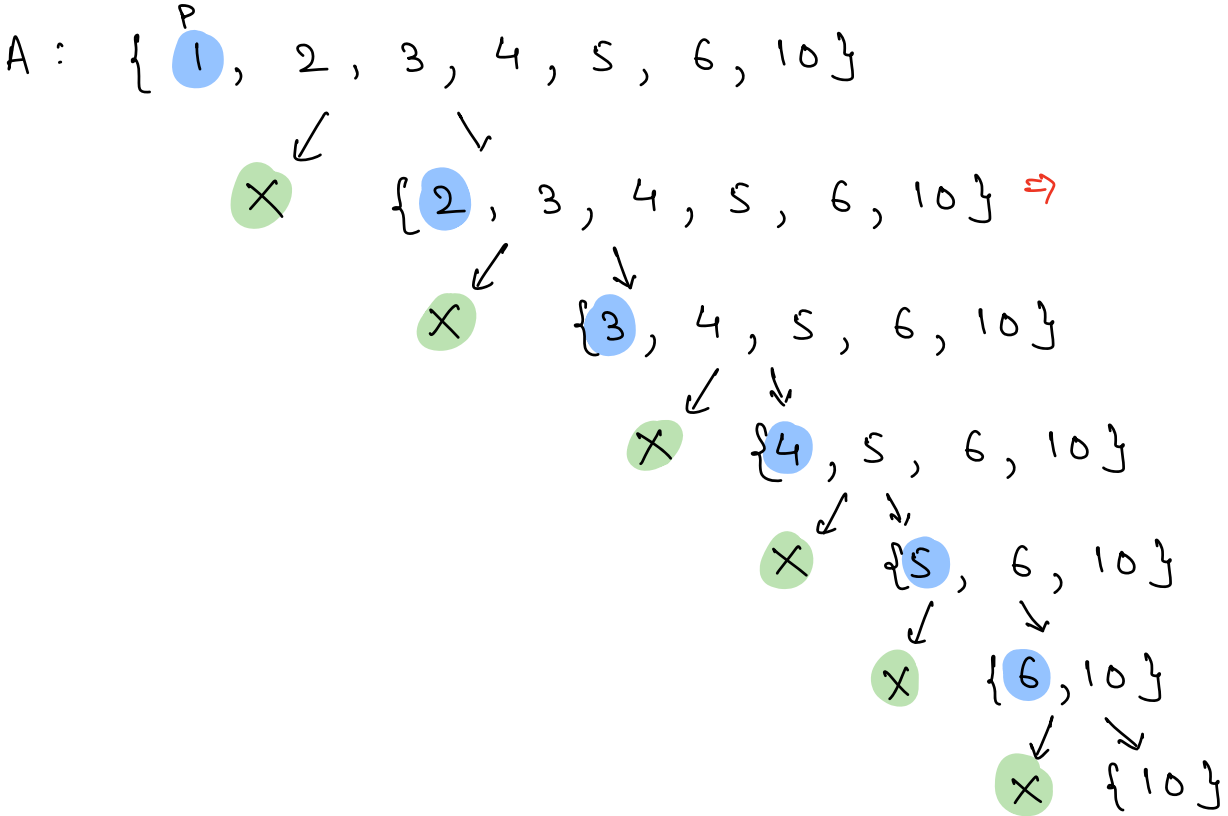
* Assumption : quicksort(A[], s, e) will sort the subarray from s to e.

```
void quicksort(A[], s, e){
    if(s>=e)
        return;
    int pivot = partition(A, s, e);
    quicksort(A, s, pivot-1);
    quicksort(A, pivot+1, e);
}
```

Time Complexity

⇒ Best case :-

Ex If the Array is already sorted.



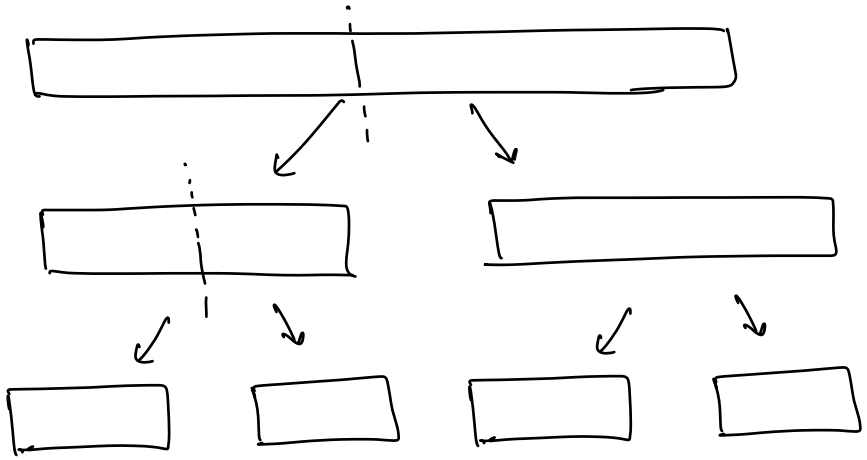
# of iterations :-

$= N + (N-1) + (N-2) + \cdots\cdots 1$

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

TC : $O(N^2)$: Worst Case

Best Case : Pivot partitions the array into exactly half every time.



TC: $O(N \log N)$

$$\boxed{T(N) = 2T(N/2) + O(N)}$$ Recurrence relation

A: {1, 2, 3, 4, 5, 6, 10}

$\frac{1}{N} \times \frac{1}{N-1} \times \frac{1}{N-2} \cdots\cdots \Rightarrow 0$ } Practically impossible.

Probability of getting $O(N^2)$ case if we are picking the random index everytime is practically zero.

$\Rightarrow$ Randomized Quick Sort

Code

```
randomIndex = rand(——) ⇒ O(1)
Swap( A[0], A[randomIndex]);
P = partition( A, S, e);
QS( A, S, P-1);
QS (A, P+1, e);
```

Avg case :- O(NlogN)

TC of Q.S

Best case : O(NlogN)
Worst case : $O(N^2)$
Avg case : O(NlogN)

* See the implementation of library sort method.

Is Q.S Stable ? NO

S.C Inplace Sorting

Worst case : O(N) { Skewed Tree } } Call Stack

Best case : O(logN) }

* Are we creating any extra space explicitly ?

⇒ NO

Count Sort

a c b a d e

a a b c d e

If the input in range specific then we can apply the Count Sort.