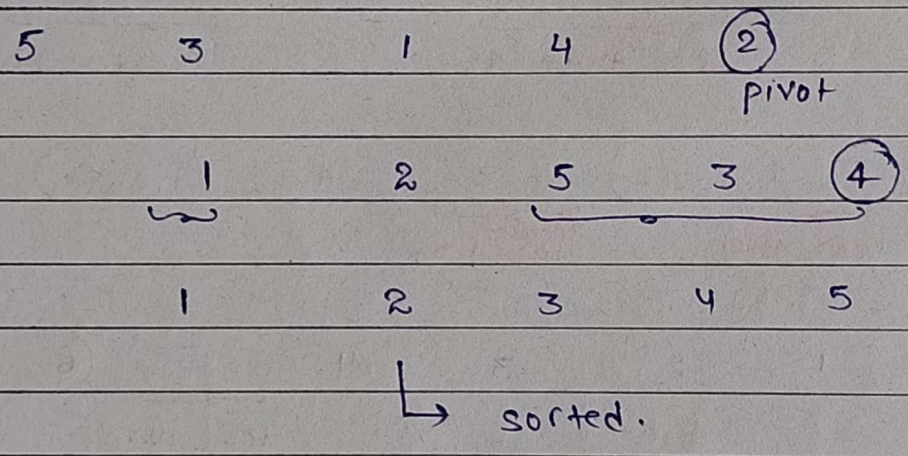
Page No.:
Date: / /

Lecture - 27

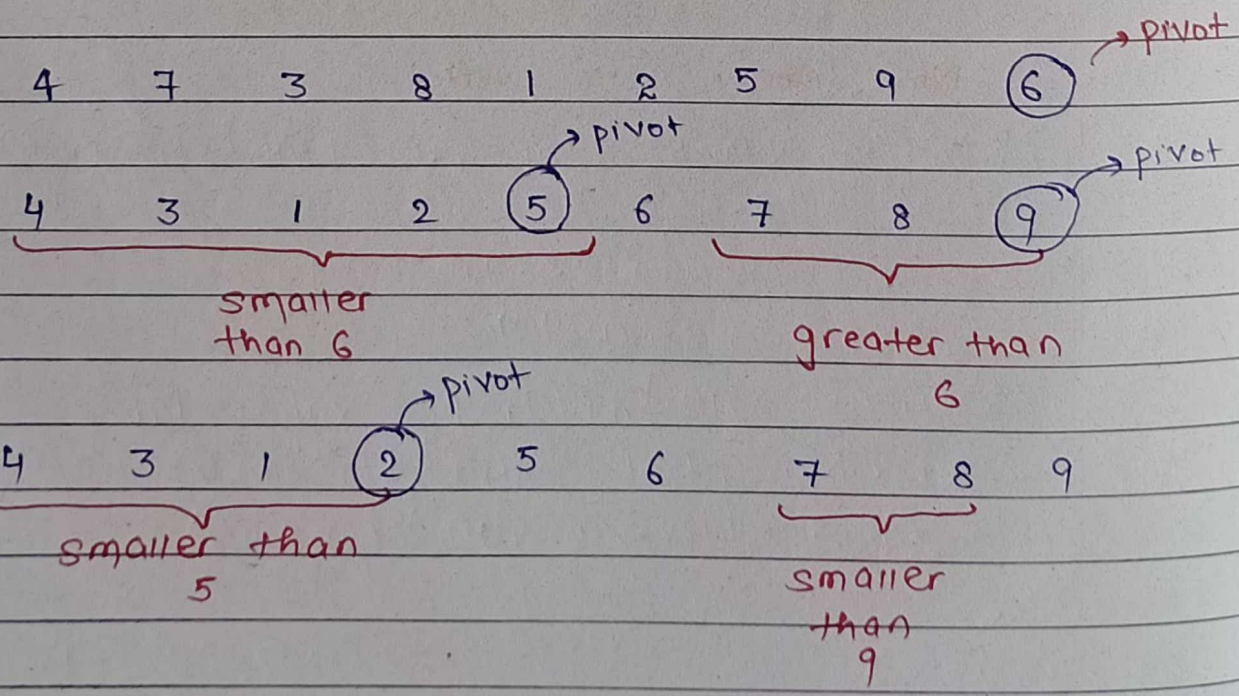
Quick sort

5 3 1 2 4

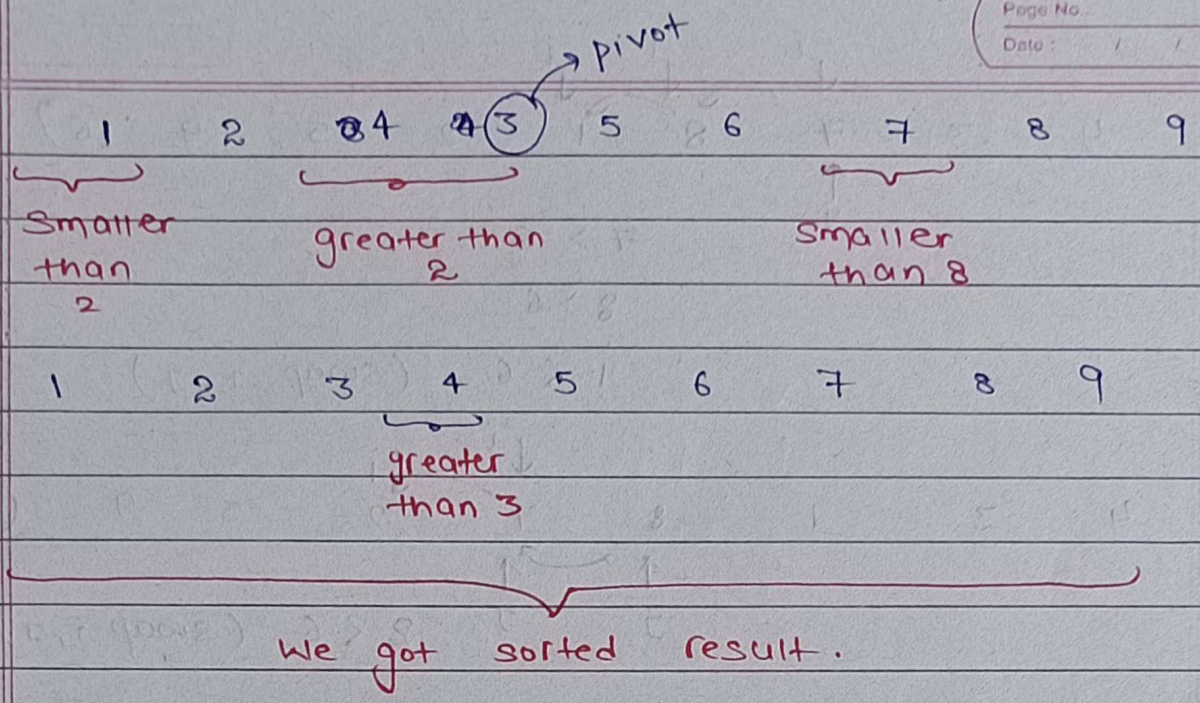
→ Choose a pivot element
→ start from first smaller element shift to left side and greater element shift to right side.



Ex:



Teacher's Signature.....

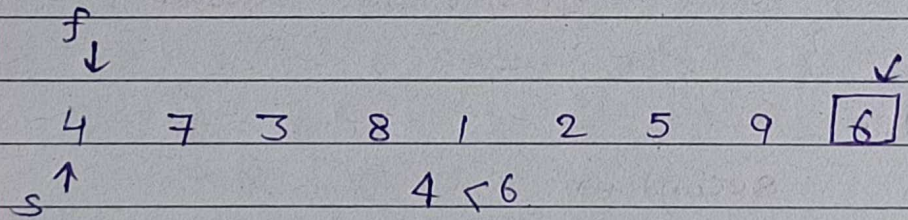


→ Find pivot Element

→ put pivot element in its Correct position.

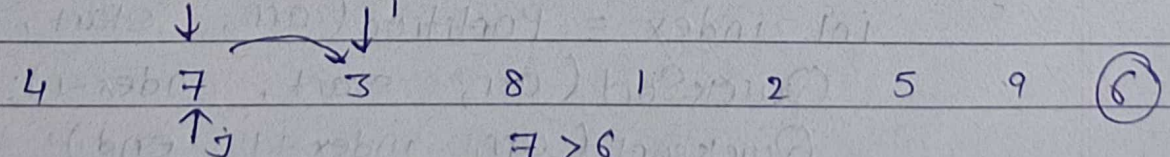
→ Visit left part of array and visit right part of array.

→ Repeat 3 step until Array size become 1.



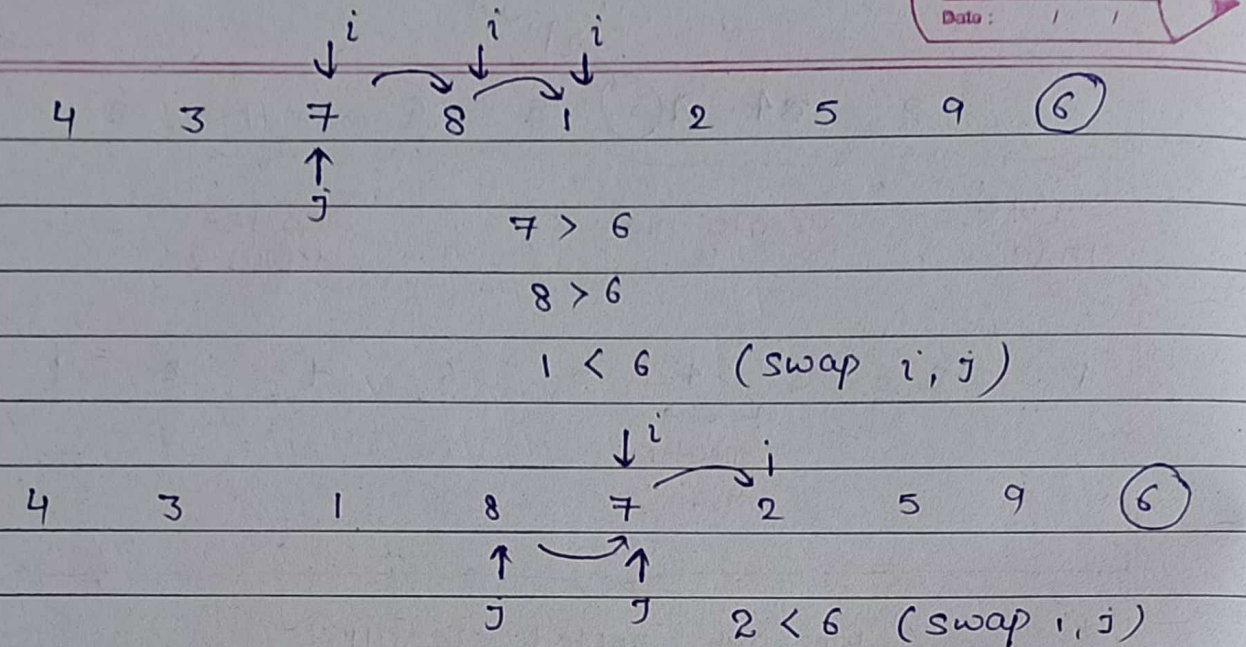
1st pointer : value ko fill kaha karna hai uska position batayega.

2nd pointer : Array ko traverse karega.



3 < 6 swap (i, j)

Teacher's Signature.....




```
if ( arr[second] <= pivot ) {
        swap ( arr[first], arr[second] );
            first++;   second++;
    }
else
    second++;
```

→

```
QuickSort ( int[] arr, int start, int end ) {

    if ( start >= end )
        return;
    int index = Partition ( arr, start, end );
    QuickSort ( arr, start, index-1 );
    Quicksort ( arr, index+1, end );

}
```

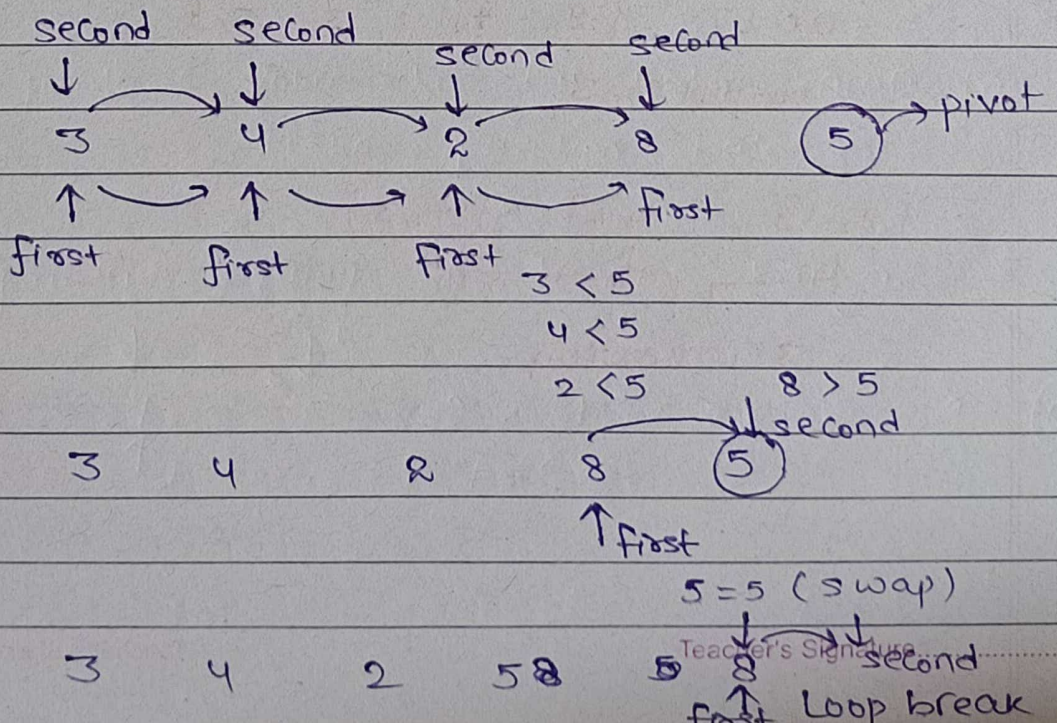
Teacher's Signature.....

Page No.
Date : / /

Array ko Break tab tak karna hai.

1) ya to Array khatm ho Jaye.
2) ya Array ka size 1 ho Jaye.

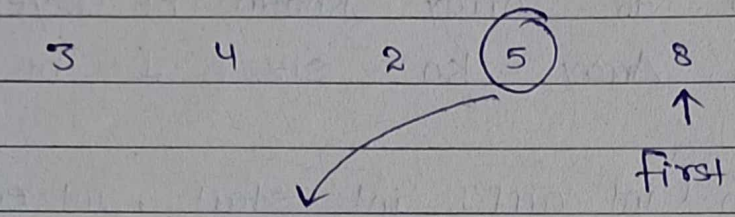
```
int partition (int arr[], int start , int end) {

    int pos = start;
    int first = start;
    int second = start;

    While ( second <= end ) {
        if (arr[end] >= arr[second]) {
            int temp = arr[first];
            arr[first] = arr[second];
            arr[second] = temp;
        } first++;    second++;
    } else
        second++;
}
return first - 1;
}
```



Teacher's Signature

Loop break

second ↓



return first − 1;

Time Complexity :

$O(N \log N)$

If array sorted : $O(N^2)$

Space Complexity :

$O(N)$ → of array sorted

$O(\log N)$

* Inversion of array :

N = 5,

arr[] = {2, 4, 1, 3, 5}

output = 3

{2, 1}, {4, 1}, {4, 3}

3 inversion

Page No.:
Date: / /

Check for every element, How many smaller element are present after element.

Count and return.

2 3 1 4 5

Condition

$a[i] > a[j]$ } inversion
$i < j$

2 4 1 3 5

for 2 : 1

1 element is smaller

Count = 1

for 3 : 1

1 element is smaller

Count = 2

for 1 : 0

for 4 : 1

Count = 3

for 5 : 0

return 3

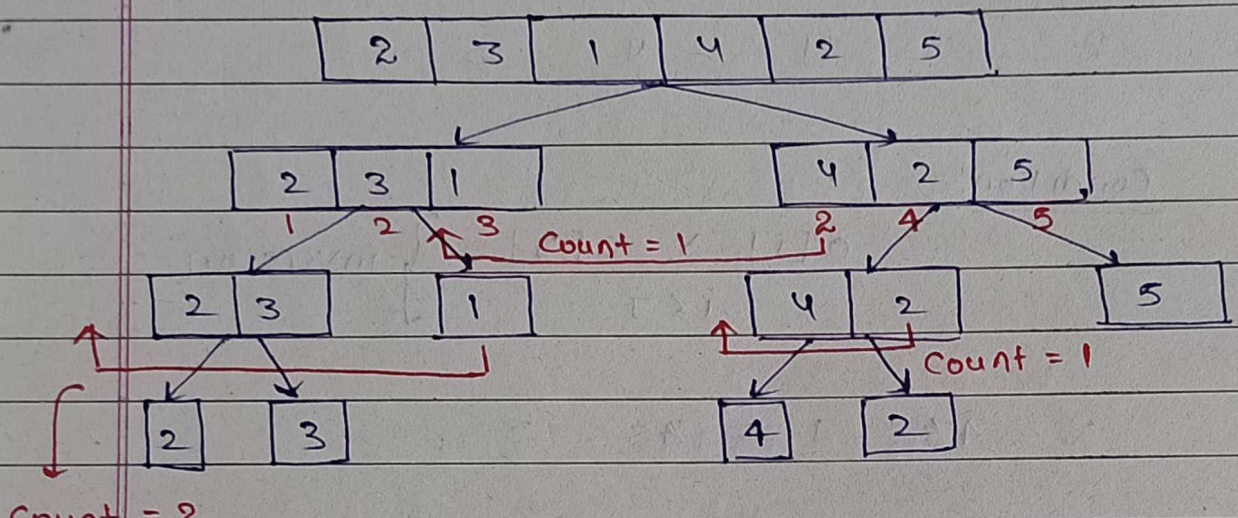
By this, We take $O(N^2)$

But We have to solve in $O(N \log N)$

Teacher's Signature.....

Merge sort (NlogN) ↓ Try By this

Quick sort (Nlogn)



Total = 2 + 1 + 1
= 4

Merge sort
↳ and take a variable Count and increment Whenever we shift any element.

Code :-

```
Class Solution {
    static long ans;
Static Void merge (long arr[], int l, int mid, int r) {
    long []c = new long [r-l+1];
    int i = l, j = mid+1, k = 0;
While (i <= mid && j <= r) {
    if (arr[i] > arr[j]) {
        ans += mid - i + 1;
        c[k++] = arr[j++];
    }
```

Teacher's Signature.....

```
        else {
            c[k++] = arr[i++];
        }
    }
    while (i <= mid) {
        c[k++] = arr[i++];
    }
    while (j <= r) {
        c[k++] = arr[j++]; }
    k = 0, mid = l;
    while (mid <= r)
        arr[mid++] = c[k++];
}
static void mergeSort (long arr[], int l, int r){
        if (l = r) return;
        int mid = (l+r)/2;
    mergeSort (arr, l, mid);
    mergeSort (arr, mid+1, r);
    merge (arr, l, mid, r);
}
static long inversionCount (long arr[], long N) {
        ans = 0;
    mergeSort (arr, 0, (int)(N-1));
    return ans;
}
}
```

Teacher's Signature.....

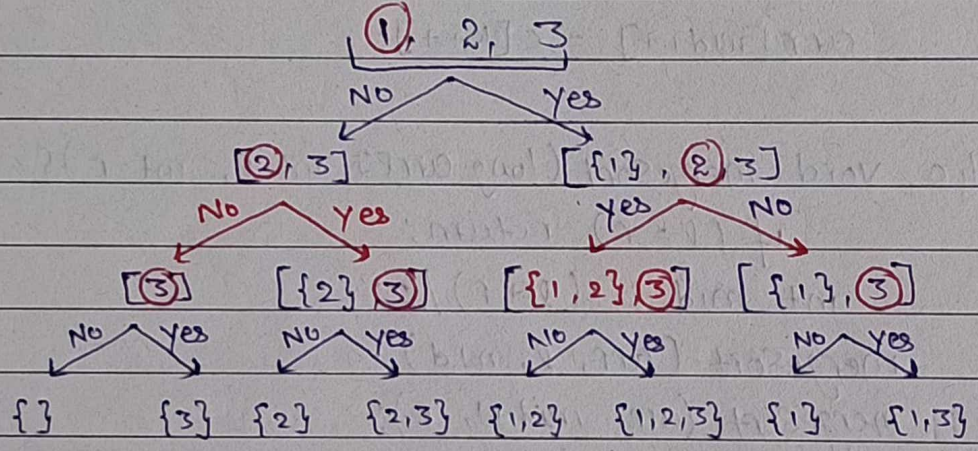
Page No.:
Date: / /

* Coins : 1, 2, 3

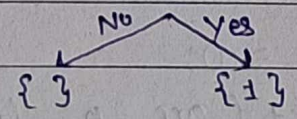
You have select as you want
either 0 Coin, 1 Coin, 2 Coin, 3 Coin at one time.

possibility : { }, {1}, {2}, {3}, {1,2}, {1,3},
{2,3}, {1,2,3}

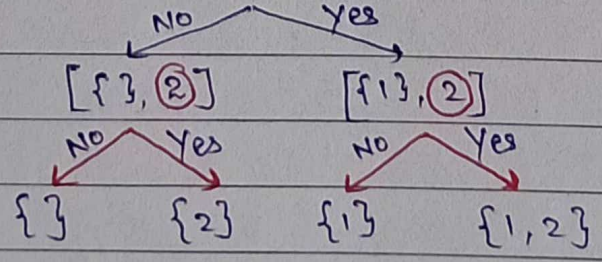
3 Coins → 2^3 possibility
= 8



* Coin = 1



* Coins = 1, 2



Teacher's Signature.....

*

6	3	1	2	4	8

Sum = 13

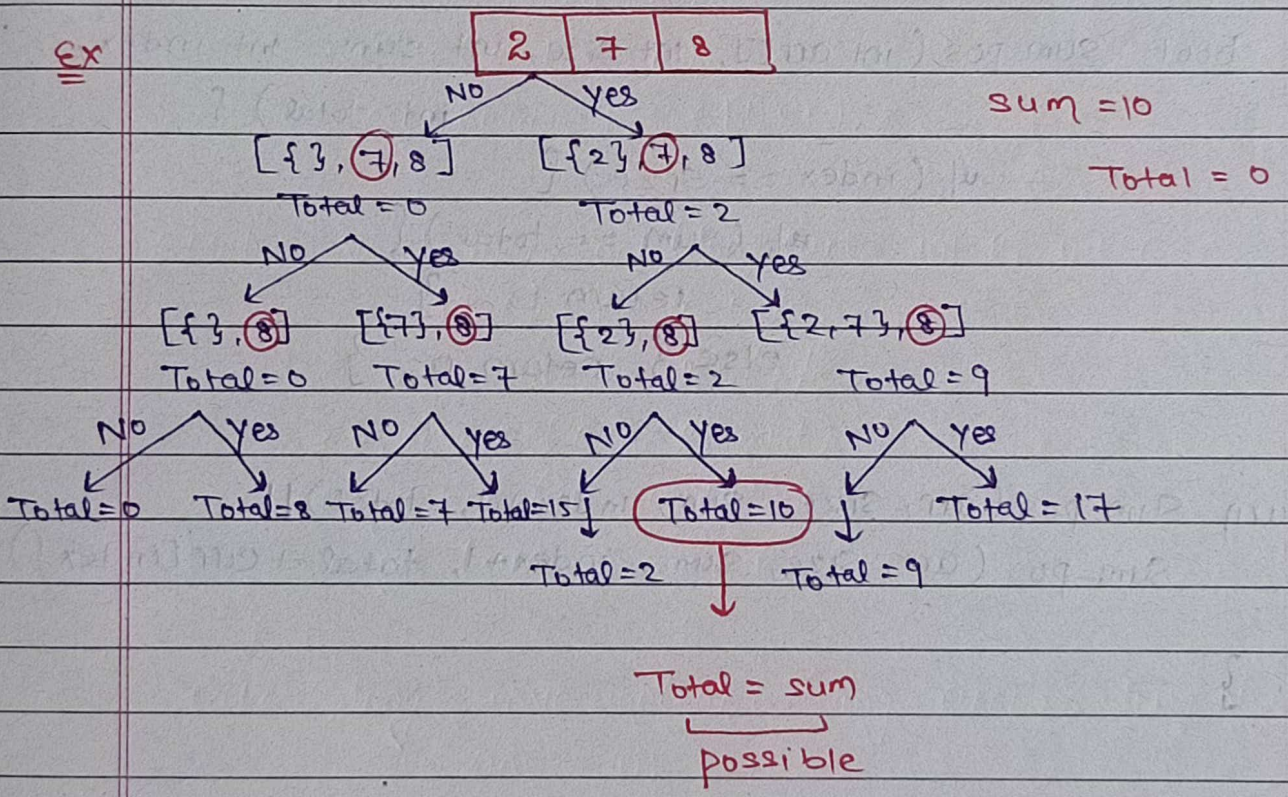
Choose any element any number of time. can we find sum of element which is equal to sum.

When we make tree:

Total possible case

$2^6 = 64.$

Ex



Total = sum possible

Time Complexity : 2^N.

```
int main () {
    int arr[] = { 2, 5, 1, 6, 7, 11 };
    int sum = 20;
    int total = 0;

    bool ans = sum_pos ( arr, 6, sum, 0, total );
        cout << ans;
    return 0;
}

bool sum_pos ( int arr[], int size, int sum, int index,
                                        int total ) {
        if ( index == size ) {
            if ( sum == total ) {
                return 1; }
            else { return 0; }

return sum_pos ( arr, size, sum, index+1, total ) ||
    sum_pos ( arr, size, sum, index+1, total + arr [index] );

}
```

Teacher's Signature.....