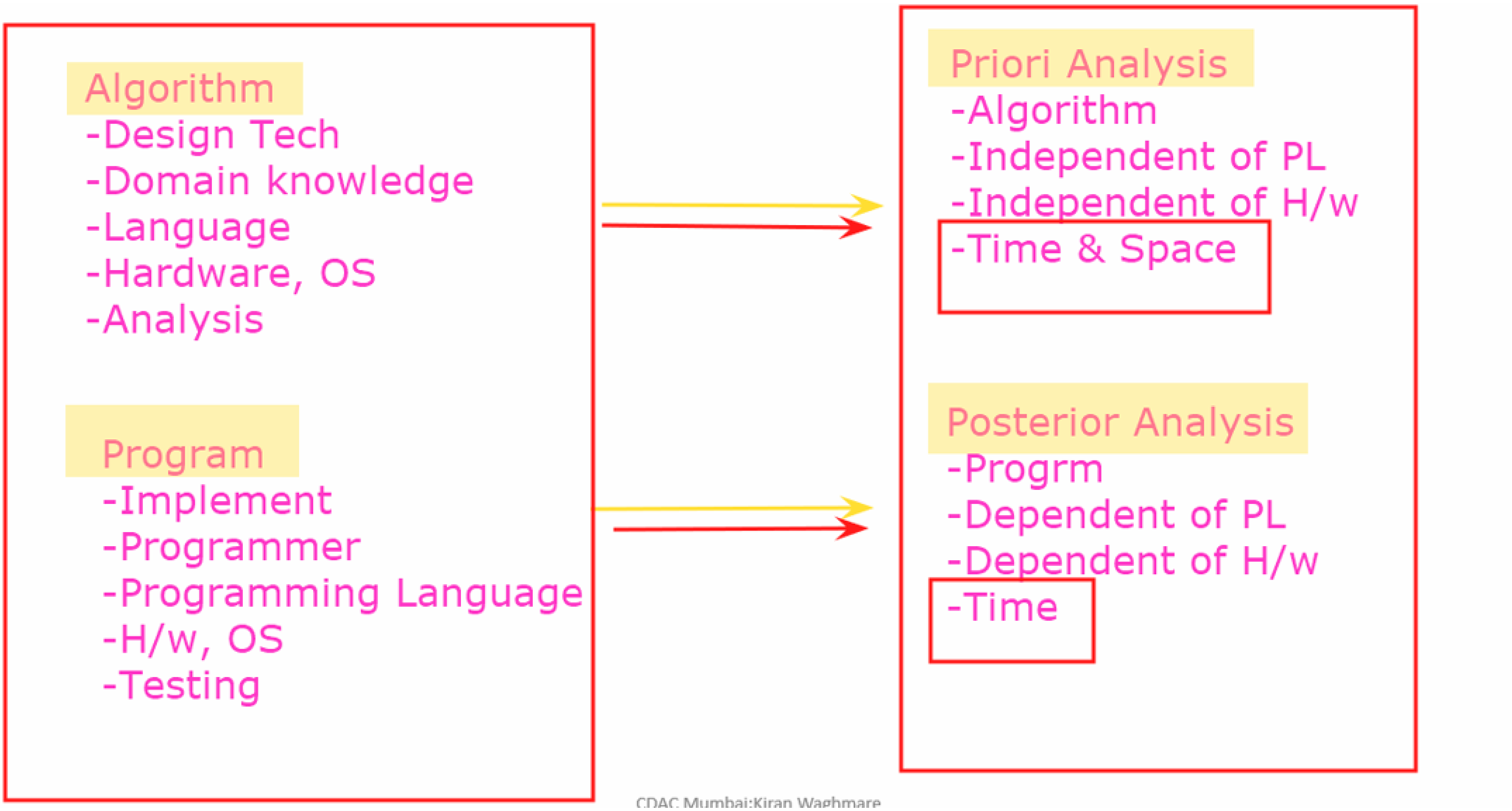
Algorithms & Data Structure

Kiran Waghmare



Algorithm

-Design Tech

-Domain knowledge

-Language

-Hardware, OS

-Analysis

Program

-Implement

-Programmer

-Programming Language

-H/w, OS

-Testing

Priori Analysis

-Algorithm

-Independent of PL

-Independent of H/w

-Time & Space

Posterior Analysis

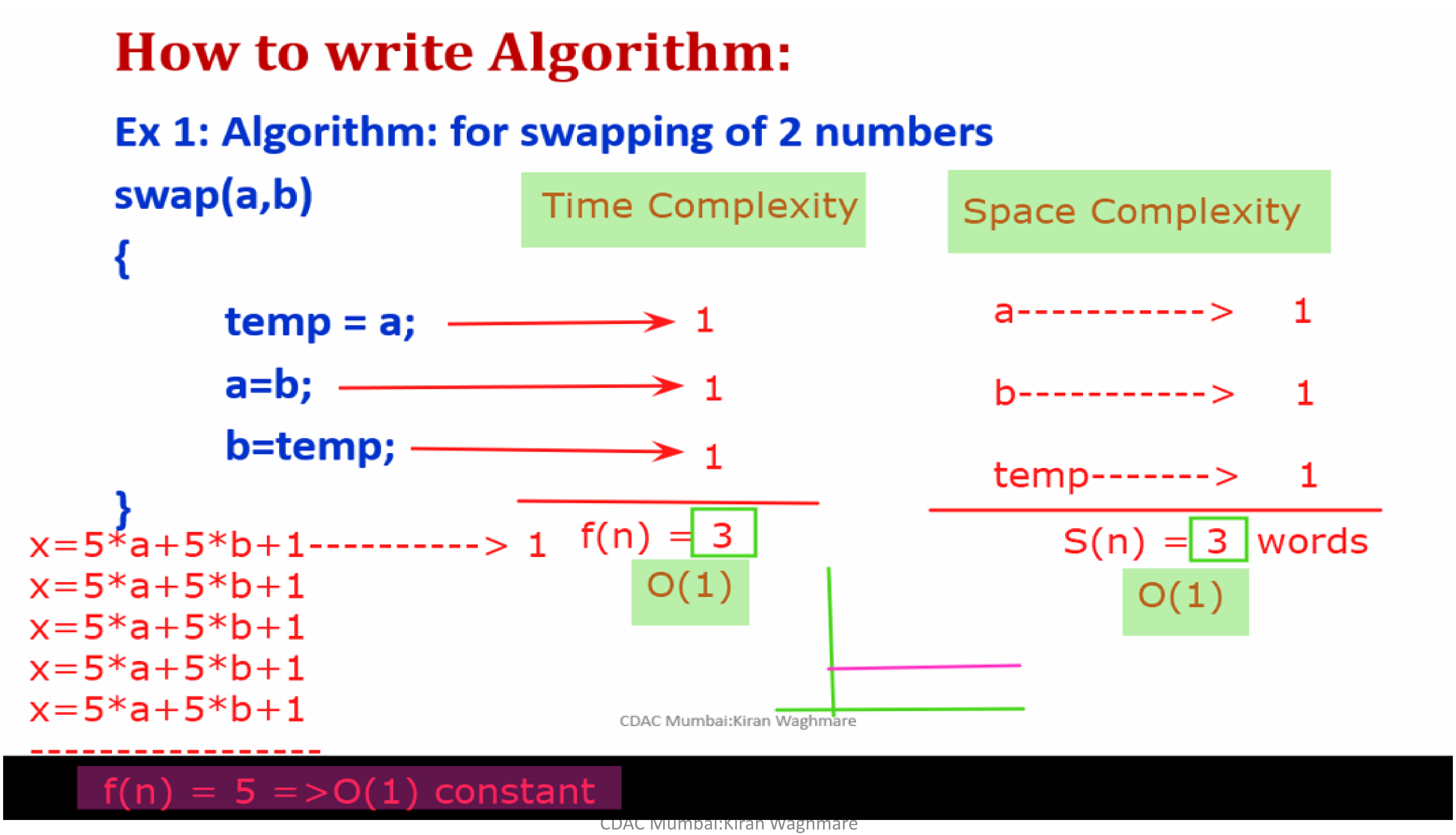
-Progrm

-Dependent of PL

-Dependent of H/w

-Time

CDAC Mumbai:Kiran Waghmare



How to write Algorithm:

Ex 1: Algorithm: for swapping of 2 numbers

```
swap(a,b)
{
        temp = a;    ------>  1
        a=b;         ------>  1
        b=temp;      ------>  1
}
```

Time Complexity

$f(n) = \boxed{3}$

O(1)

Space Complexity

a----------> 1

b----------> 1

temp-------> 1

$S(n) = \boxed{3}$ words

O(1)

x=5*a+5*b+1----------> 1
x=5*a+5*b+1
x=5*a+5*b+1
x=5*a+5*b+1
x=5*a+5*b+1

f(n) = 5 =>O(1) constant

Ex 2:Algorithm:sum of array elements

A= 5 7 9 3 1

0 1 2 3 4

n=5 s=0

execution= n+1

```
sum(A,n)
{
    s=0;                  → 1
    for(i=0;i<n;i++)      → n+1
    {
        s=s+A[i];         → n
    }
    return s;             → 1
}
```

Time Complexity

1

n+1

n

1

$f(n)=2n+3$

$\boxed{O(n)}$

Space Complexity

s---->1

i----->1

n----->1

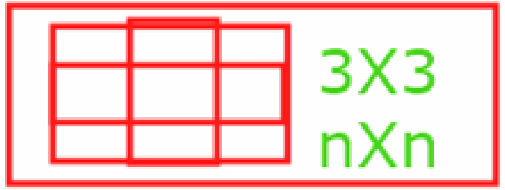
A----->n

$s(n)=n+3$

$\boxed{O(n)}$

Frequency Count Method:

Ex 3:Algorithm to add 2D array elements



```
Add(A,B,n)
{
        for(i=0;i<n;i++)
        {
        for(j=0;j<n;j++)
        {
        C[i,j]=A[i,j]+B[i,j];
        }
        }
}
```

Time Complexity

n+1

n(n+1) =n^2+n

n(n) =n^2

f(n)=2n^2+2n+1

O(n^2)

Space Complexity

A--->n^2

B--->n^2

C---->n^2

n----->1

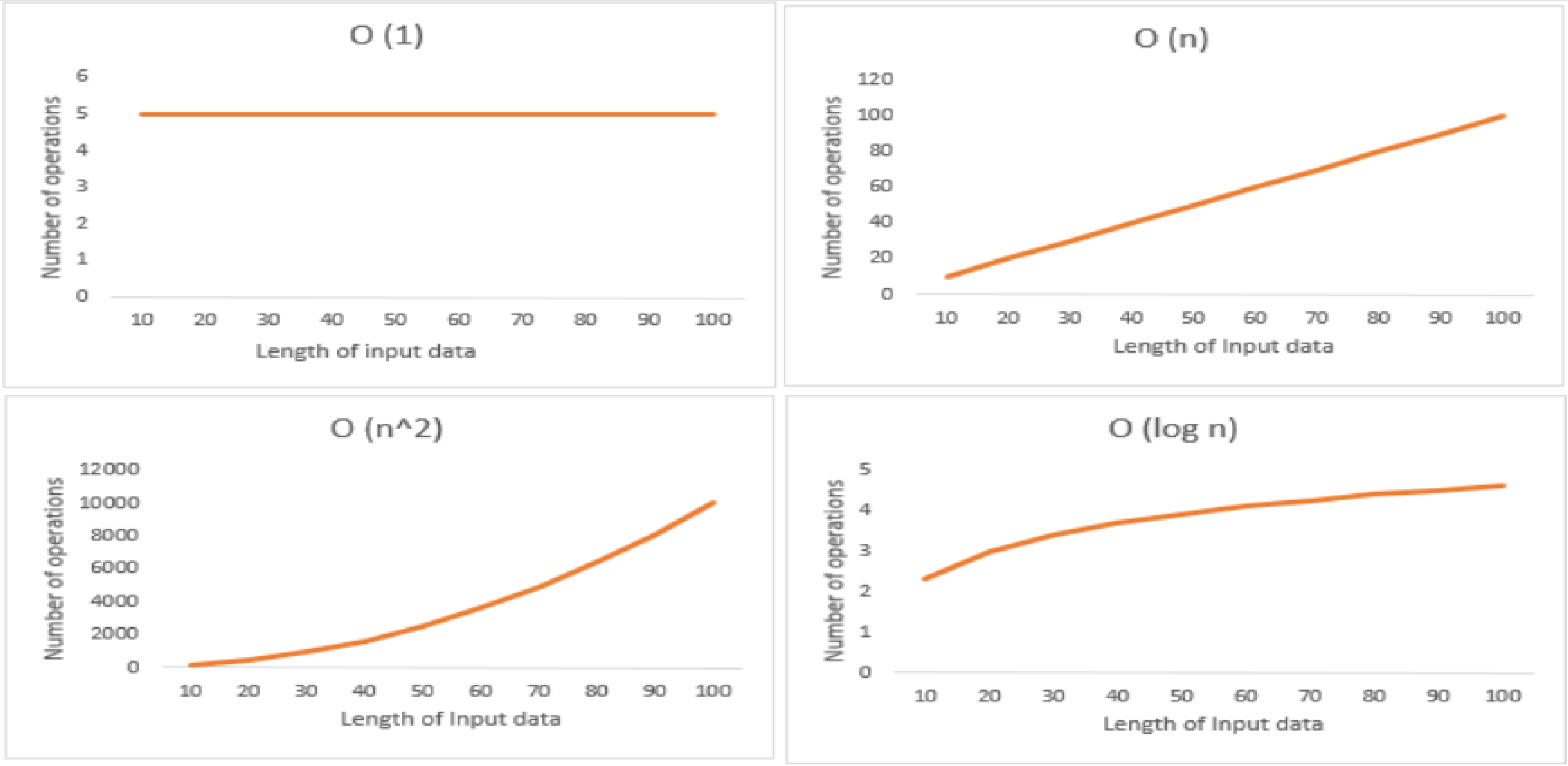
i------>1

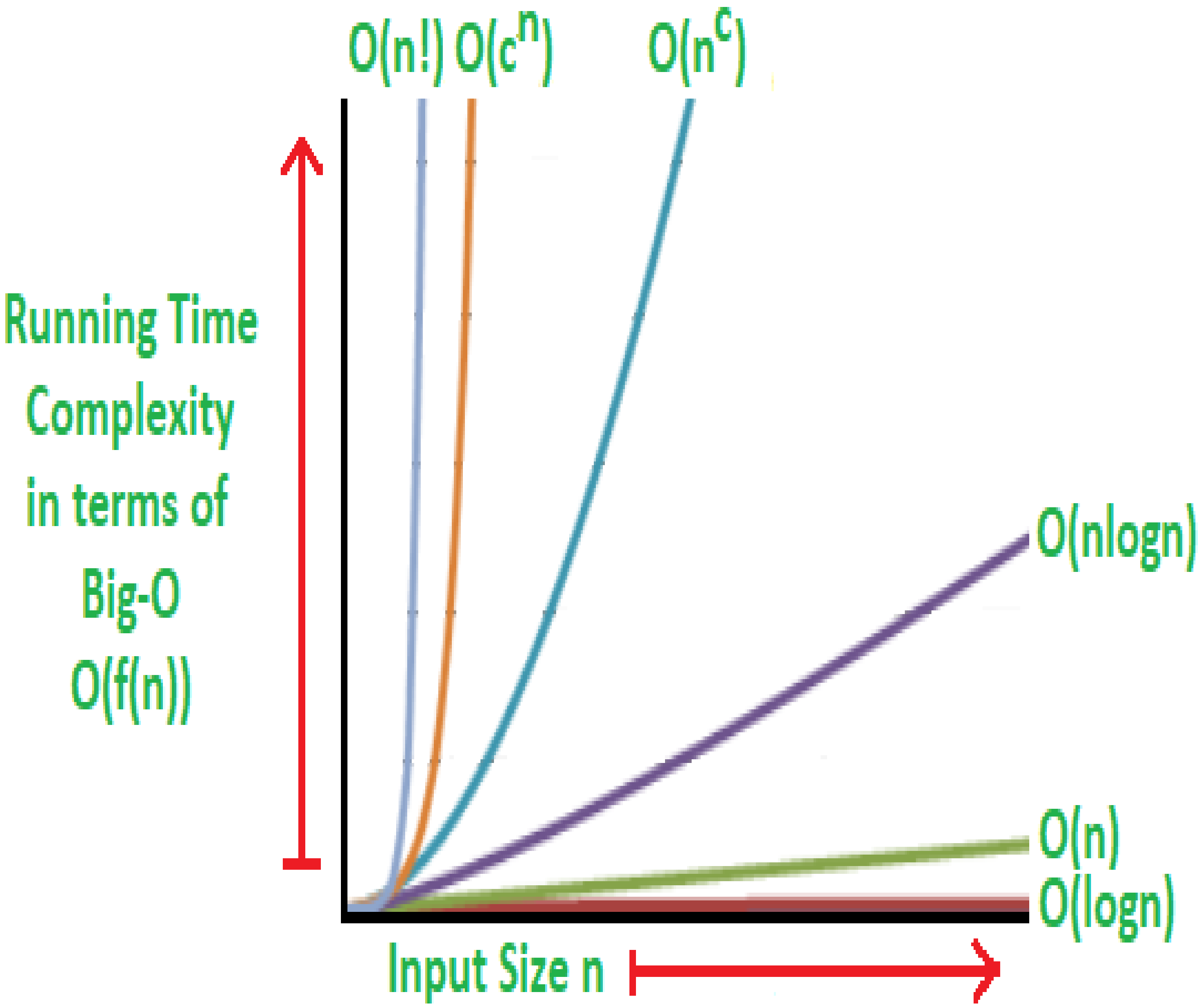
j------>1

S(n)=3n^2+3

O(n^2)

The order of growth for all time complexities are indicated in the graph below:





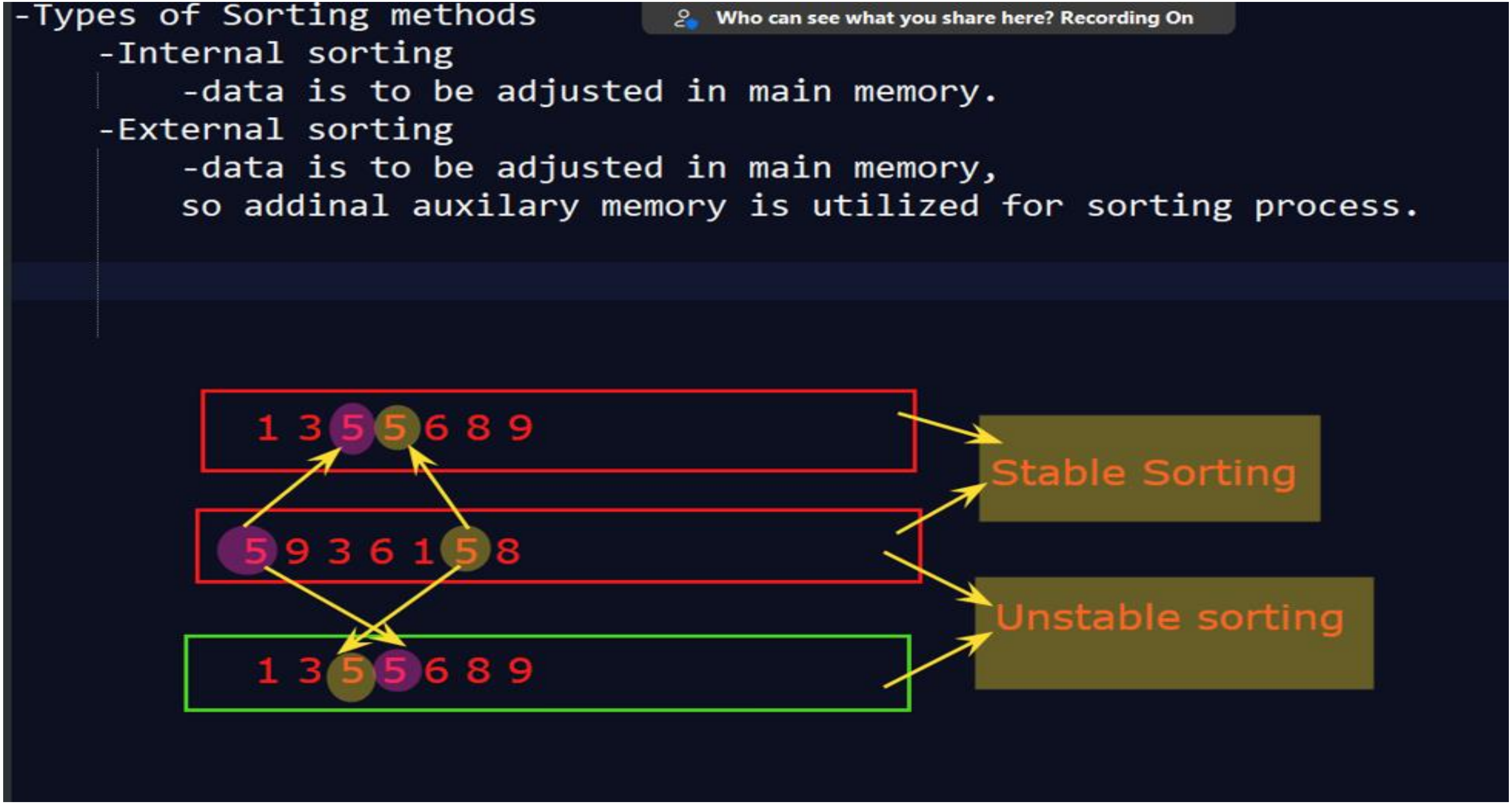
$O(n!), O(c^n), O(n^c)$ - Worst

$O(nlogn)$ - Bad

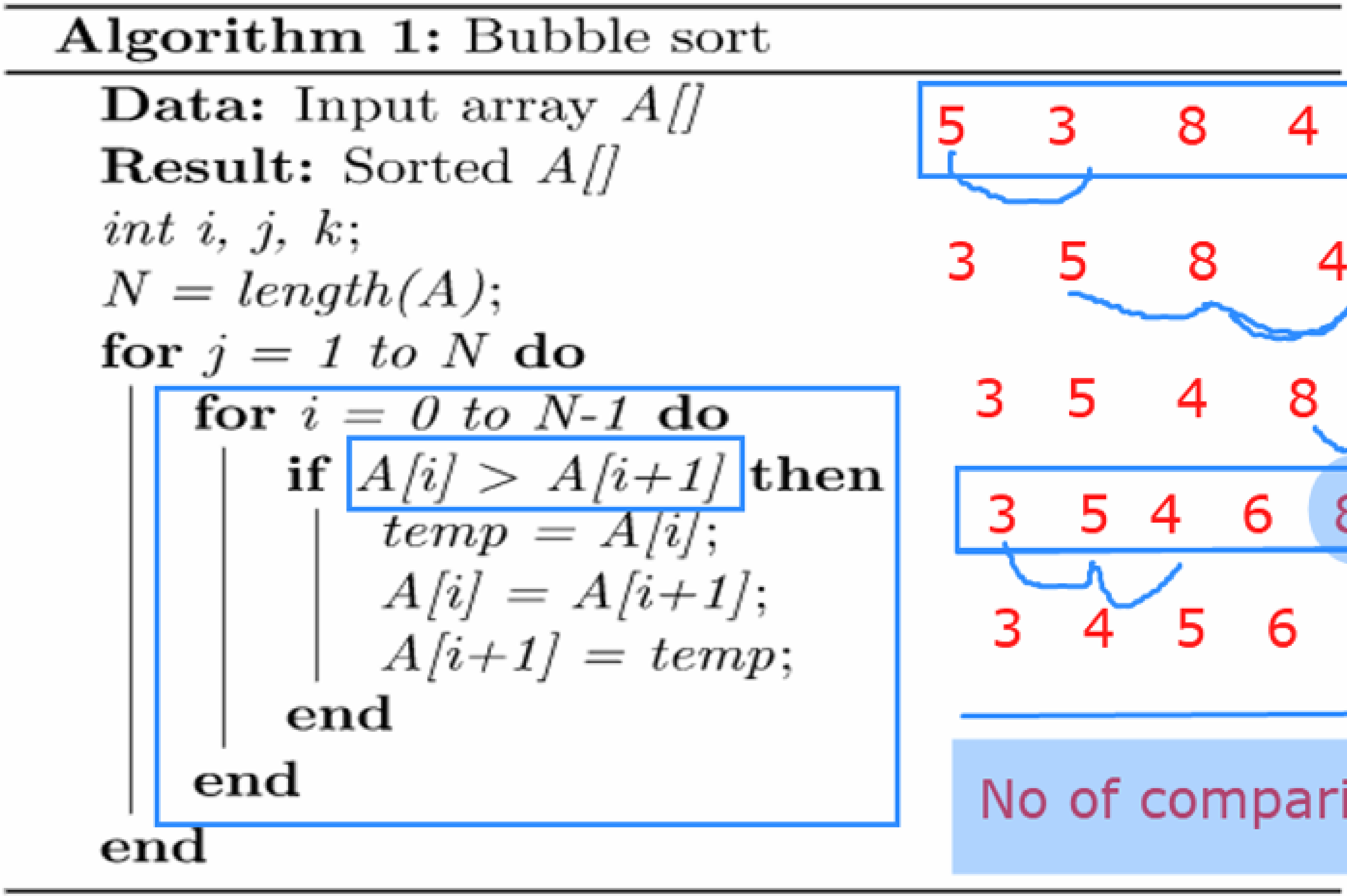
$O(n)$ - Fair

$O(logn)$ - Good

$O(1)$ - Best

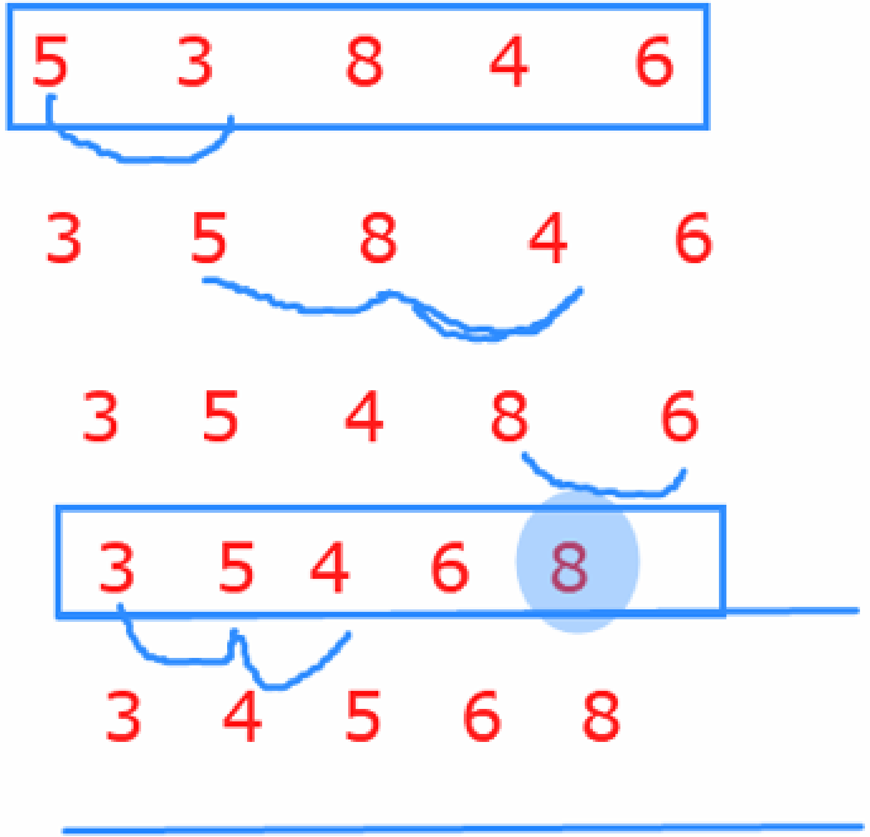


```
-Types of Sorting methods
    -Internal sorting
        -data is to be adjusted in main memory.
    -External sorting
        -data is to be adjusted in main memory,
        so addinal auxilary memory is utilized for sorting process.
```

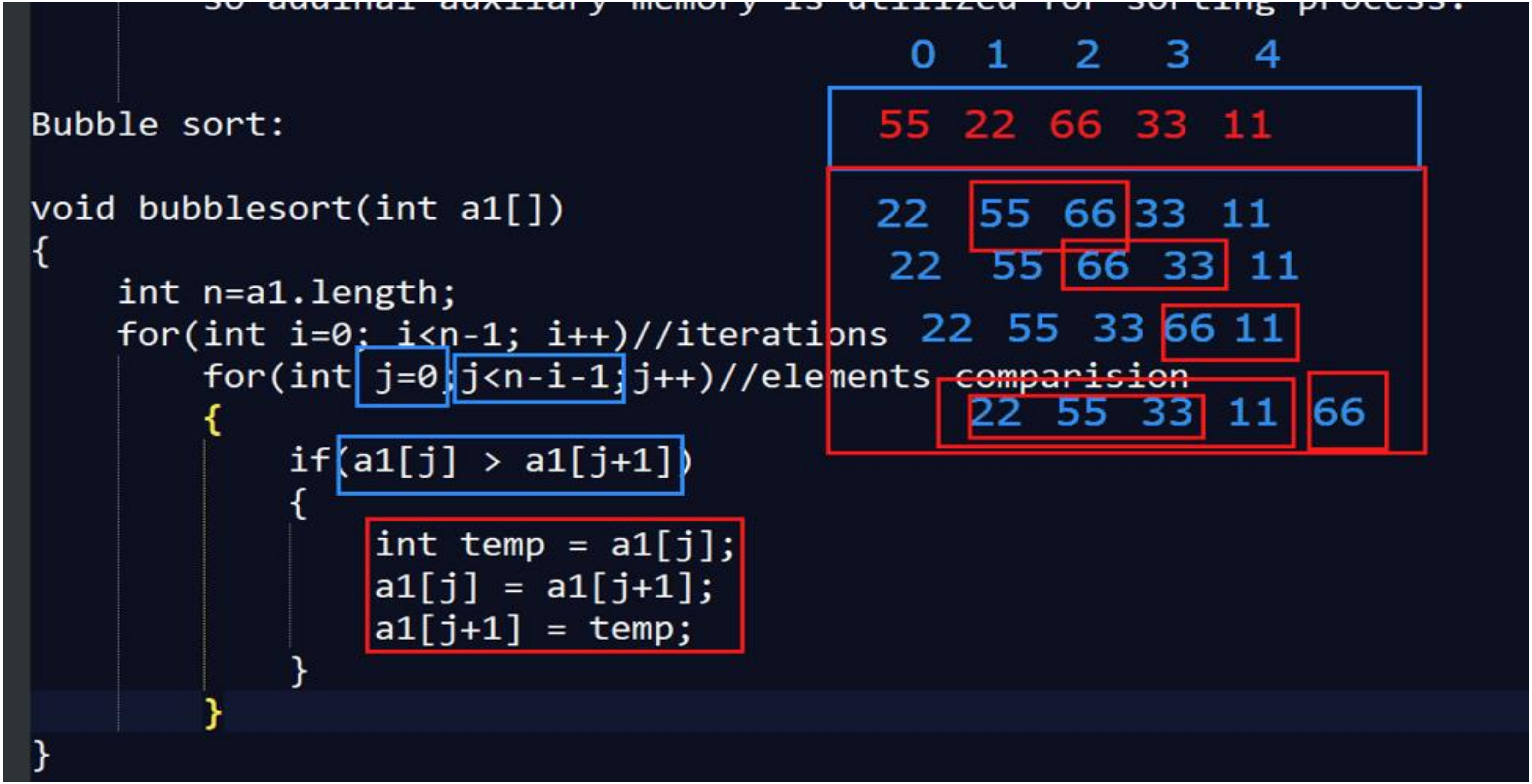


Algorithm 1: Bubble sort

```
Data: Input array $A[]$
Result: Sorted $A[]$
$int\ i,\ j,\ k;$
$N = length(A);$
for $j$ = $1$ to $N$ do
    for $i$ = $0$ to $N$-$1$ do
        if $A[i] > A[i+1]$ then
            $temp = A[i];$
            $A[i] = A[i+1];$
            $A[i+1] = temp;$
        end
    end
end
```

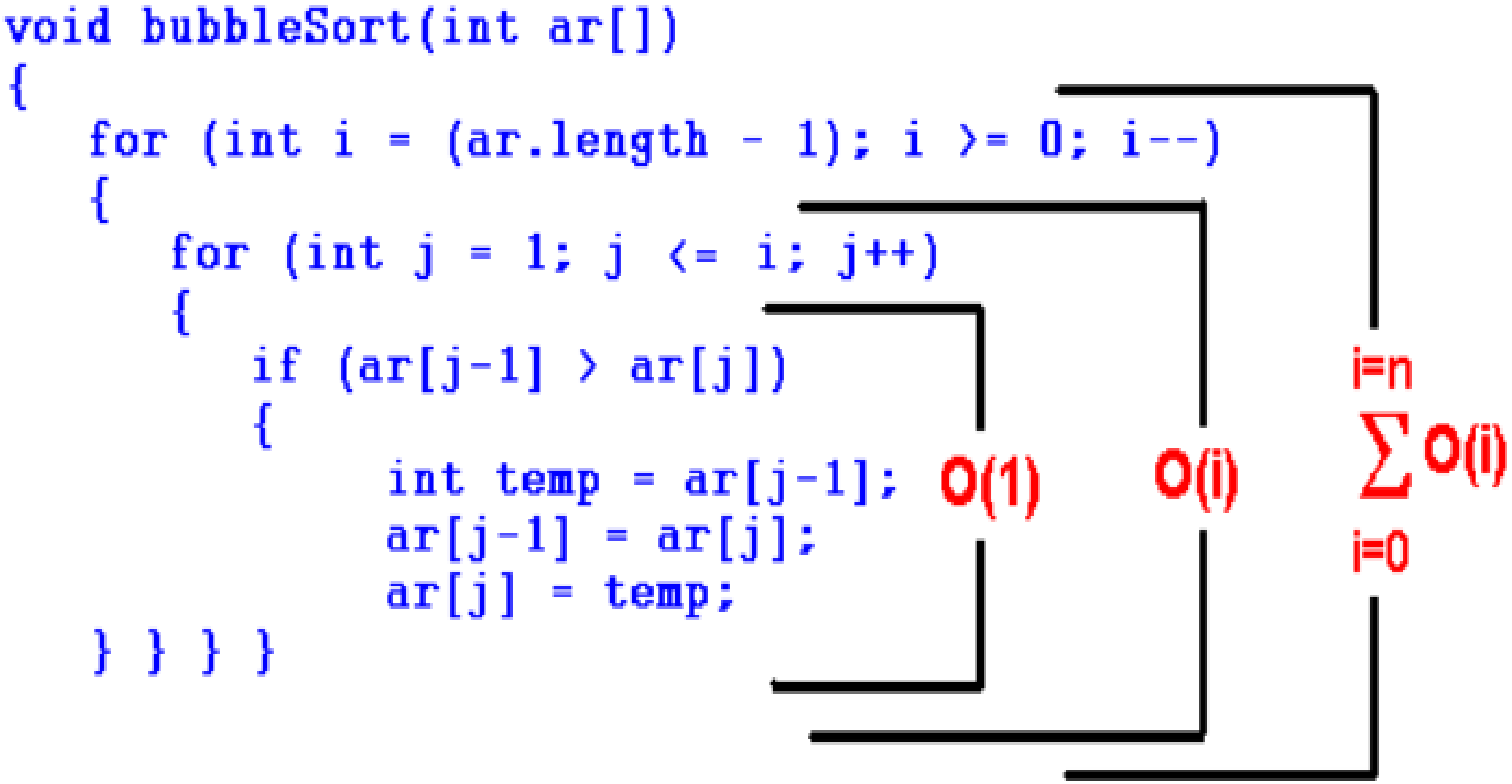


No of comparisions: n-1



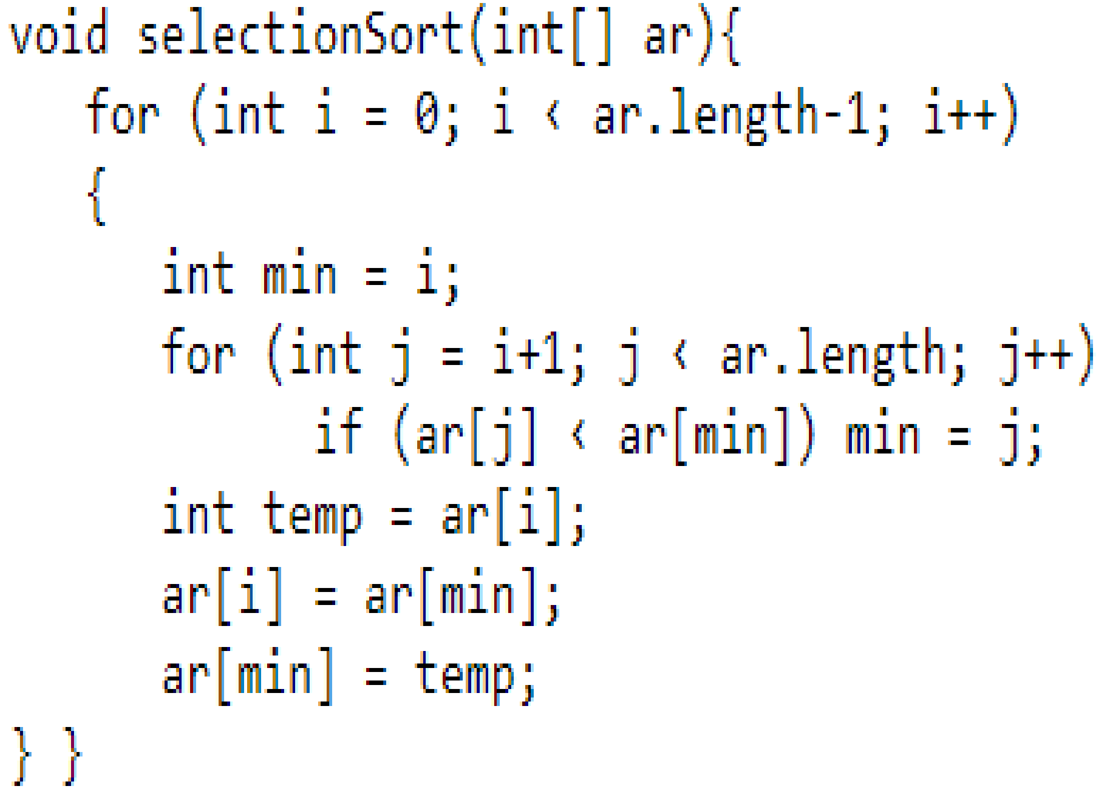
```
so addinal auxiliary memory is utilized for sorting process.

Bubble sort:

void bubblesort(int a1[])
{
    int n=a1.length;
    for(int i=0; i<n-1; i++)//iterations
        for(int j=0;j<n-i-1;j++)//elements comparision
        {
            if(a1[j] > a1[j+1])
            {
                int temp = a1[j];
                a1[j] = a1[j+1];
                a1[j+1] = temp;
            }
        }
}
```

```
 0  1  2  3  4
55 22 66 33 11

22 55 66 33 11
22 55 66 33 11
22 55 33 66 11
22 55 33 11 66
```



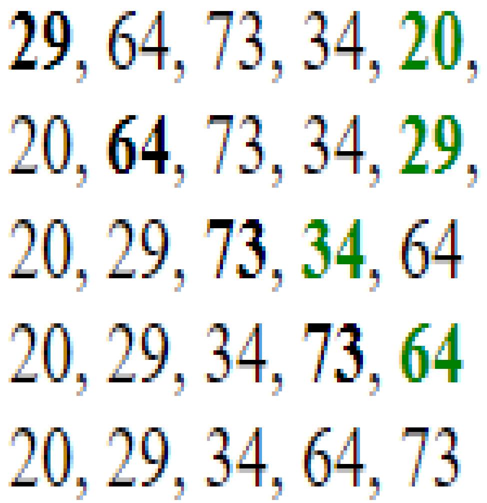
```
void bubbleSort(int ar[])
{
   for (int i = (ar.length - 1); i >= 0; i--)
   {
      for (int j = 1; j <= i; j++)
      {
         if (ar[j-1] > ar[j])
         {
              int temp = ar[j-1];
              ar[j-1] = ar[j];
              ar[j] = temp;
   } } } }
```

$$\sum_{i=0}^{i=n} O(i) = 1 + 2 + 3 + \ldots + (n-1) = O(n^2)$$



```
void selectionSort(int[] ar){
   for (int i = 0; i < ar.length-1; i++)
   {
      int min = i;
      for (int j = i+1; j < ar.length; j++)
            if (ar[j] < ar[min]) min = j;
      int temp = ar[i];
      ar[i] = ar[min];
      ar[min] = temp;
} }
```

Example.



29, 64, 73, 34, **20**,

20, **64**, 73, 34, **29**,

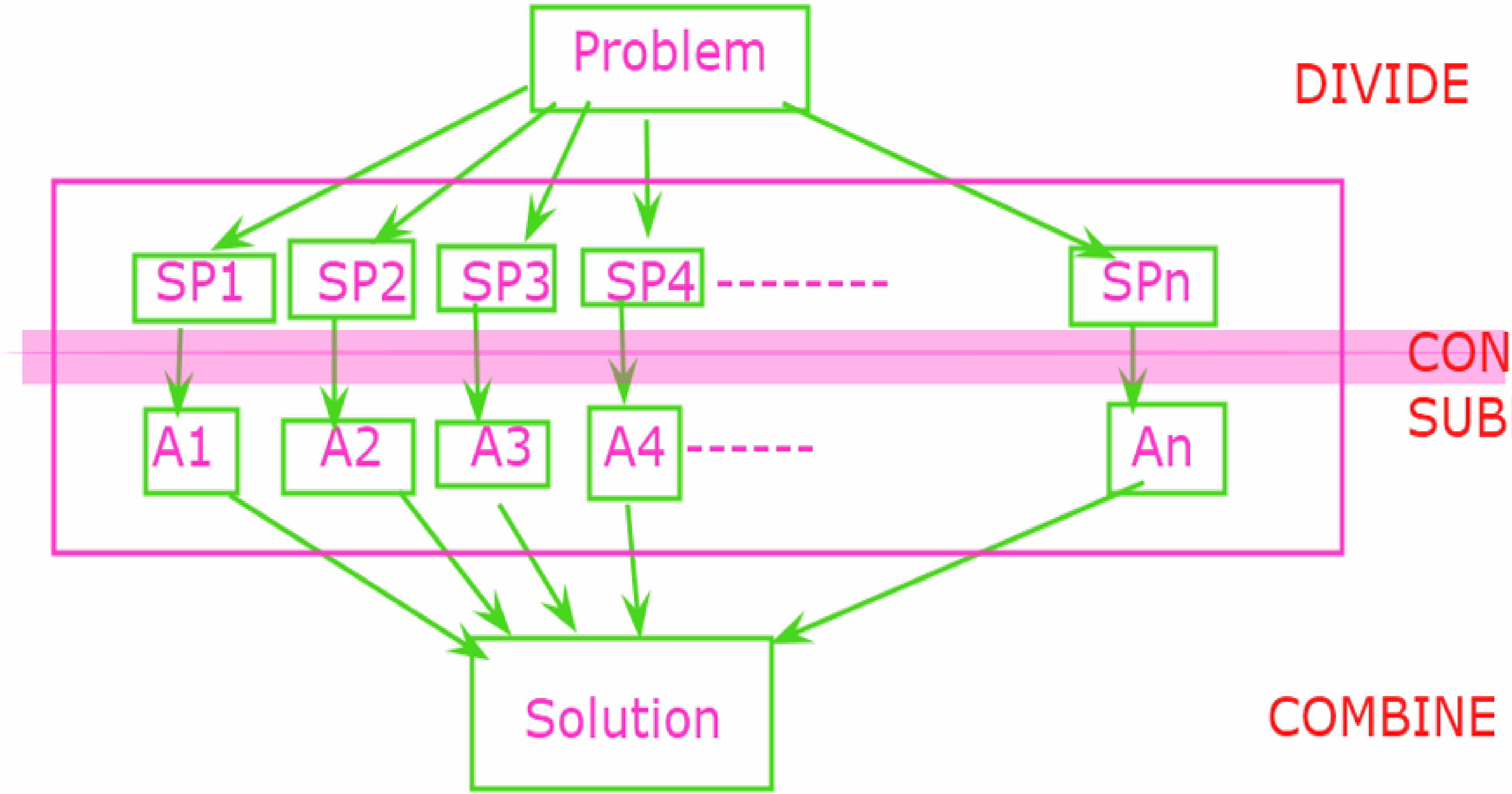
20, 29, **73**, **34**, 64

20, 29, 34, **73**, **64**

20, 29, 34, 64, 73

Divide-and-Conquer

- **Divide the problem into a number of sub-problems**
    - Similar sub-problems of smaller size
- **Conquer the sub-problems**
    - Solve the sub-problems recursively
    - Sub-problem size small enough ⇒ solve the problems in straightforward manner
- **Combine the solutions of the sub-problems**
    - Obtain the solution for the original problem

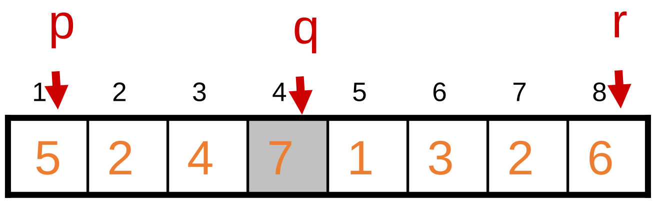


DIVIDE

CONQUER THE SUBPROBLEMS

COMBINE

Merge Sort

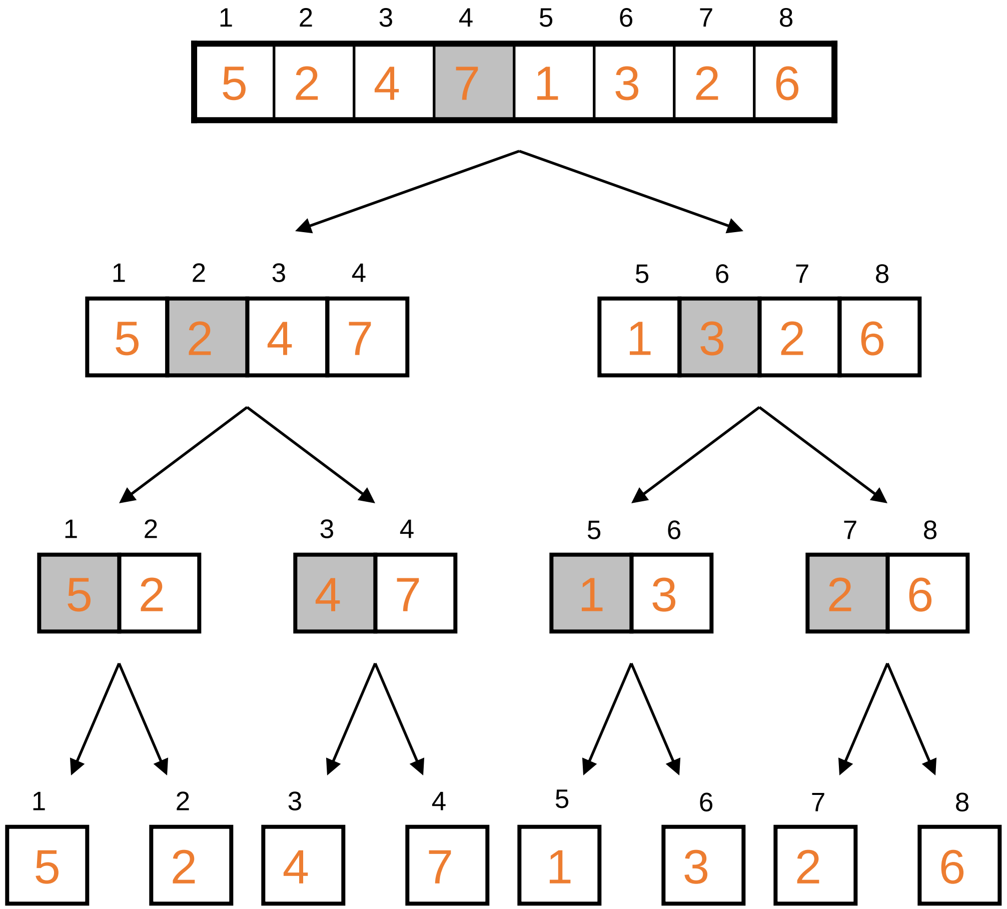


```
Alg.: MERGE-SORT(A, p, r)

  if p < r                                  ▷ Check for base case

    then $q \leftarrow \lfloor (p + r)/2 \rfloor$                ▷ Divide

         MERGE-SORT(A, p, q)           Conquer▷

         MERGE-SORT(A, q + 1, r)            ▷ Conquer

         MERGE(A, p, q, r)                  ▷ Combine
```

- **Initial call:** MERGE-SORT(A, 1, n)

Example – *n* Power of 2

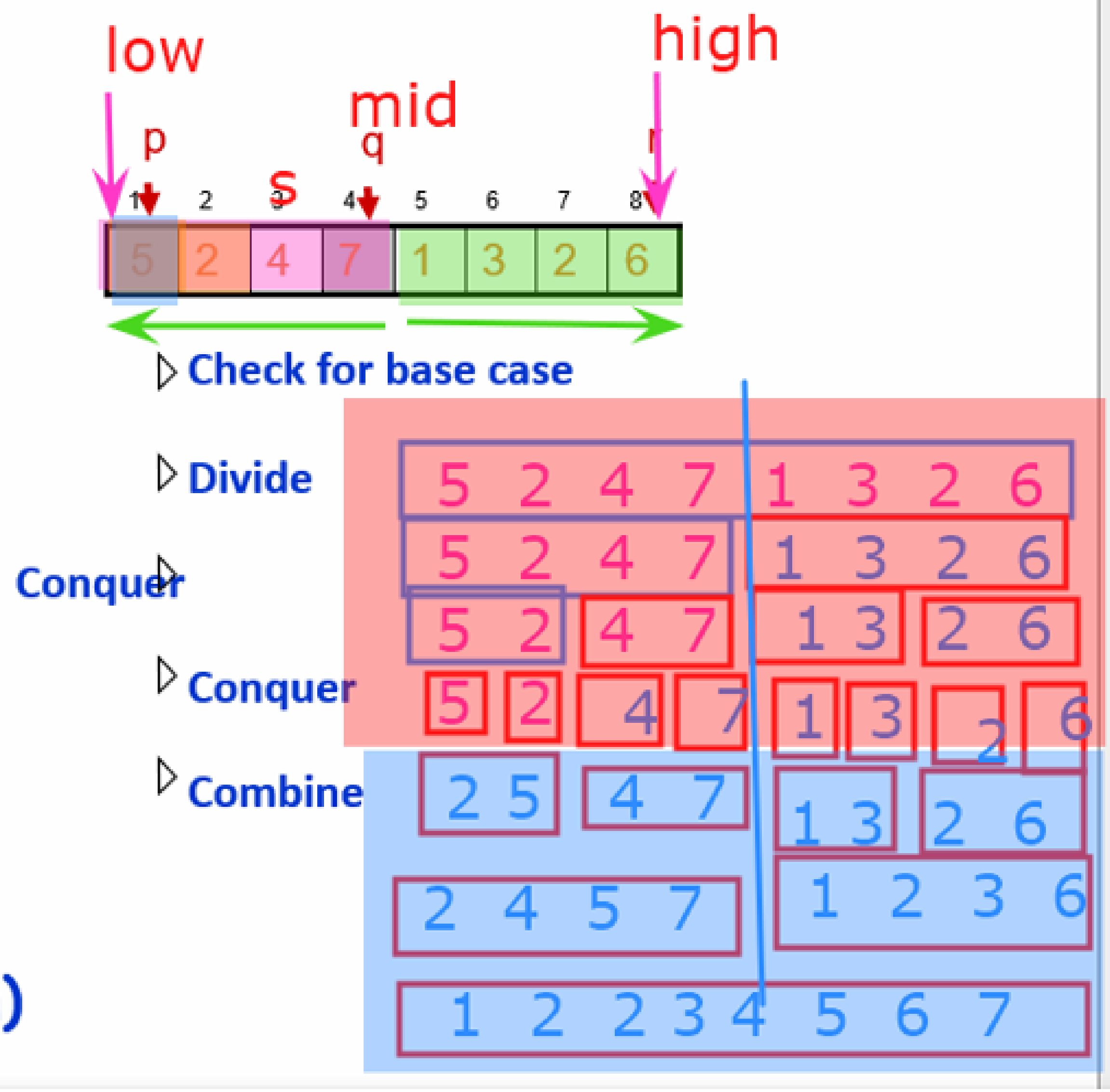
Divide



q = 4

Merge Sort

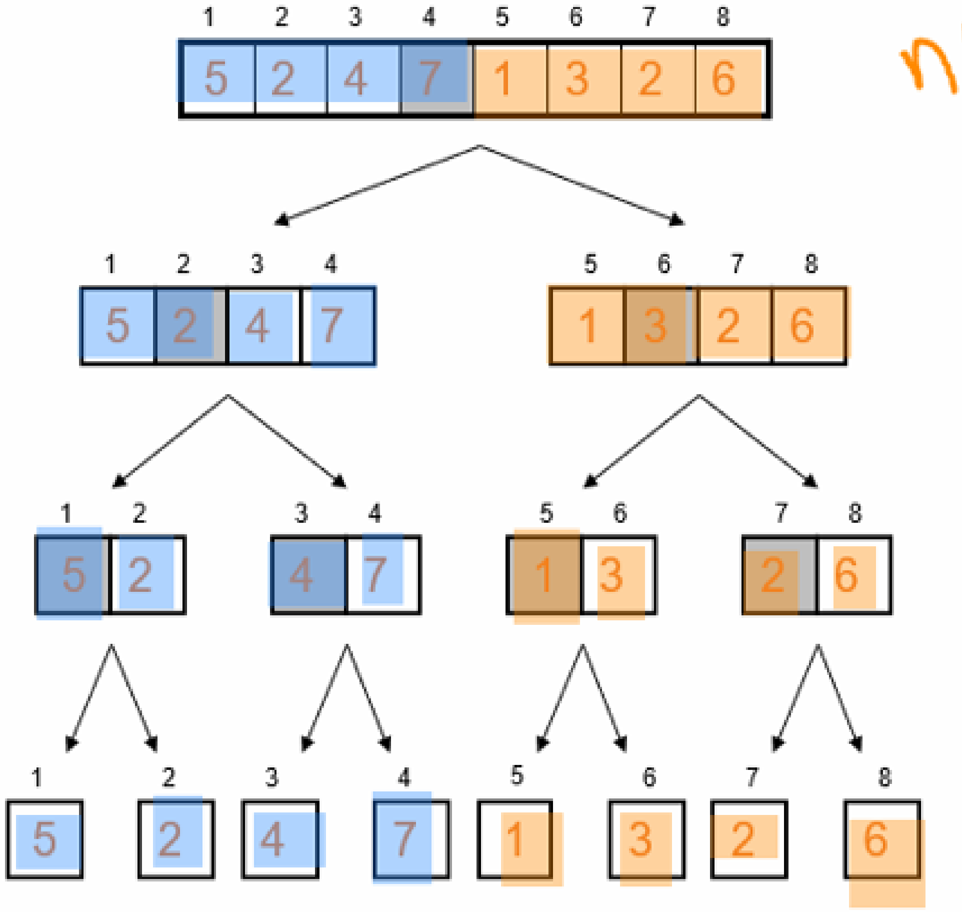
```
Alg.: MERGE-SORT(A, p, r)
    if p < r
        then q ← $\lfloor (p + r)/2 \rfloor$
            MERGE-SORT(A, p, q)
            MERGE-SORT(A, q + 1, r)
            MERGE(A, p, q, r)
```



- **Initial call: MERGE-SORT(A, 1, n)**

Example – *n* Power of 2

Divide



1 2 3 4 5 6 7 8

5 2 4 7 1 3 2 6

1 2 3 4 | 5 6 7 8

5 2 4 7 | 1 3 2 6

1 2 | 3 4 | 5 6 | 7 8

5 2 | 4 7 | 1 3 | 2 6

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

5 | 2 | 4 | 7 | 1 | 3 | 2 | 6

q = 4

$O(n \log n)$

$O(\log n)$

$n/2$

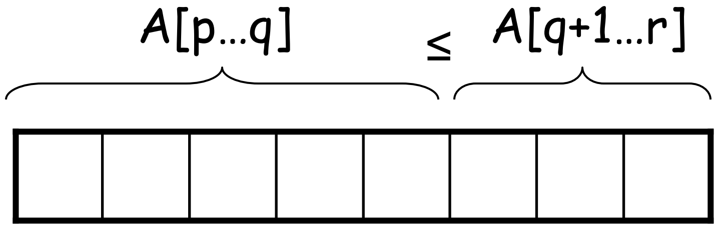
$(n/2)/2 = n/4 = \frac{n}{2^2}$

$(n/4)/2 = n/8 = \frac{n}{2^3}$

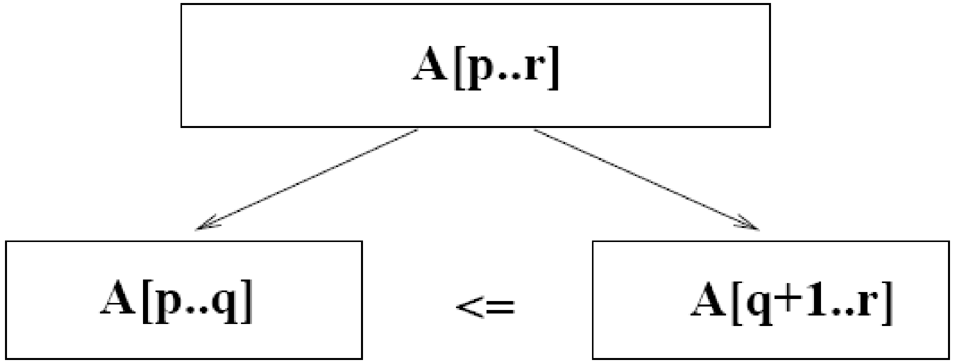
$(n/8)/2 = n/16 = \frac{n}{2^4}$

Quicksort

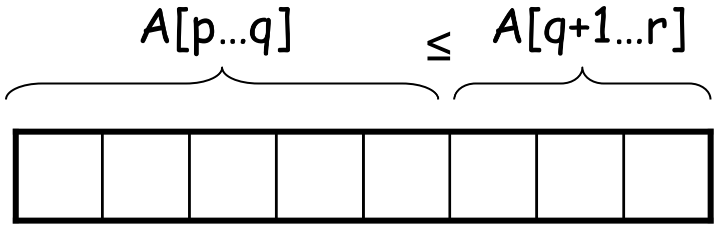
- **Sort an array *A[p…r]***



- **Divide**
  - Partition the array *A* into 2 subarrays *A[p..q]* and *A[q+1..r]*, such that each element of *A[p..q]* is smaller than or equal to each element in *A[q+1..r]*
  - Need to find index *q* to partition the array



Quicksort



- **Conquer**
  - Recursively sort $A[p..q]$ and $A[q+1..r]$ using Quicksort
- **Combine**
  - Trivial: the arrays are sorted in place
  - No additional work is required to combine them
  - The entire array is now sorted

QUICKSORT

```
Alg.: QUICKSORT(A, p, r)

  if p < r

    then q ← PARTITION(A, p, r)

            QUICKSORT (A, p, q)

            QUICKSORT (A, q+1, r)
```

Initially: $p=1$, $r=n$

Recurrence:

$$T(n) = T(q) + T(n - q) + f(n)$$

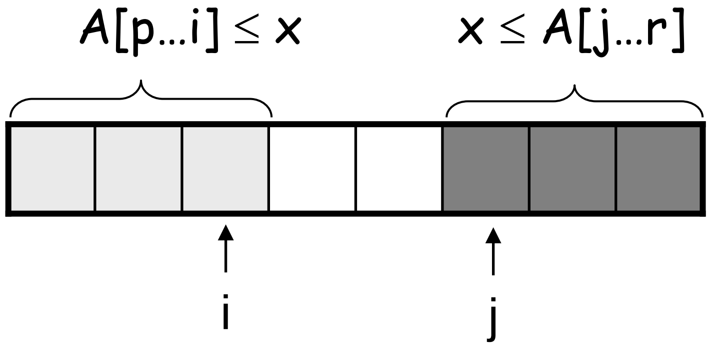
($f(n)$ depends on PARTITION())

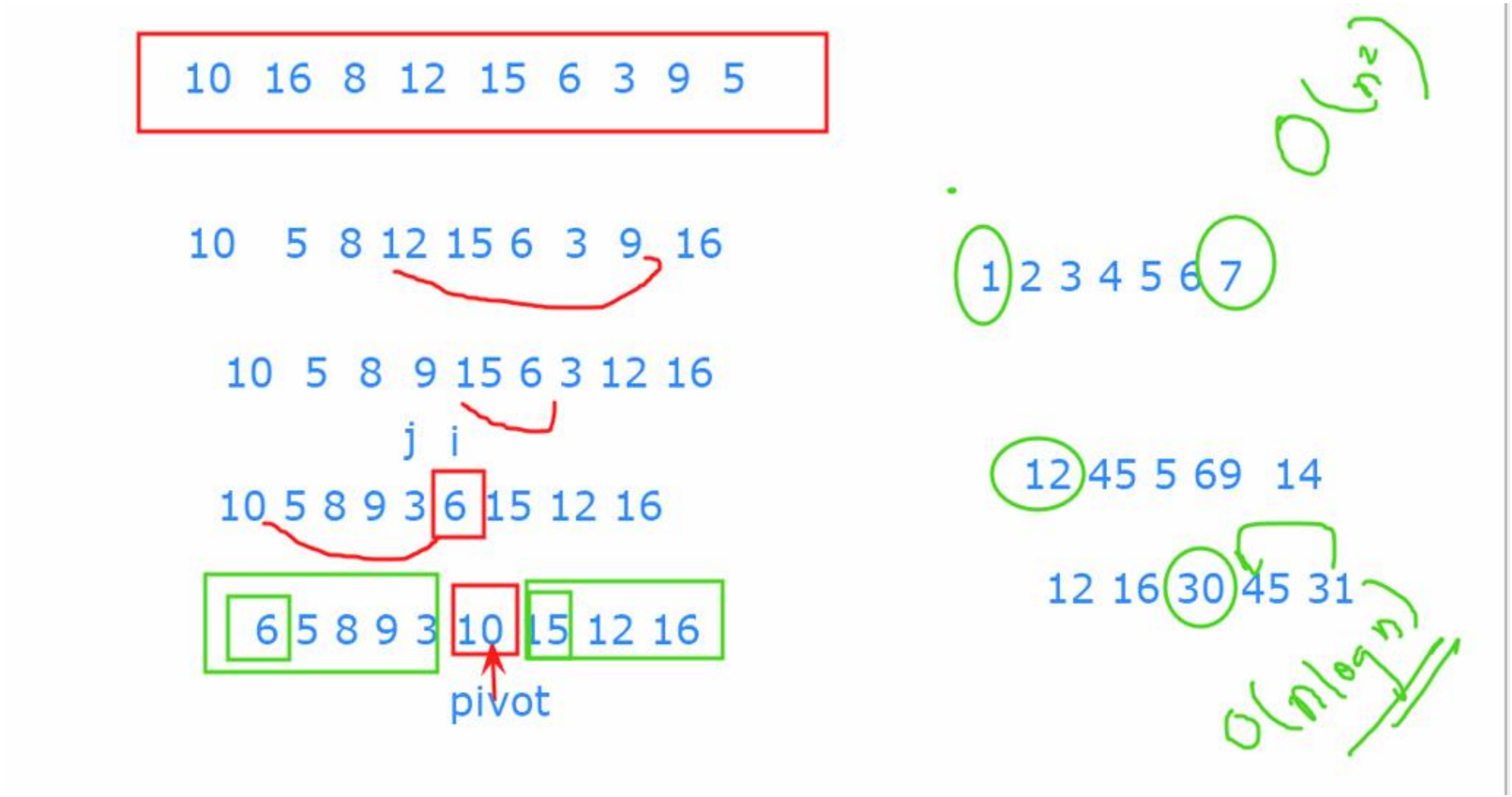
Partitioning the Array

- **Choosing PARTITION()**
  - There are different ways to do this
  - Each has its own advantages/disadvantages
- **Hoare partition**
- **Select a pivot element x around which to partition**
  - Grows two regions

    $A[p \ldots i] \le x$

    $x \le A[j \ldots r]$





10 16 8 12 15 6 3 9 5

10 5 8 12 15 6 3 9 16

10 5 8 9 15 6 3 12 16

j i

10 5 8 9 3 6 15 12 16

6 5 8 9 3 10 15 12 16

pivot

$O(n^2)$

1 2 3 4 5 6 7

12 45 5 69 14

12 16 30 45 31

$O(n \log n)$

Thanks