Sep22 : Day 4

Kiran Waghmare

CDAC Mumbai


Unsorted


Unsorted


Sorting


Sorting


Sorted


Sorted

stable and unstable:

stable sorting

not changing the sequence of arrangement

unstable sorting

changing the sequence of arrangement

-Main meory + auxillary memory

stable and unstable:

stable sorting

not changing the sequence of arrangement

unstable sorting

changing the sequence of arrangement

Efficiency of Algorithms:
-Best case
-Worst case:
-Averge case:

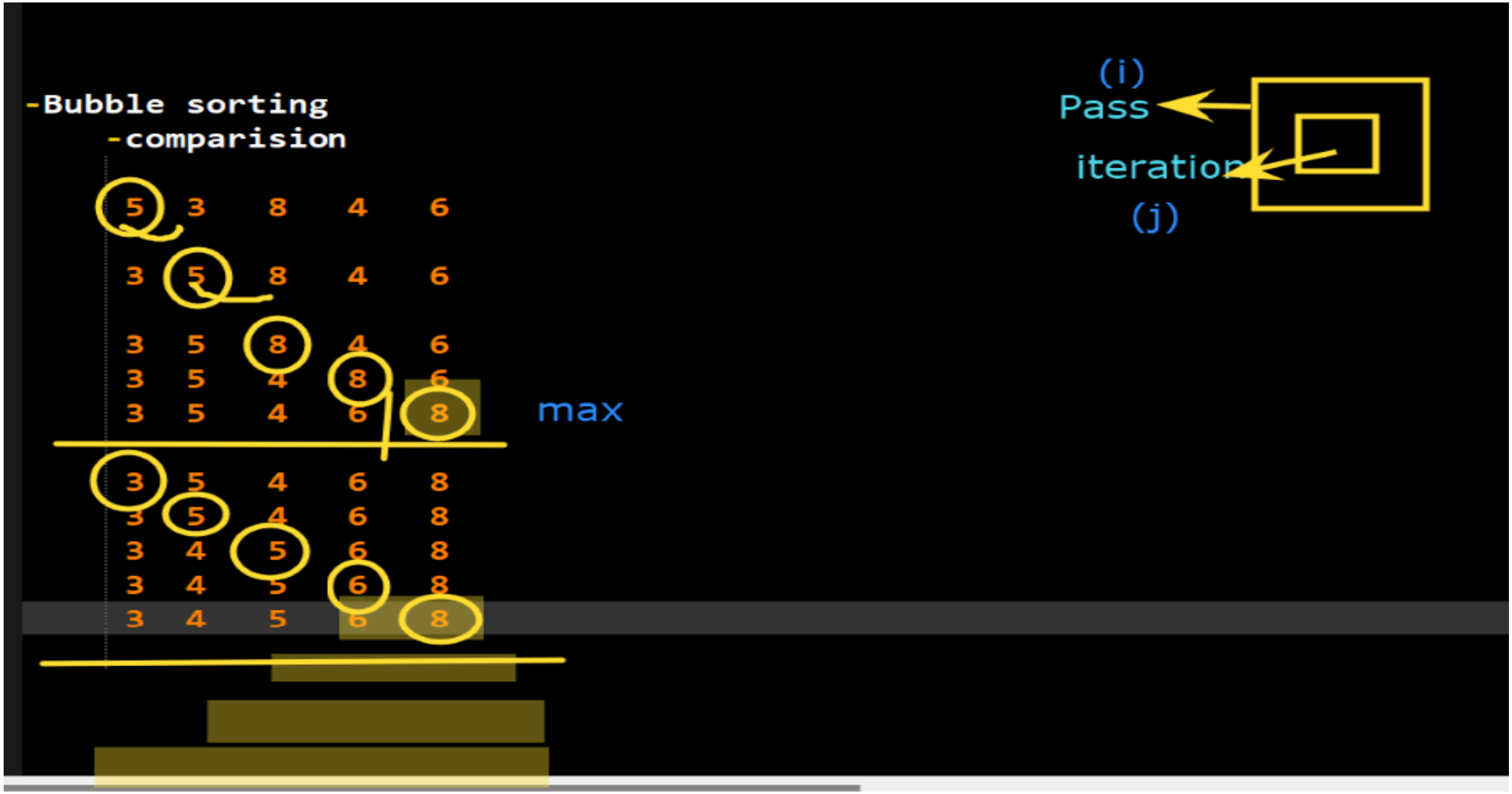
Complexity

O(nlogn)

o(n2)

Size : n

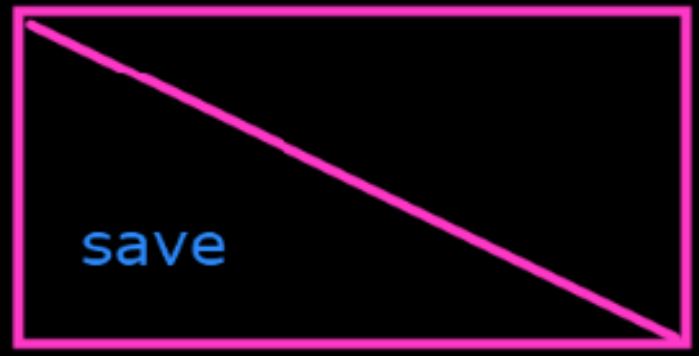
comparision


-Bubble sorting
-comparision
5 3 8 4 6
3 5 8 4 6
3 5 8 4 6
3 5 4 8 6
3 5 4 6 8
max
3 5 4 6 8
3 5 4 6 8
3 4 5 6 8
3 4 5 6 8
3 4 5 6 8
(i)
Pass
iteration
(j)

```
class Sorting{
    static void bsort(int a1[])
    {
    int n =a1.length;
    for(int i=0;i<n-1;i++)//pass
    {
        for(int j=0;j<n-i-1;j++)//internal iterations
        {
            if(a1[j] > a1[j+1])
            {
                //exchange
                int temp = a1[j];
                a1[j]=a1[j+1];
                a1[j+1]=temp;
            }
        }
    }
    }
    static void display(int a1[])
    {
        int n =a1.length;
        for(int i=0;i<n;i++)
        {
            System.out.print(a1[i]+" ");
        }
    }


    public static void main(String args[]){
        //int a1[] = new int[10];
        int a1[]={2,13,24,45,9,30};
```

boolean flag=0;

flag =1;



```java
int n =a1.length;
for(int i=0;i<n-1;i++)//pass
{
    for(int j=0;j<n-i-1;j++)//internal iterations
    {
        if(a1[j] > a1[j+1])
        {
            //exchange
            int temp = a1[j];
            a1[j]=a1[j+1];
            a1[j+1]=temp;

        }

            display(a1);
            System.out.println();

    }
    //display(a1);

}
}
static void display(int a1[])
{
    int n =a1.length;
    for(int i=0;i<n;i++)
    {
        System.out.print(a1[i]+" ");
    }
}
```

1 2 3 4 5 6

Best case: O(n)

n: Elements
n-1: comparisions

6 5 4 3 2 1

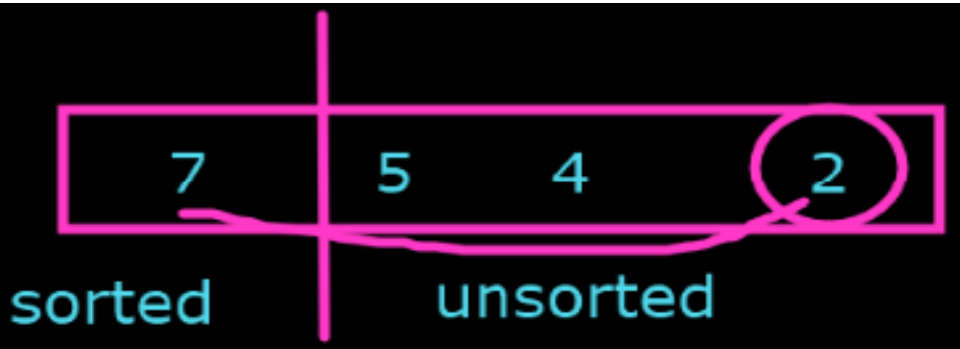
Worst case: $O(n^2)$

Average case: $O(n^2)$

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
```

```
Selection sort:
---------------
min=
7    5    4    2
2    5    4    7
2    4    5    7
2    4    5    7
2    4    5    7
```



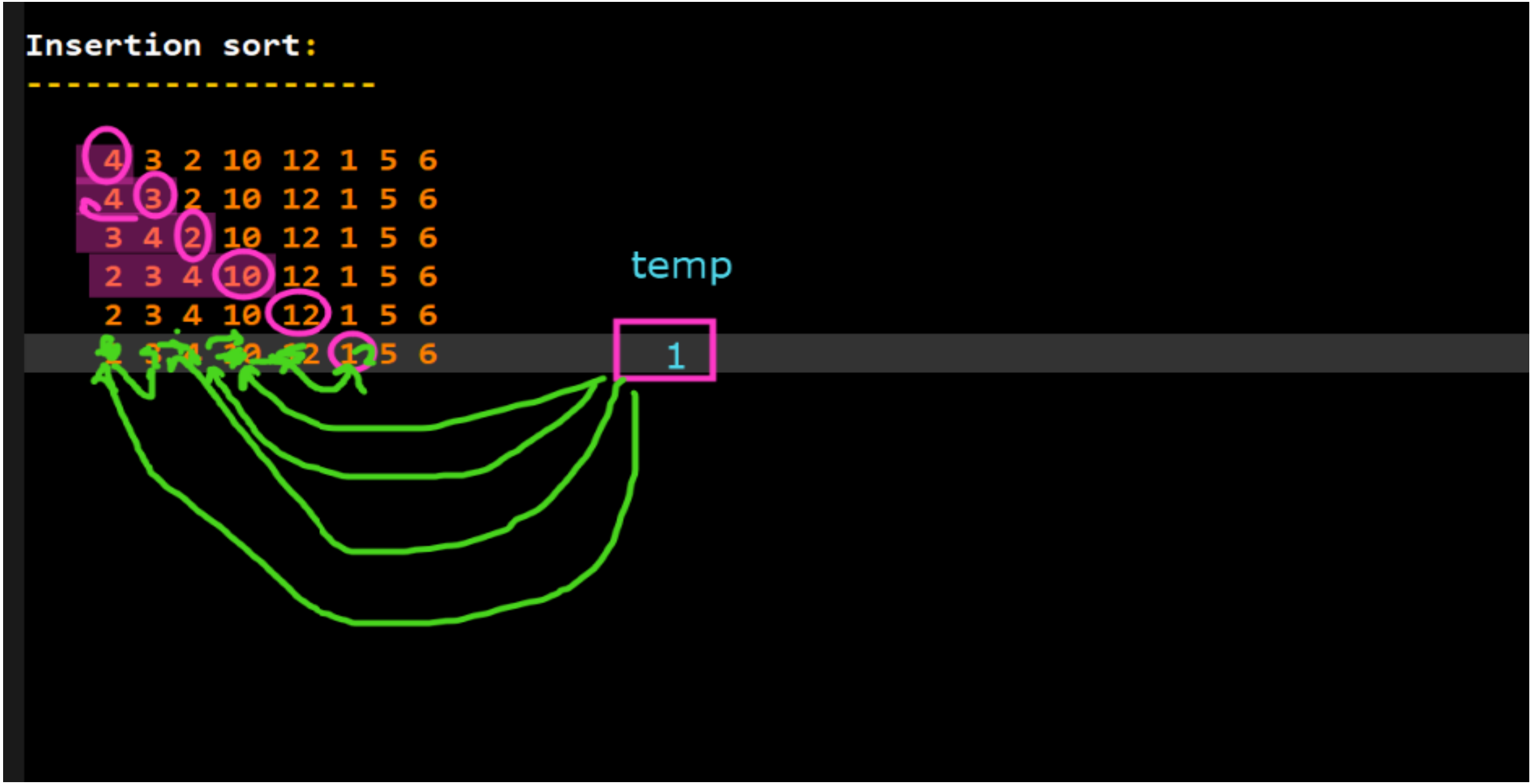
min=2 4

2 4 5 7

Algorithm:

```
SelectionSort(A)
{
        for( i = 0;i < n ;i++)
        {
                least=A[i];
                p=i;
                for ( j = i + 1;j < n ;j++)
                {
                        if (A[j] < A[i])
                        least= A[j]; p=j;
                }
        }
        swap(A[i],A[p]);
}
```

```
void ssort(int a1[])
{
    int n=a1.length;
    for(int i=0;i<n-1;i++)
    {
        int min = i;
        for(int j=i+1;j<n;j++)
        {
            if(a1[j] < a1[min])
                min =j;
        }
        //swapping
        int temp = a1[min];
        a1[min]=a1[i];
        a[i]=temp;
    }
}
```


Insertion sort:
4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
temp
1

INSERTION-SORT(*A*)

```
for j ← 2 to n
    do key ← A[ j ]
    ▷Insert A[ j ] into the sorted sequence A[1 . . j -1]
        i ← j - 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
                i ← i – 1
        A[i + 1] ← key
```

t_j: # of times the while statement is executed at iteration j

```
static void isort(int a1[])
{
    int n=a1.length;
    for(int i=1;i<n;++i)
    {
        int key = a1[i];
        int j=i-1;

        while(j>=0 && a1[j] > key)
        {
            a1[j+1]=a1[j];
            j=j-1;
        }
        a1[j+1]=key;
    }
}
```

```java
static void isort(int a1[])
{
    int n=a1.length;
    for(int i=1;i<n;++i)
    {
        int key = a1[i];
        int j=i-1;

        while(j>=0 && a1[j] > key)
        {
            a1[j+1]=a1[j];
            j=j-1;
        }
        a1[j+1]=key;
        display(a1);
                System.out.println();
    }
}
```

Average case

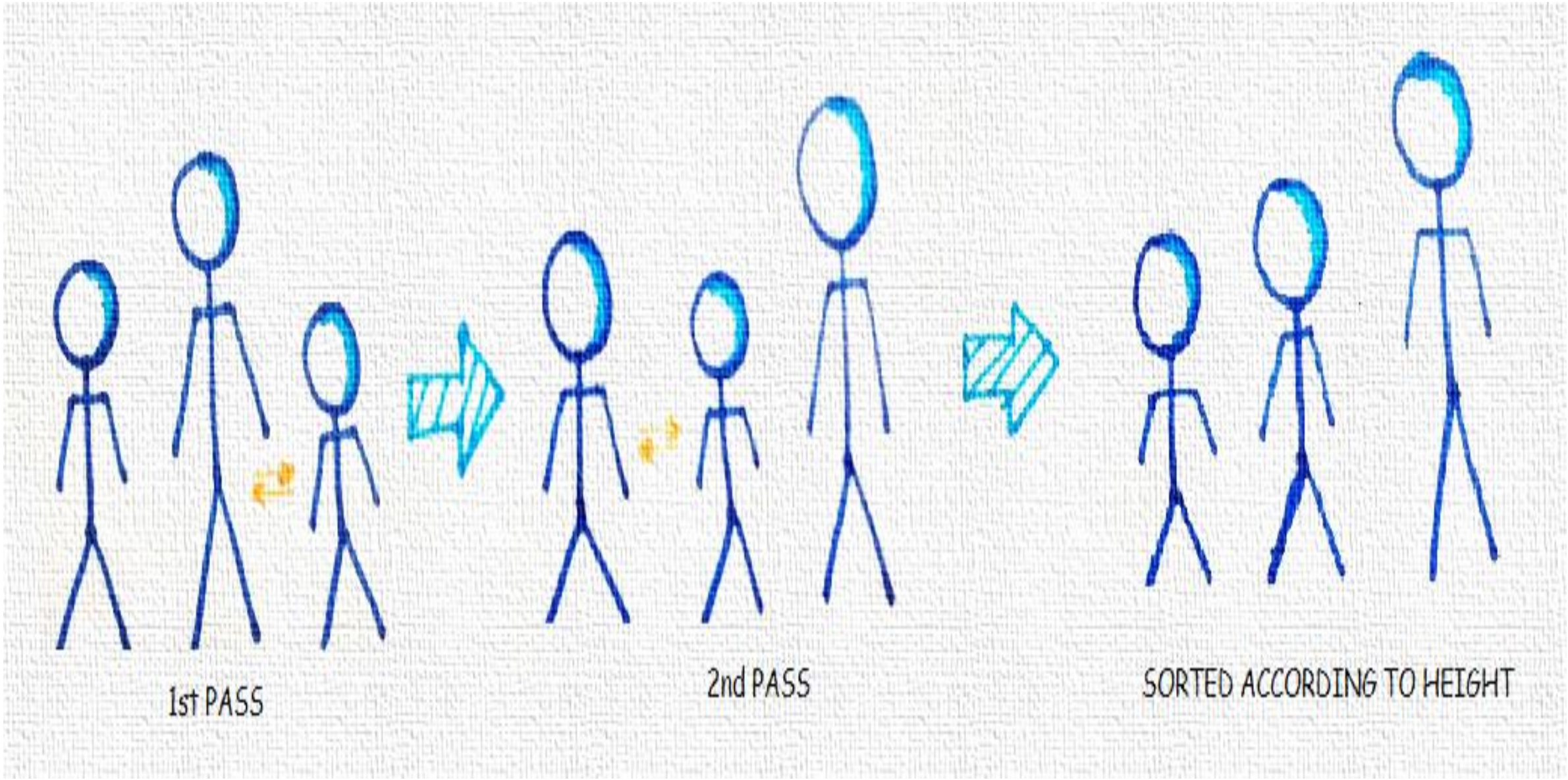
Worst case : $O(n^2)$

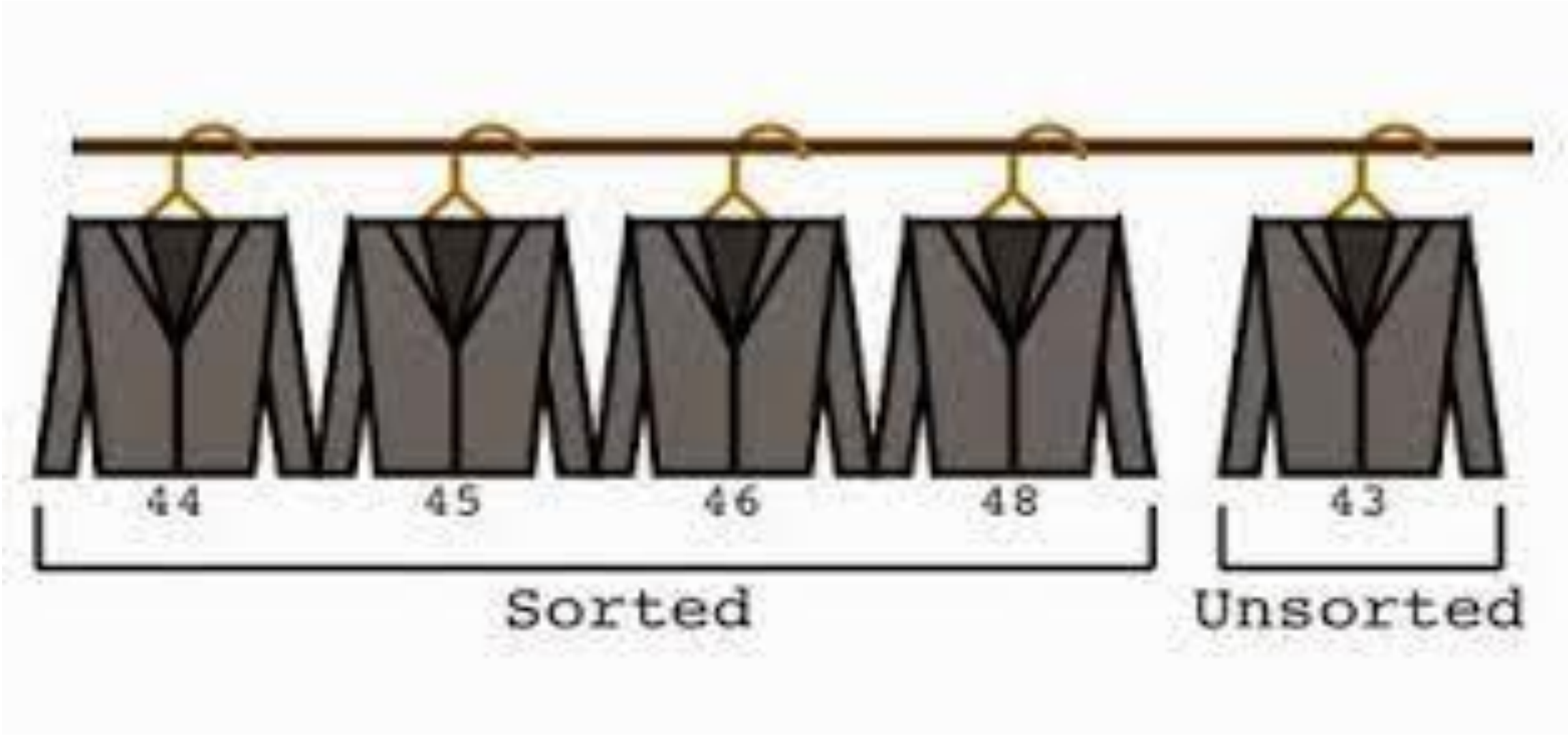
Best Case : Sorted array

1 2 3 4 5

O(n)

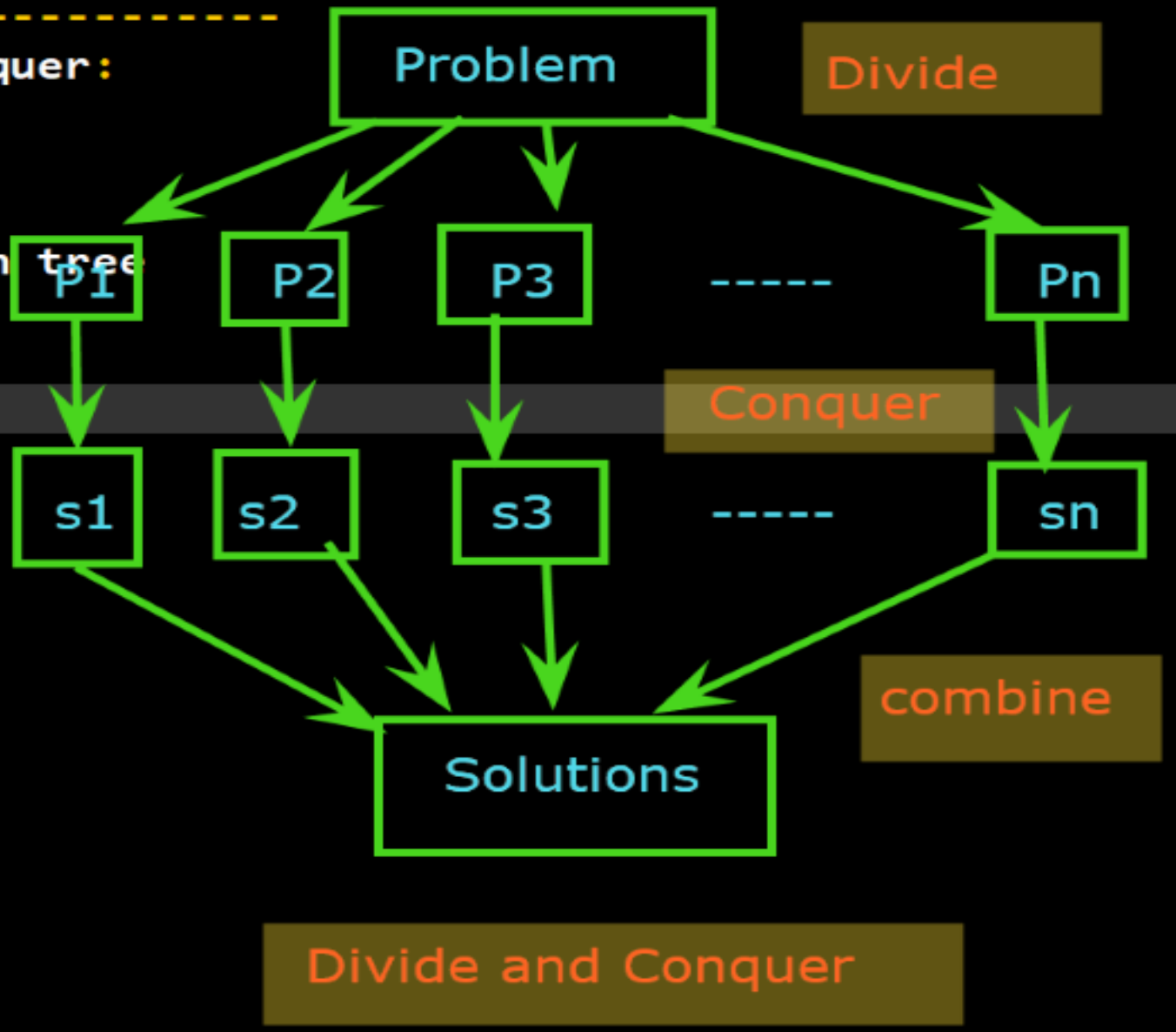
space: O(n)


1st PASS
2nd PASS
SORTED ACCORDING TO HEIGHT

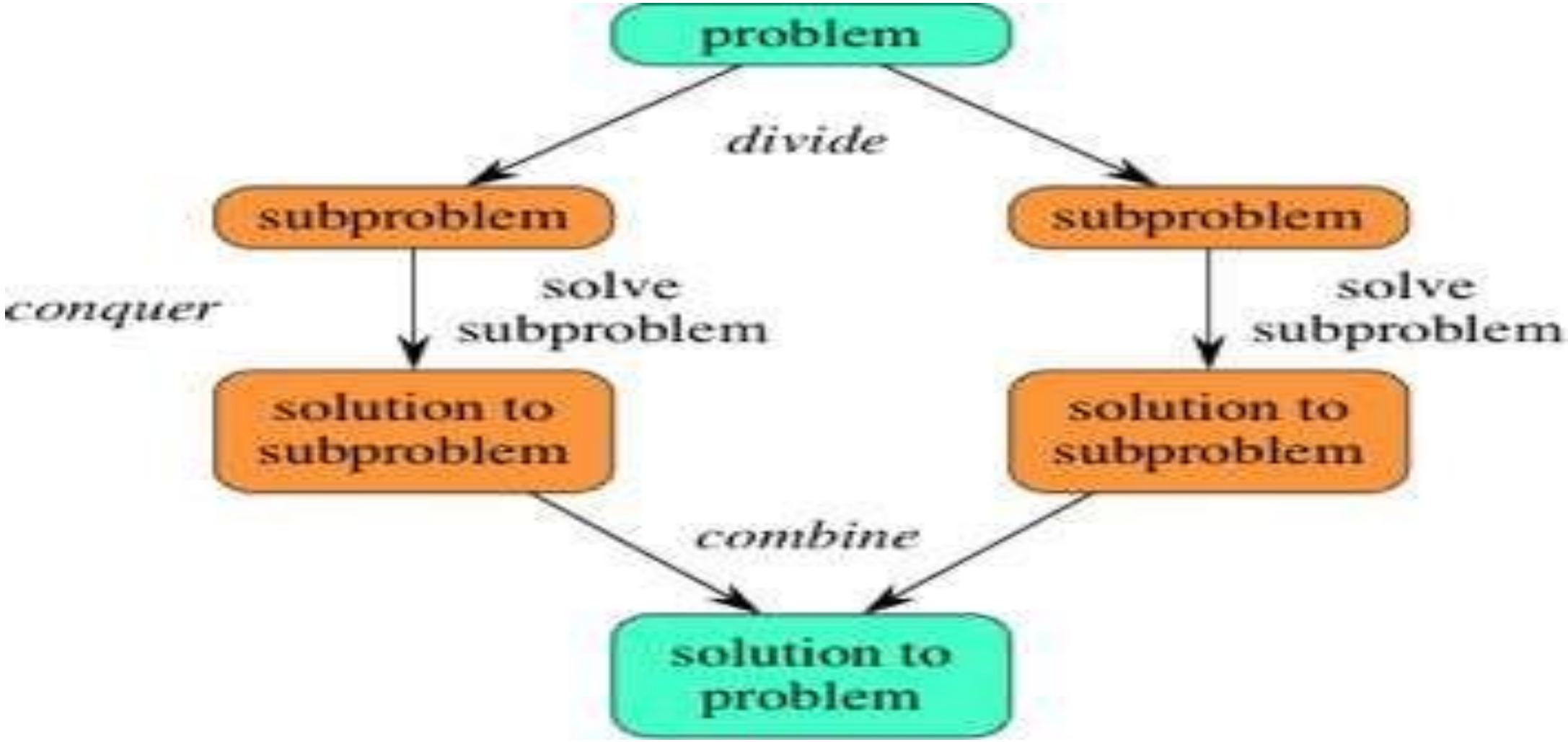

44
45
46
48
43
Sorted
Unsorted

k = log n

Divide and Conquer:
-Binary search
-Merge sort
-Quick sort
-Binsary search tree



Divide-and-conquer



Merge sort:

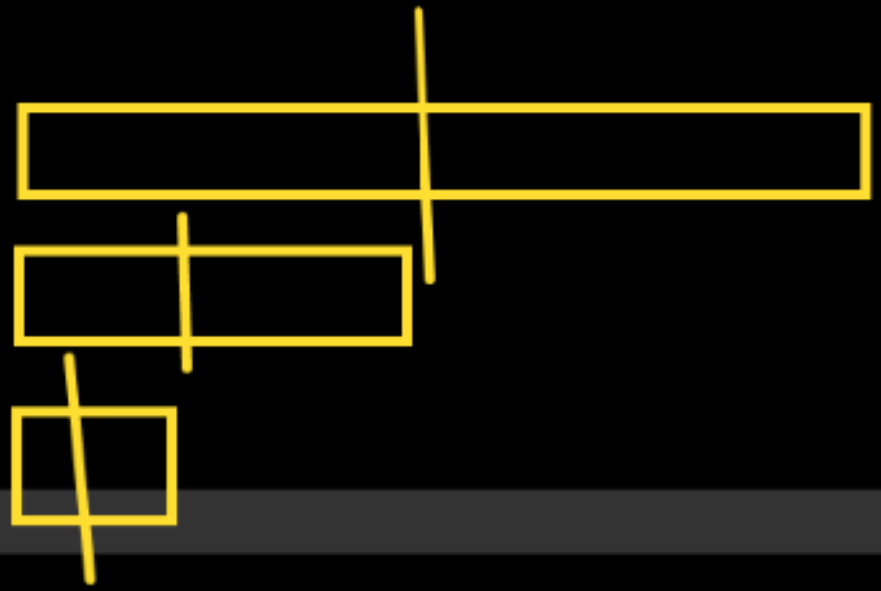
-follws divide and conquer strategy

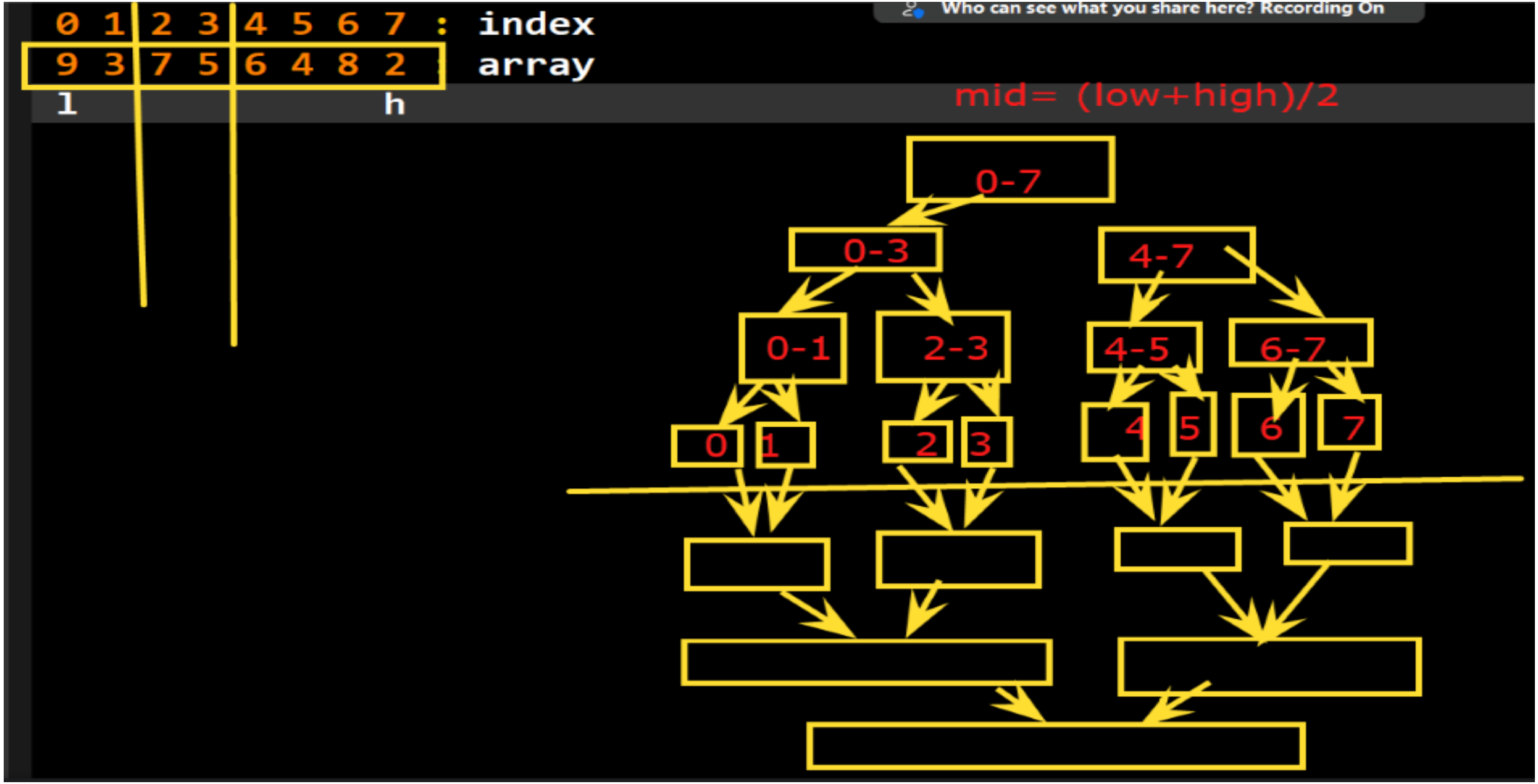
-divide an array in 2 parts

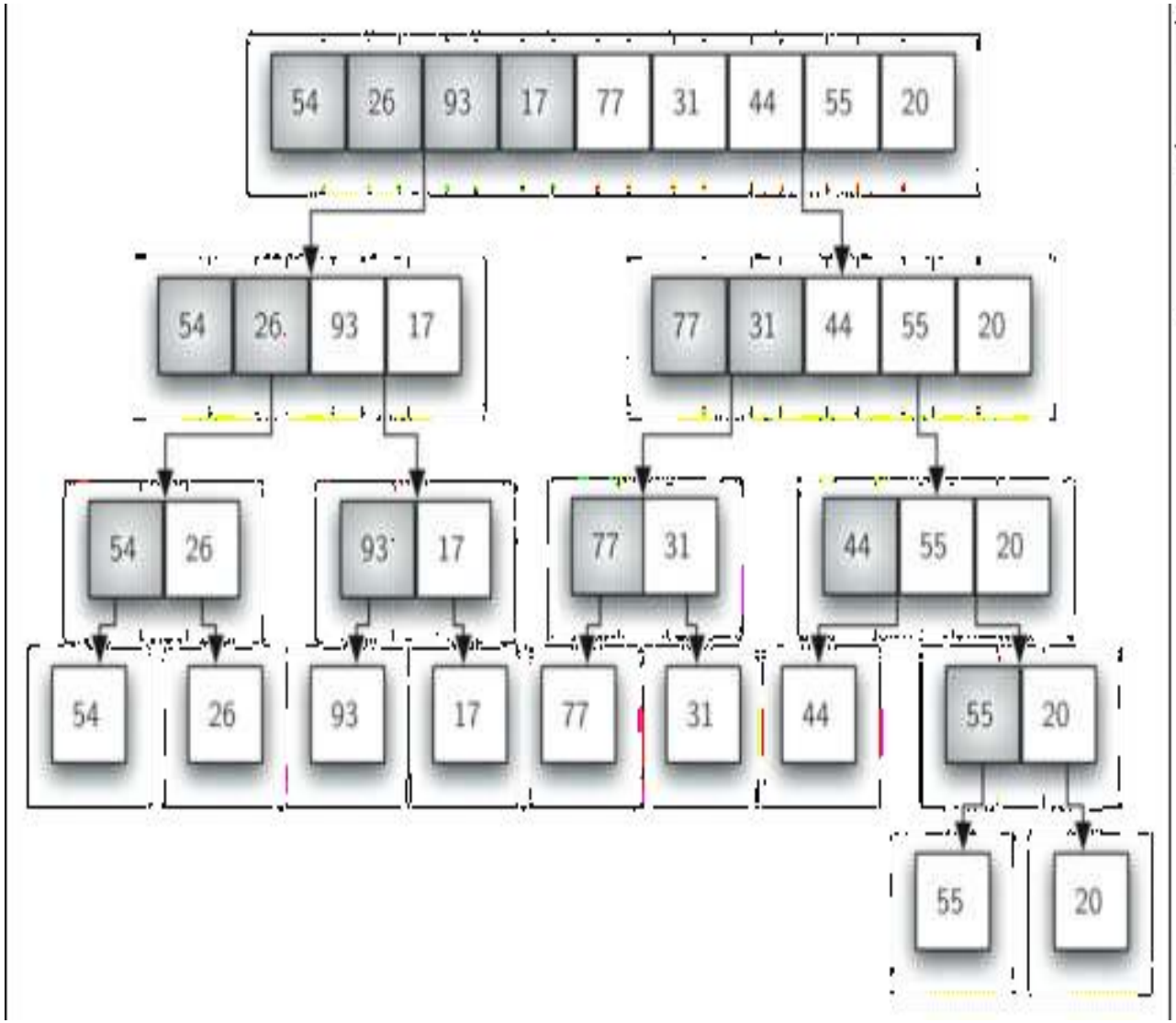
-recursive order

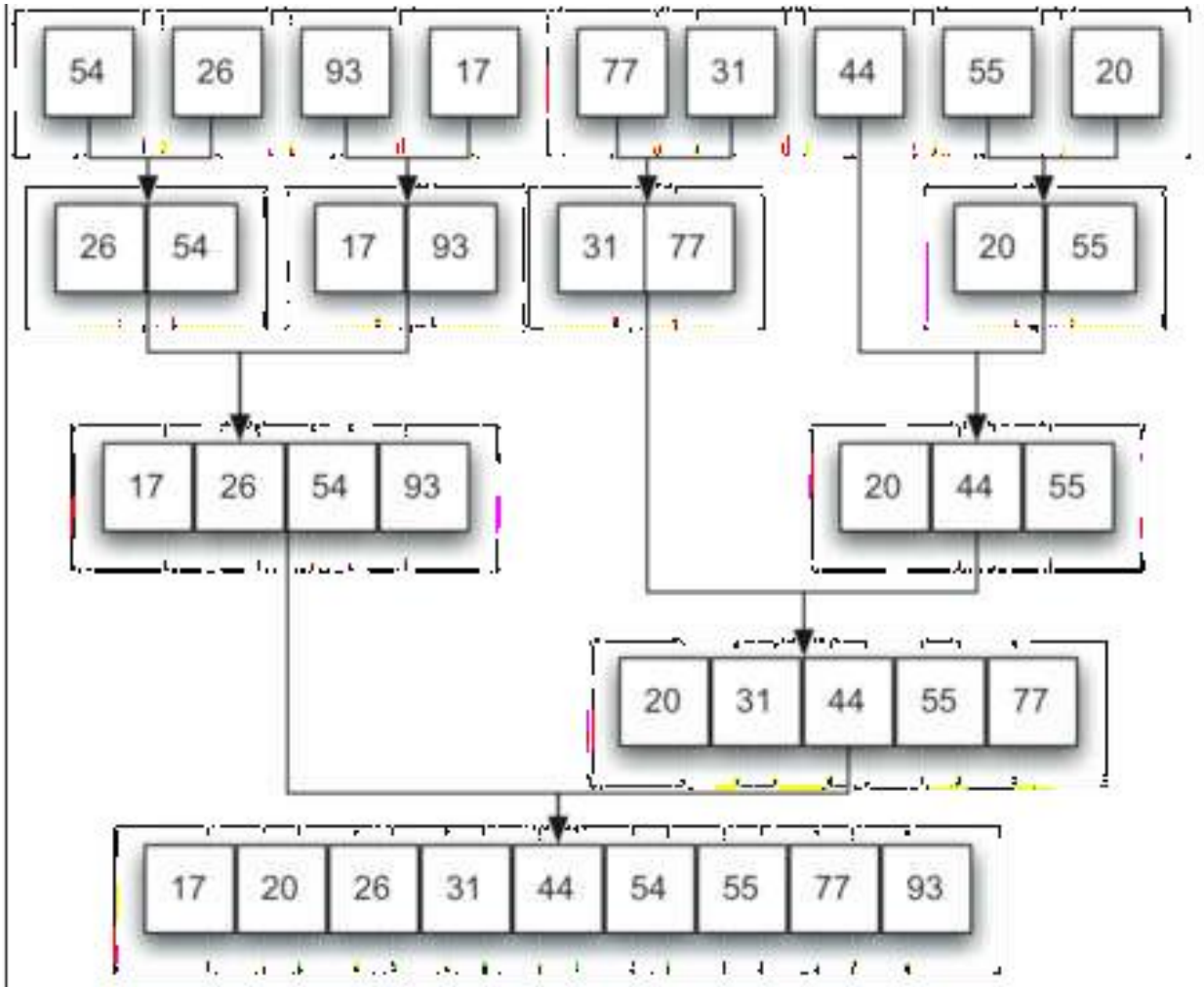
n---->elements in the array

```
          i
A = 2 8 15 18  ➔ m elements
B = 5 9 12 17  ➔ n elements
        j
C = 2 5 8 9 12 15 17 18        (m+n)
```




0 1 2 3 4 5 6 7 : index
9 3 7 5 6 4 8 2 array
l h
Who can see what you share here? Recording On
mid= (low+high)/2
0-7
0-3
4-7
0-1
2-3
4-5
6-7
0
1
2
3
4
5
6
7


54 26 93 17 77 31 44 55 20
54 26 93 17
77 31 44 55 20
54 26
93 17
77 31
44 55 20
54
26
93
17
77
31
44
55 20
55
20


54 26 93 17 77 31 44 55 20
26 54
17 93
31 77
20 55
17 26 54 93
20 44 55
20 31 44 55 77
17 20 26 31 44 54 55 77 93

Merge Sort:

Here is the pseudocode for Merge Sort, modified to include a counter:

```
count ← 0
Merge_Sort(A, p, r)
1       if p < r
2               then    q ← ⌊(p + r)/2⌋
3                       Merge-Sort (A, p, q)
4                       Merge-Sort (A, q+1, r)
5                       Merge (A, p, q, r)
```

And here is the modified algorithm for the Merge function used by Merge Sort:

```
Merge (A, p, q, r)
1       n1 ← (q − p) + 1
2       n2 ← (r − q)
3       create arrays L[1..n1+1] and R[1..n2+1]
4       for i ← 1 to n1 do
5               L[i] ← A[(p + i) −1]
6       for j ← 1 to n2 do
7               R[j] ← A[q + j]
8       L[n1 + 1] ← ∞
9       R[n2 + 1] ← ∞
10      i ← 1
11      j ← 1
12      for k ← p to r do
12.5            count ← count + 1
13              if L[I] <= R[j]
14                      then  A[k] ← L[i]
15                            i ←i + 1
16                      else A[k] ← R[j]
17                            j ← j + 1
```

Merge Sort:

Here is the pseudocode for Merge Sort, modified to include a counter:

```
count ← 0
Merge_Sort(A, p, r)
1       if p < r
2               then  q ← ⌊(p + r)/2⌋
3                     Merge-Sort (A, p, q)
4                     Merge-Sort (A, q+1, r)
5                     Merge (A, p, q, r)
```

And here is the modified algorithm for the Merge function used by Merge Sort:

```
Merge (A, p, q, r)
1       n1 ← (q – p) + 1
2       n2 ← (r – q)
3       create arrays L[1..n1+1] and R[1..n2+1]
4       for i ← 1 to n1 do
5               L[i] ← A[(p + i) –1]
6       for j ← 1 to n2 do
7               R[j] ← A[q + j]
8       L[n1 + 1] ← ∞
9       R[n2 + 1] ← ∞
10      i ← 1
11      j ← 1
12      for k ← p to r do
12.5            count ← count + 1
13              if L[I] <= R[j]
14                      then  A[k] ← L[i]
15                            i ←i + 1
16                      else A[k] ← R[j]
17                            j ← j + 1
```

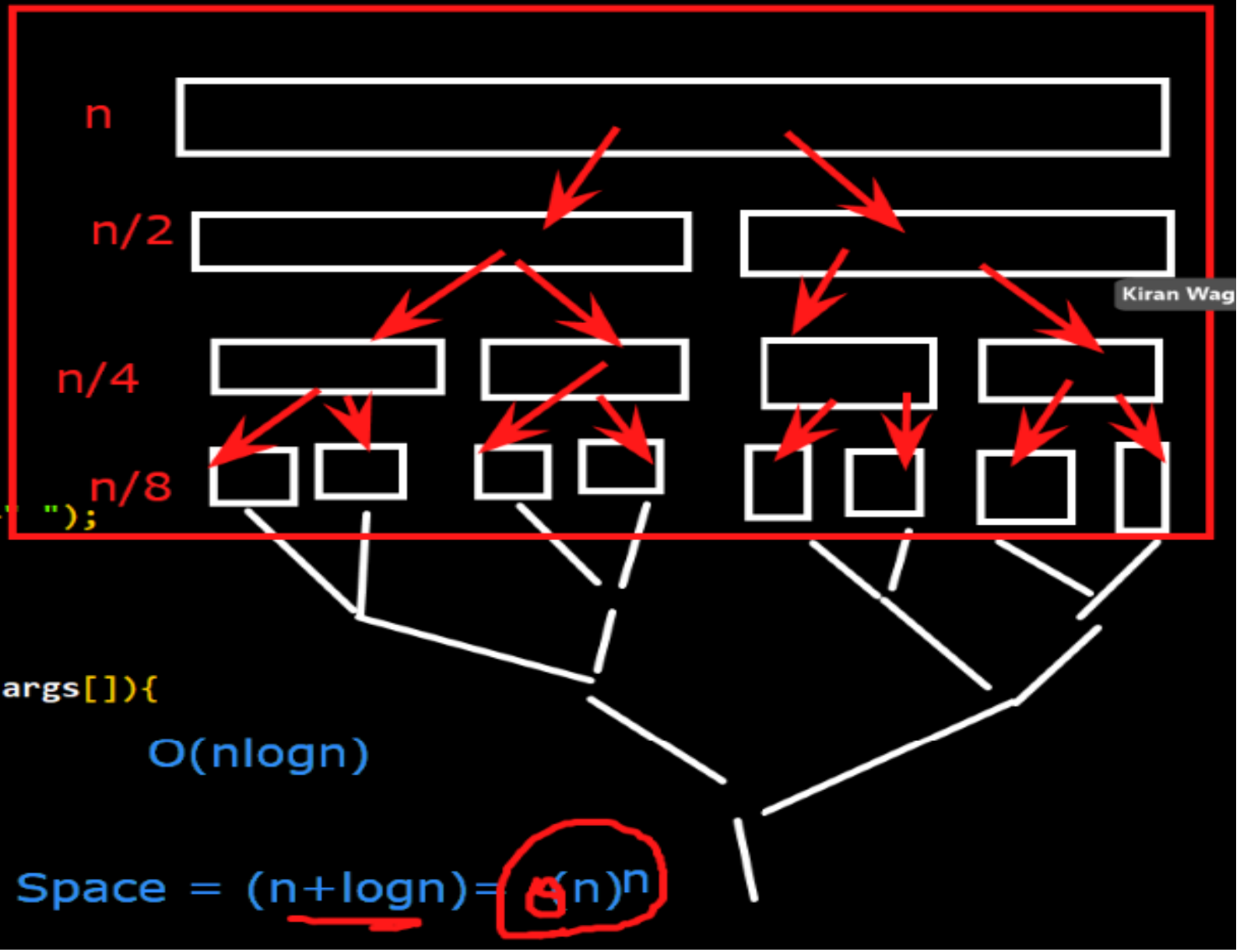
```
    }
    while(j<n2)
    {
        a1[k] = R[j];
        j++;
        k++;
    }

}

    static void display(int a1[])
    {
        int n =a1.length;
        for(int i=0;i<n;i++)
        {
            System.out.print(a1[i]+" ");
        }
    }

    public static void main(String args[]){
        //int a1[] = new int[10];
        int a1[]={2,13,24,45,9,30};
        int n =a1.length;
        display(a1);
        System.out.println();
        //bsort(a1);
        //ssort(a1);
```

n

n

n

n



O(nlogn)

Space = (n+logn)= O(n)

The following procedure implements quicksort:

```
QUICKSORT(A, p, r)
1   if p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q − 1)
4       QUICKSORT(A, q + 1, r)
```

To sort an entire array A, the initial call is QUICKSORT$(A, 1, A.length)$.

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p\,..\,r]$ in place.

```
PARTITION(A, p, r)
1   x = A[r]
2   i = p − 1
3   for j = p to r − 1
4       if A[j] ≤ x
5           i = i + 1
6           exchange A[i] with A[j]
7   exchange A[i + 1] with A[r]
8   return i + 1
```