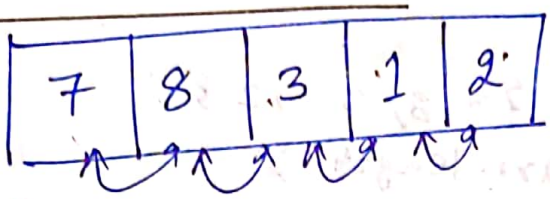
Lecture - 14

Sorting in Java

(1) Bubble Sort



(Unsorted array)

length = n

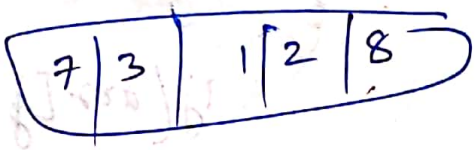
no of pass = (n-1)

(1) 7 8 3 1 2

1st loop

7 3 8 1 2

7 3 1 8 2

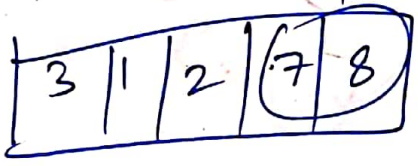


(largest element ⇒ end.

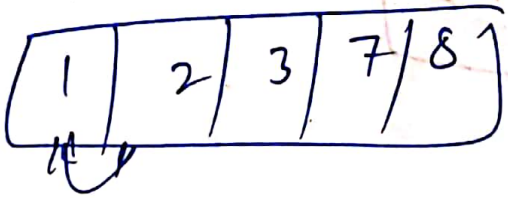
(2) 2nd loop

3 7 1 2 8

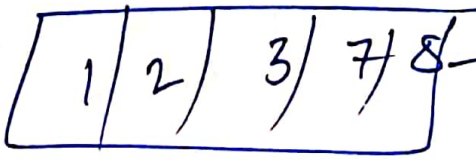
3 1 7 2 8



(3) 1 3 2 7 8



(4)



→ sorted array

* n = length of array

* no. of comparison = $(n-1) \rightarrow (n-i-1)$

Code :

```
int arr[] = {7, 8, 3, 1, 2};
int n = arr.length;
for (i = 0; i < (n-1); i++)          // (n-1)
{
    for (j = 0; j < (n-i-1); j++)    // (n-i-1)
    {
        if (arr[j] > arr[j+1])
        {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

$T_n = O(n^2)$

(20) Selection Sort

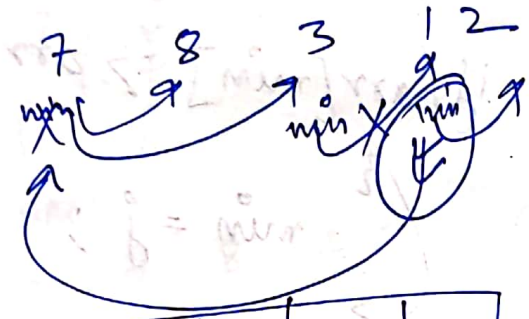
7	8	3	1	2

unsorted array

→ 1 swap per iteration

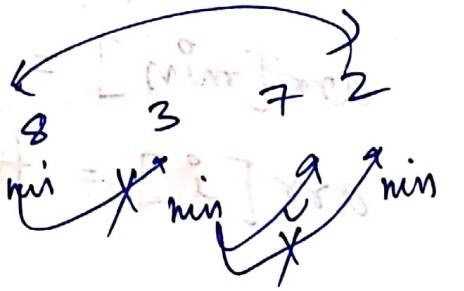
→ select any 1 as min / smallest elemt and then compare it with other array element

(1) $i = 0$



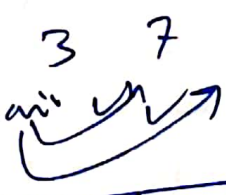
1	8	3	7	2

(2) $i = 1$



1	2	3	7	8

(3) $i = 2$



1	2	3	7	8

(4) $i = 3$

no swaps.
7 is min

1	2	3	7	8

Code

```
int arr[] = { 7, 8, 3, 1, 2 };
    int n = arr.length;
for(i=0 ; i<n ;i++)
  {
      int min = i ;
for(j=i+1; j<n ;j++)
  {
    if(arr[min] > arr[j])
      {
        min = j ;
      }
  }
int temp = arr[min];
arr[min] = arr[i];
arr[i] = temp;

}//for
```

min elem ko min assume & compare with rest

$T_n = O(n^2)$

(3) Insertation Sort

7	8	3	1	2

Divide Array into 2 parts

Sorted ↙ ↘ Unsorted

Sorted | Unsorted

(1) 7 | 8 3 1 2

place

compare 8 & 7 and place accordingly

(2) 7 8 | 3 1 2

3 7 8 | 1 2

(3) 3 7 8 | 1 2

(4) 1 3 7 8 |

1	2	3	7	8

sorted

Code

```
int arr[] = {7, 8, 3, 1, 2};
int n = arr.length;

for(int i=1; i<n; i++)
{
    int curr = arr[i];
    int j = i-1;

    while(j>=0 && curr < arr[j])
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = curr;
}
```

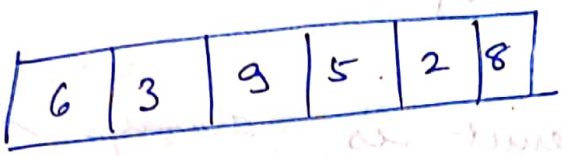
i=1, i=0 sorted part

* $T_n = O(n^2)$

Lecture -19

Quick Sort

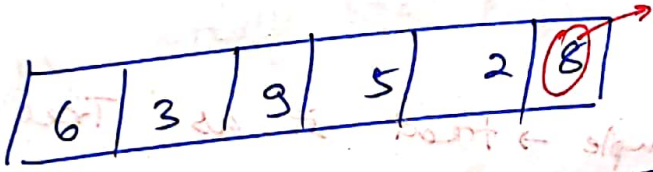
Unsorted array :



Pivot & Partition

To choose Pivot ?

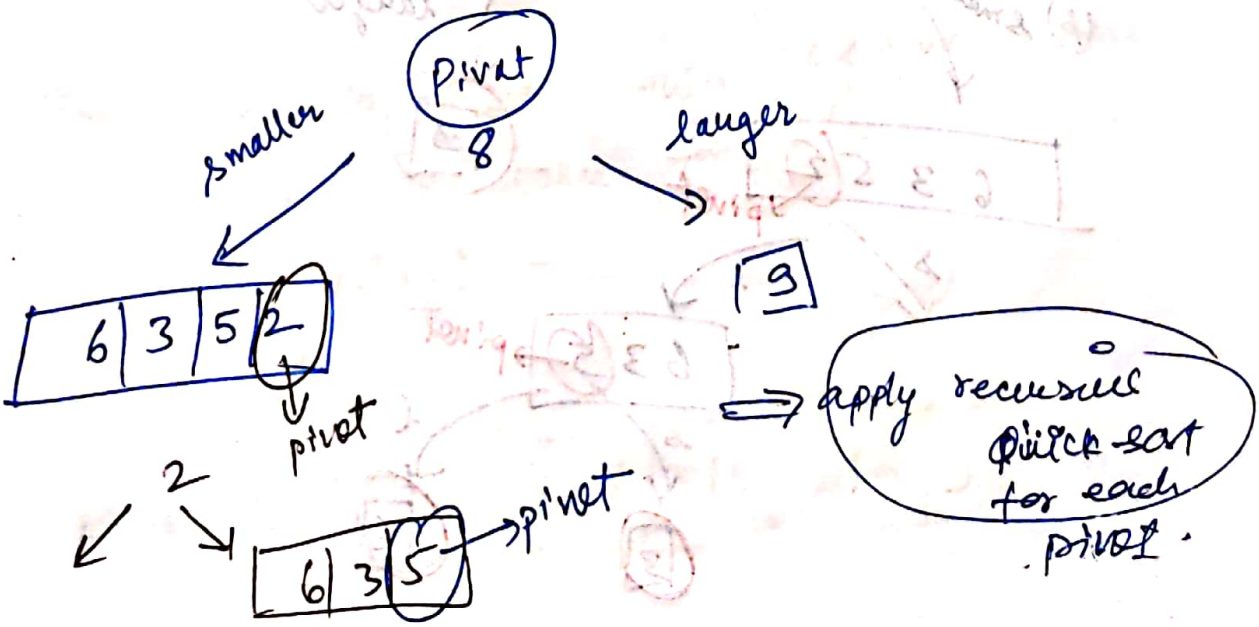
1) random
2) median
3) 1st elemt
4) last elemt



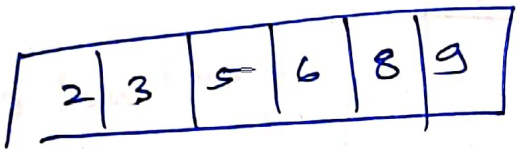
** Karan.

element < pivot ⇒ keep them at left

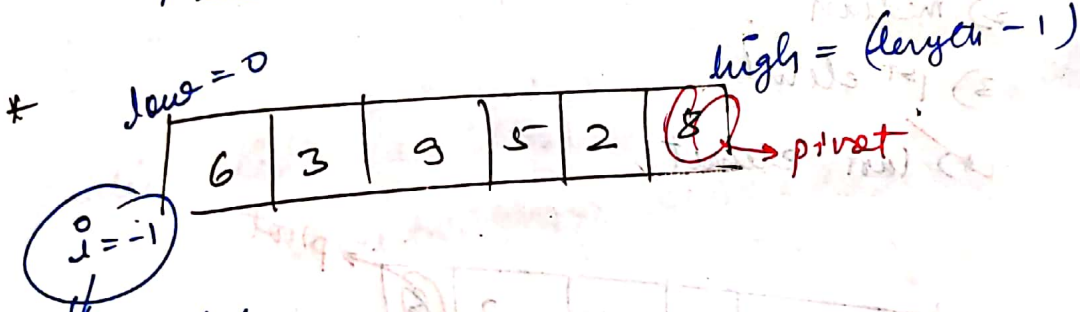
element > pivot ⇒ send them Right



* Single element is always sorted
* see from left to right in this recursive tree

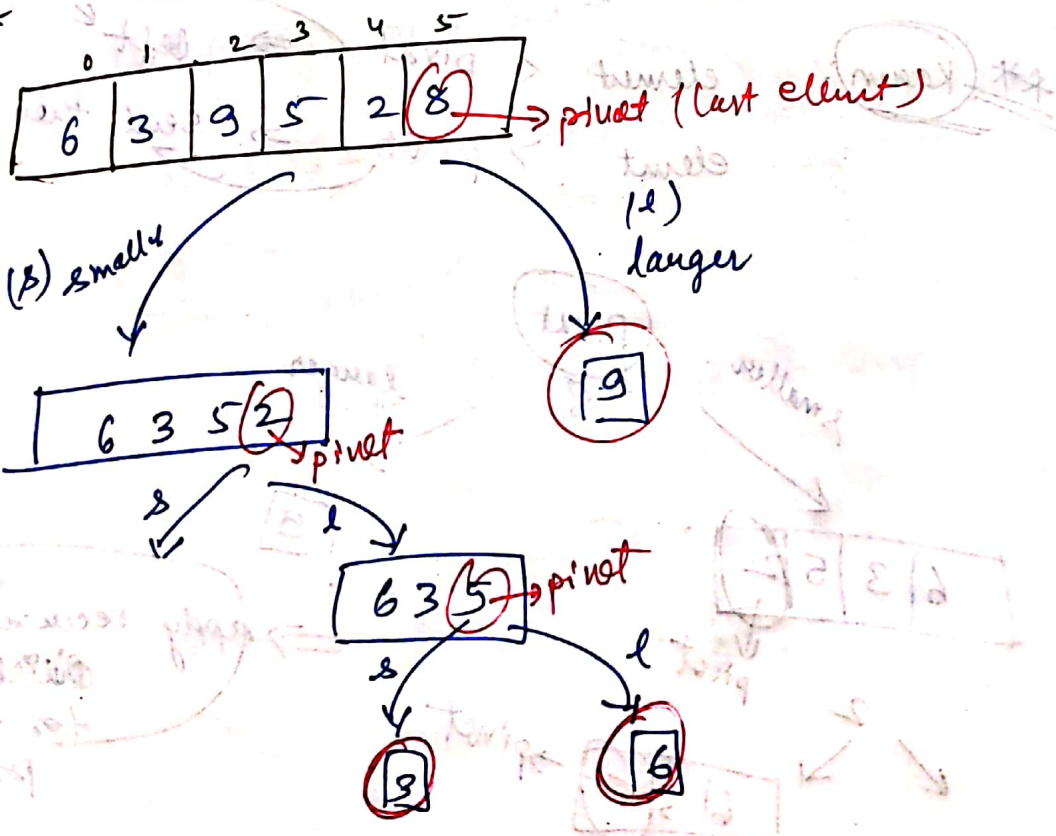


=> sorted Array



Sample → treat it as Tree

Note



⇒ Move from left to right in tree we get :

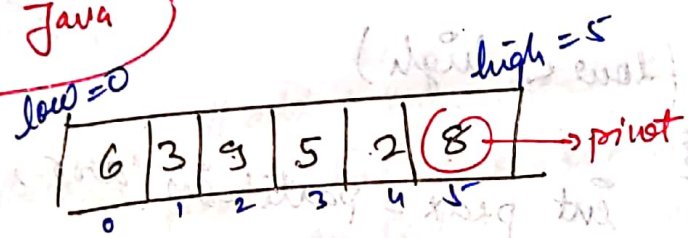
2, 3, 5, 6, 8, 9

⇒

0	1	2	3	4	5
2	3	5	6	8	9

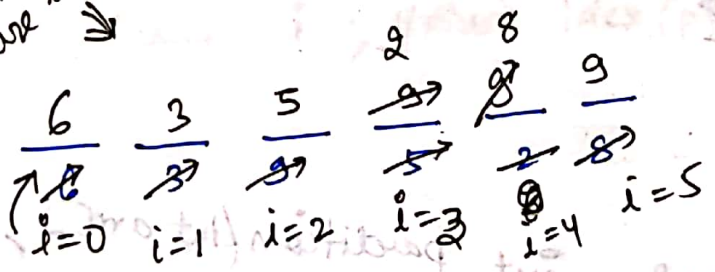
⇒ sorted array

Code in Java



i = -1 (no element less than pivot initially bzc pivot not selected)

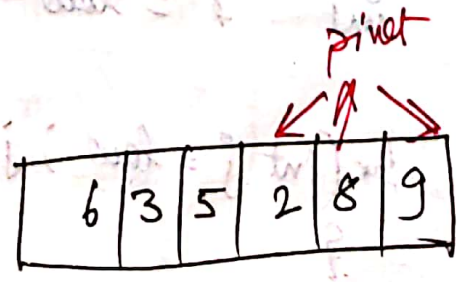
(Swap method kii use kara)

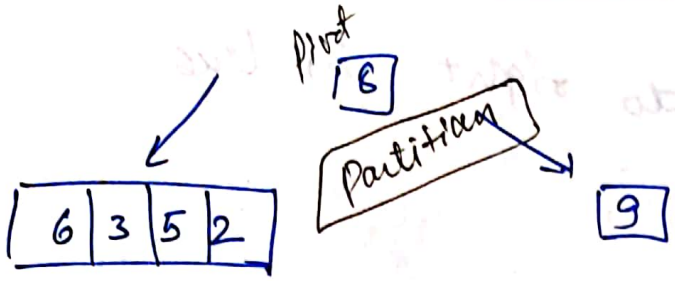


* Compare 6 with pivot, then (6 < pivot) only if follow karega then i will create space for '6'. (i++)

⇓

Now we get :





apply again same quick sort (recursive use karo)

Code

```
public static void quick(int arr[], int low, int high)
{
    if (low < high)
    {
        int pidx = partition(arr, low, high);
        quick(arr, low, pidx-1);   // less than pivot pat
        quick(arr, pidx+1, high);  // more than pivot pat
    }
}

public static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = low-1; //track for empty space

    for(int j = low; j< high ;j++)
    {
```

```
if(arr[j] < pivot) //element < pivot
{
    i++; //space create to store smaller element
    //swap → pehle jo us position par store element with abb jo small element mila.
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}//if
}//for
i++; //add 1 more space for pivot.
//swap pivot → b=c pivot end me hai abhi→to wapas sahi position pare lane ke liye
int temp = arr[i];
arr[i] = pivot;
arr[high] = temp;
return i; //pivot idex (position)
}//fn

Main()
{
    int arr[] = {6,3,9,5,2,8};
    int n = arr.length;
    quick(arr, 0, n-1); //
}
```

Time Complexity ⇒ Quicksort

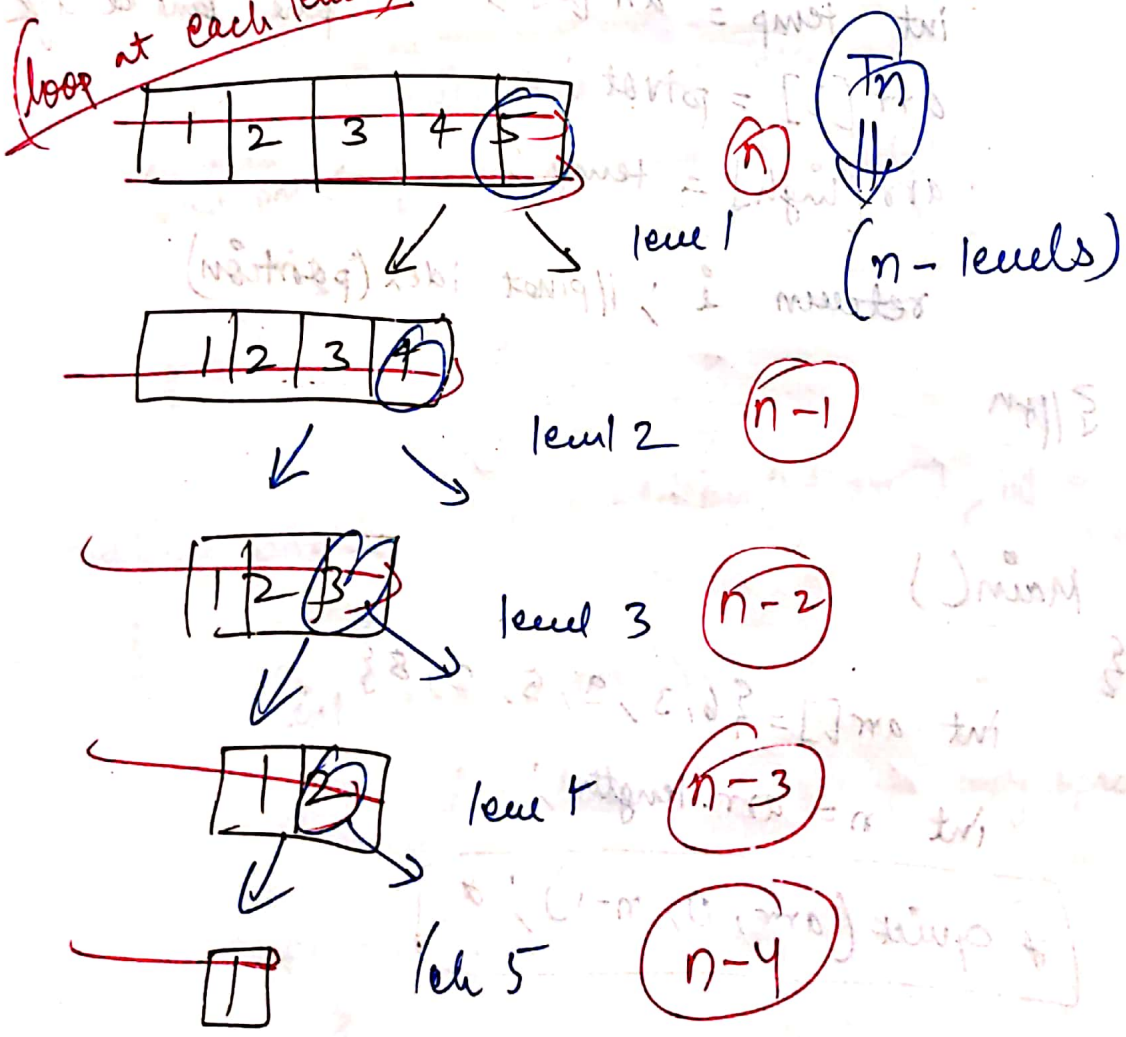
* **Worst Case** occurs when pivot is always the smallest or the largest element.

Worst Case : $T_n = O(n^2)$

Avg. Case : $T_n = (n \log n)$

→ when our array is completely sorted or ~~reversed~~ (either ascendig or descendig we sort)

(loop at each level)



⇒ Har level par loop chalane me

$T_n = O(n)$

⇒ $n + (n-1) + (n-2) + \cdots + 1$ (A.P)

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

$T_n = O(n^2)$ → $\therefore T_n = O(n^2)$

at worst case.

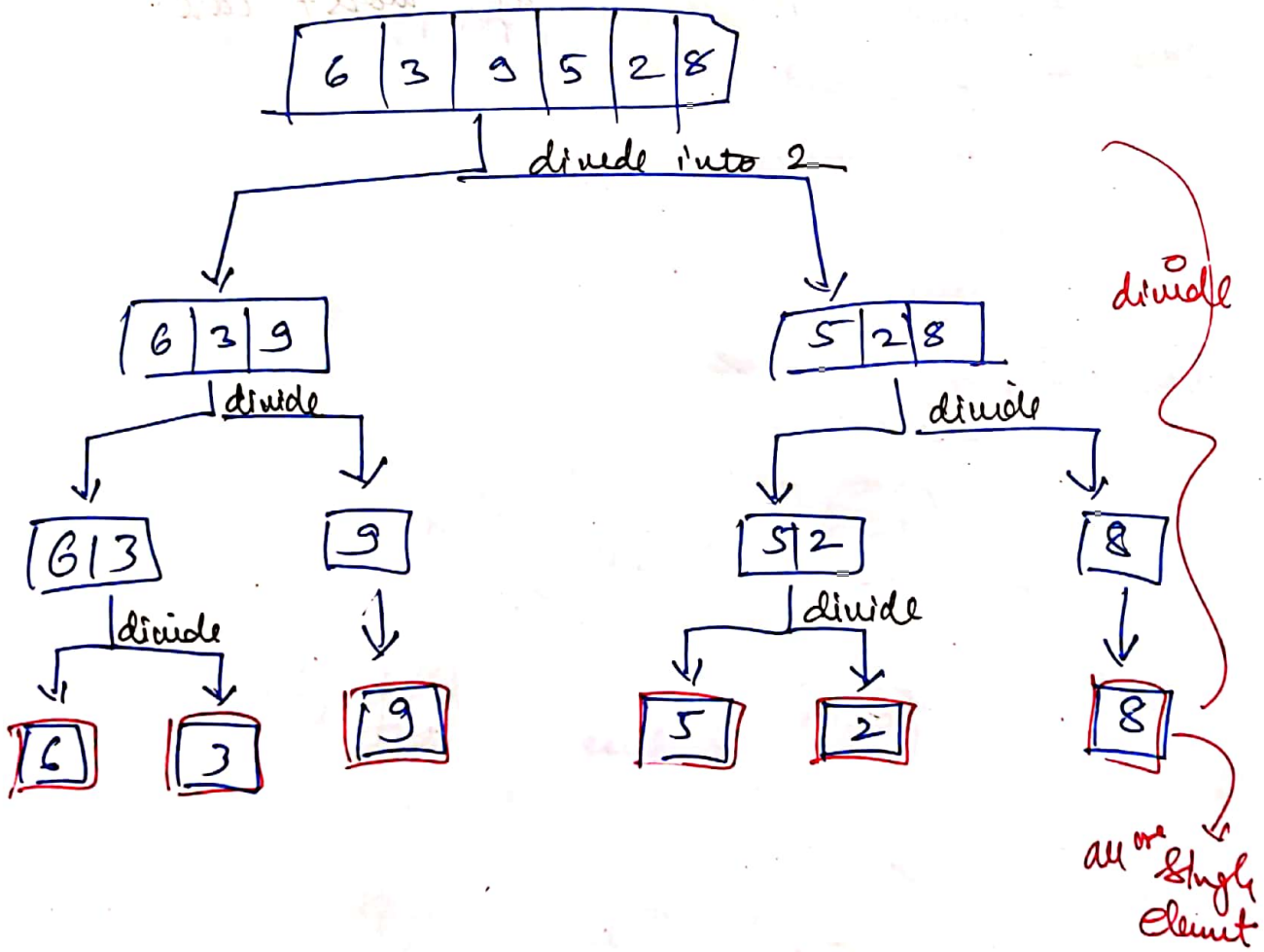
Lecture -20

Merge - Sort

Unsorted

6	3	9	5	2	8

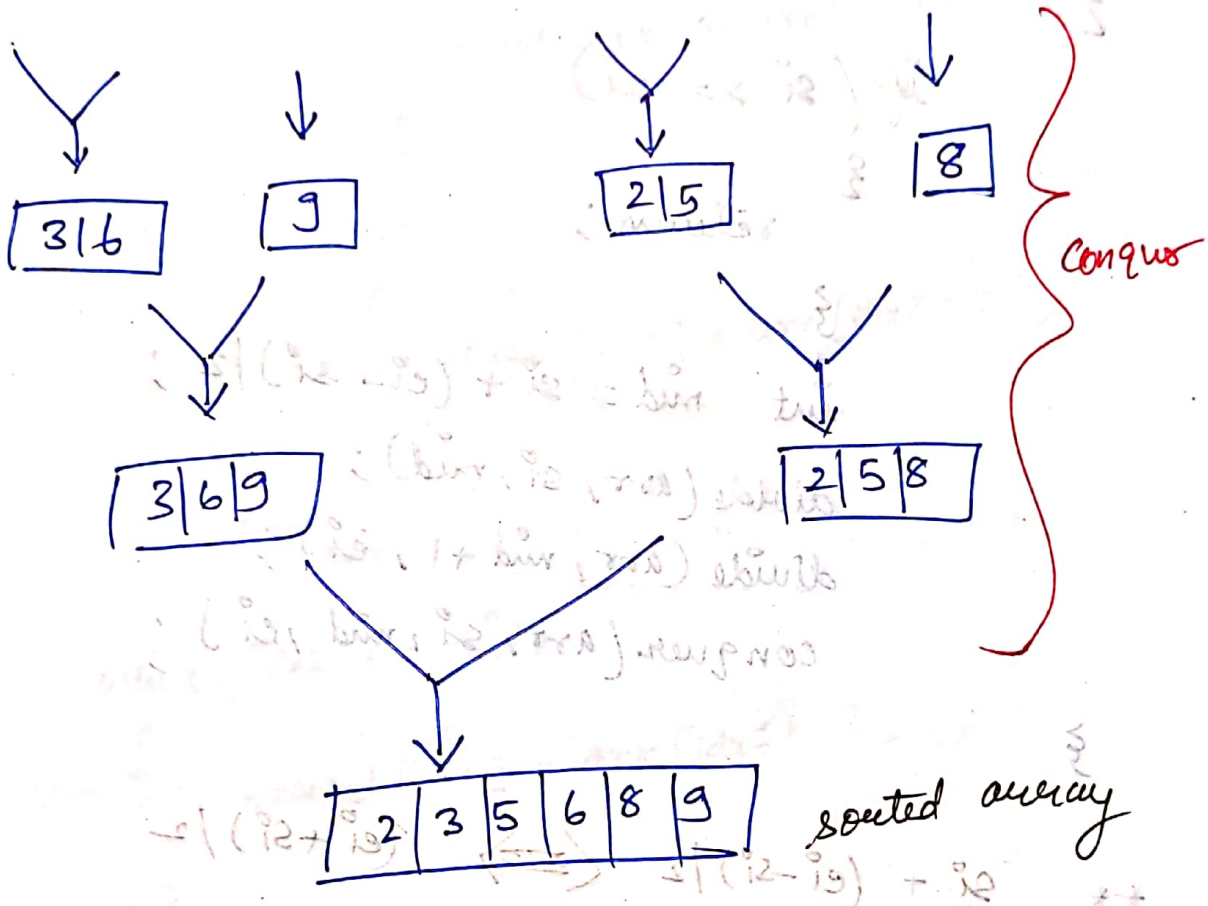
* Based on :
  Divide & Conquer Property

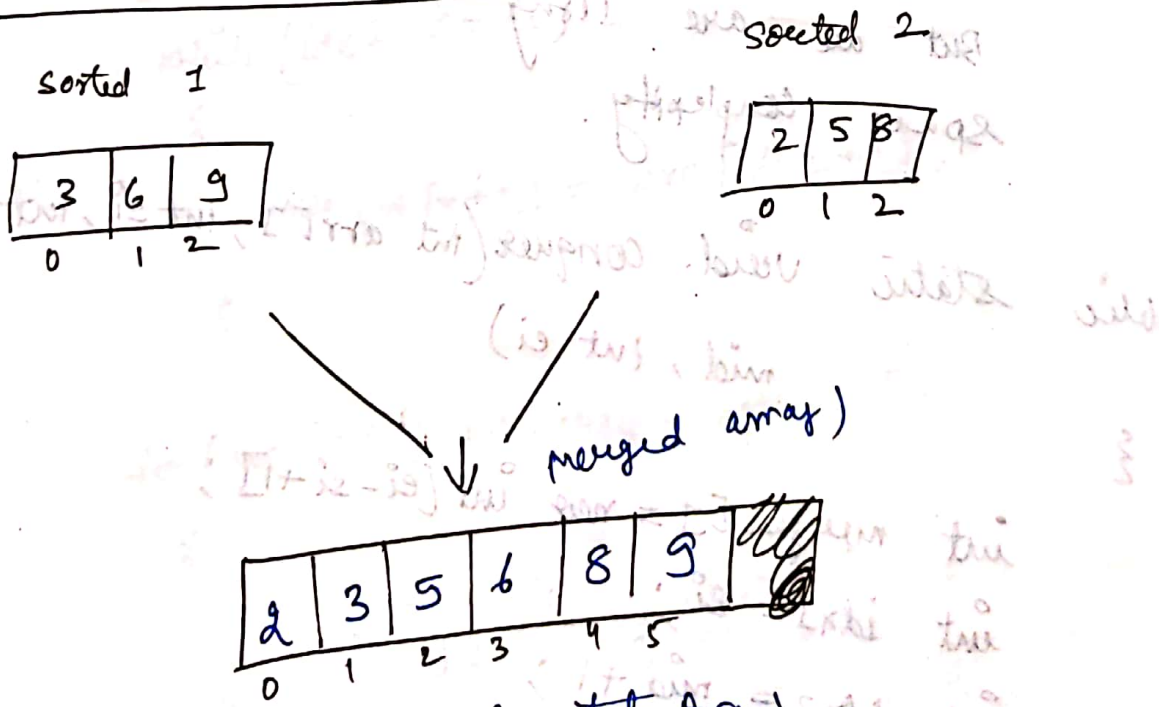


* Single element are already sorted.
* Now, Conquer these single elements and

while doing so make the comparison (i.e., sort them)

Now, extending prev. Tree structure :-





si = starting index
ei = ending index

Code

```
public static void divide(int arr[], int si, int ei)
{
    if (si >= ei)
    {
        return;
    }
    int mid = si + (ei - si)/2;
    divide(arr, si, mid);
    divide(arr, mid+1, ei);
    conquer(arr, si, mid, ei);
}
```

** $si + (ei - si)/2 \iff (ei + si)/2$

⇓

But we are using this to reduce space complexity.

```
public static void conquer(int arr[], int si, int
                           mid, int ei)
{
    int merged[] = new int[ei - si + 1];
    int idx1 = si;
    int idx2 = mid + 1;
    int x = 0;
```

```
while(idx1 <= mid && idx2 <= ei)
{
    if(arr[idx1] <= arr[idx2])
    {
        merged[x++] = arr[idx1++];
    }
    else
    {
        merged[x++] = arr[idx2++];
    }
} // while

while(idx1 <= mid)
{
    merged[x++] = arr[idx1++];
}

while(idx2 <= ei)
{
    merged[x++] = arr[idx2++];
}

for(int i=0; i<merged.length; i++)
{
    arr[j] = merged[i];
} // for

} // fn
```

```
main()
{
    int arr[] = {6, 3, 9, 5, 2, 8};
    int n = arr.length;
    divide(arr, 0, n-1);
}//main
```

$$T_n = O(n \log n)$$

* $\log_2 n$ (divide)
* n (conquer)