Agenda (TnS)

→ Searching

→ Linear Search

→ Binary Search

questions + portal + Extra.

→ Sorting.

→ Bubble

→ Selection

→ Insertion

→ Merge

→ Quick

→ Count

→ Radix.

Complexity Analysis.

Expectation → searching(arr, lo, hi, data) → True / false.

faith → searching(arr, lo, mid-1, data)
searching(arr, mid+1, hi, data)

mid = $lo + \frac{hi - lo}{2}$

$lo + \frac{(hi - lo)}{2}$

$\Rightarrow lo + \frac{hi}{2} - \frac{lo}{2}$

$= \frac{lo}{2} + \frac{hi}{2}$

$= \frac{lo + hi}{2}$

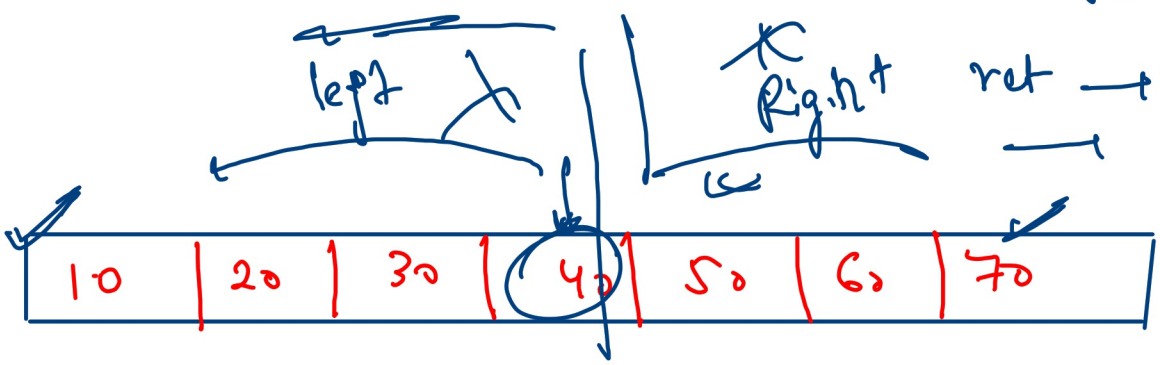
data. if(arr[mid] == data) return;

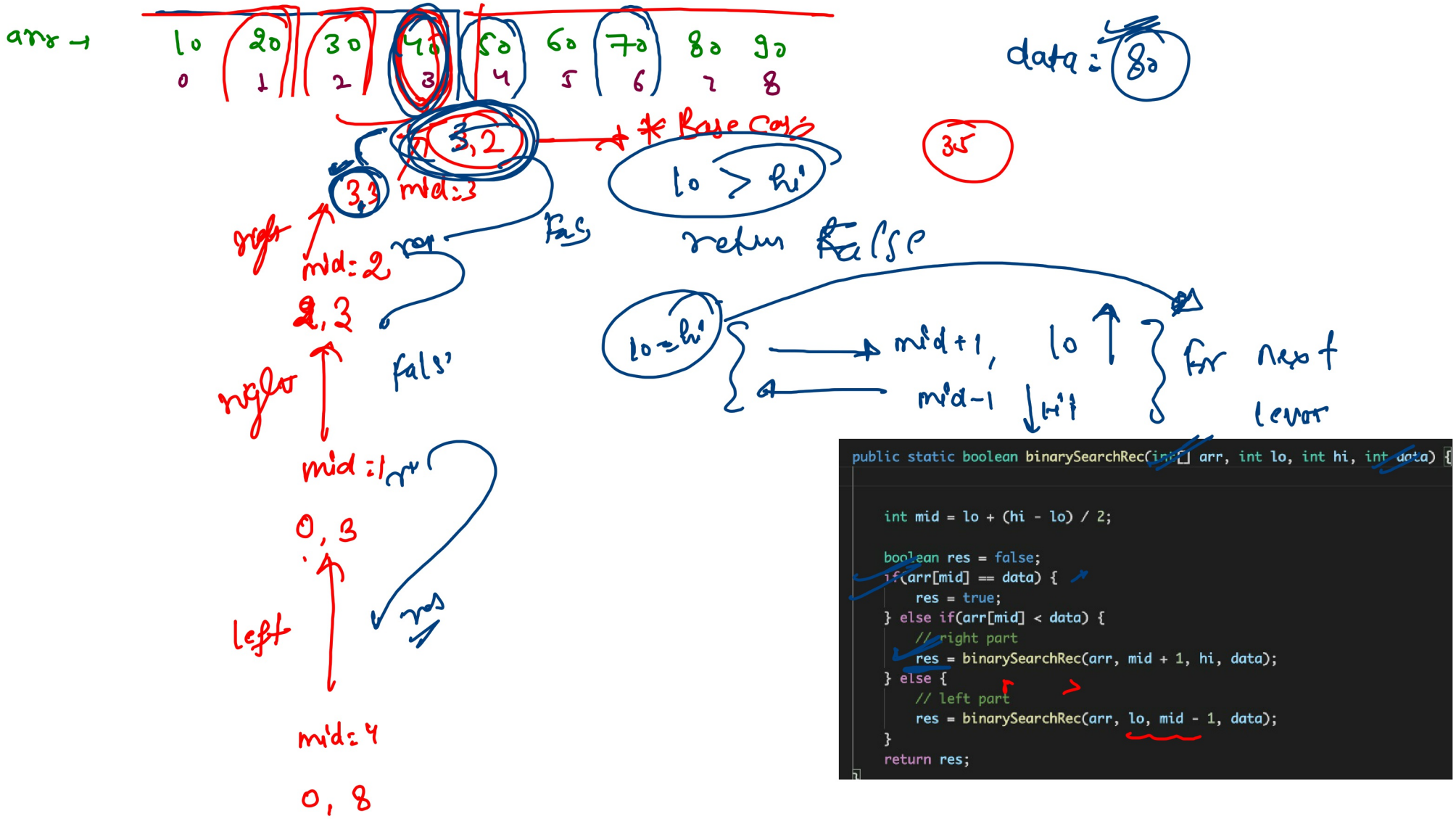
Merging →

left

Right

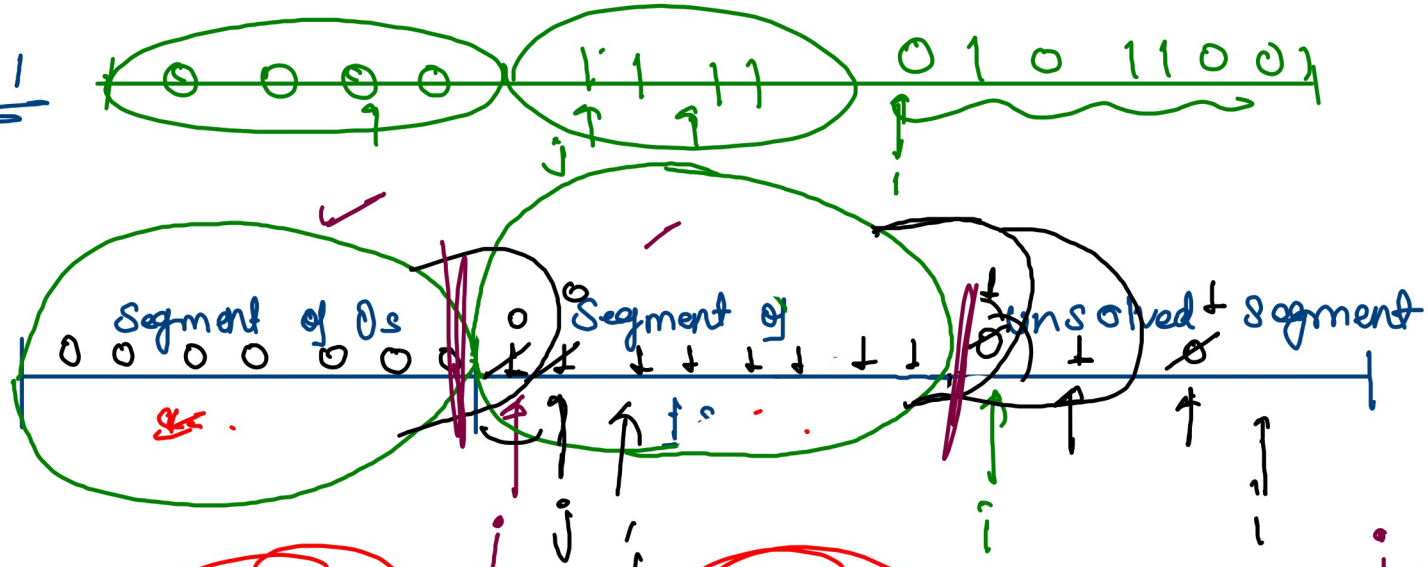
ret → left / Right






```java
public static boolean binarySearchRec(int[] arr, int lo, int hi, int data) {

    int mid = lo + (hi - lo) / 2;

    boolean res = false;
    if(arr[mid] == data) {
        res = true;
    } else if(arr[mid] < data) {
        // right part
        res = binarySearchRec(arr, mid + 1, hi, data);
    } else {
        // left part
        res = binarySearchRec(arr, lo, mid - 1, data);
    }
    return res;
}
```

Sort 01

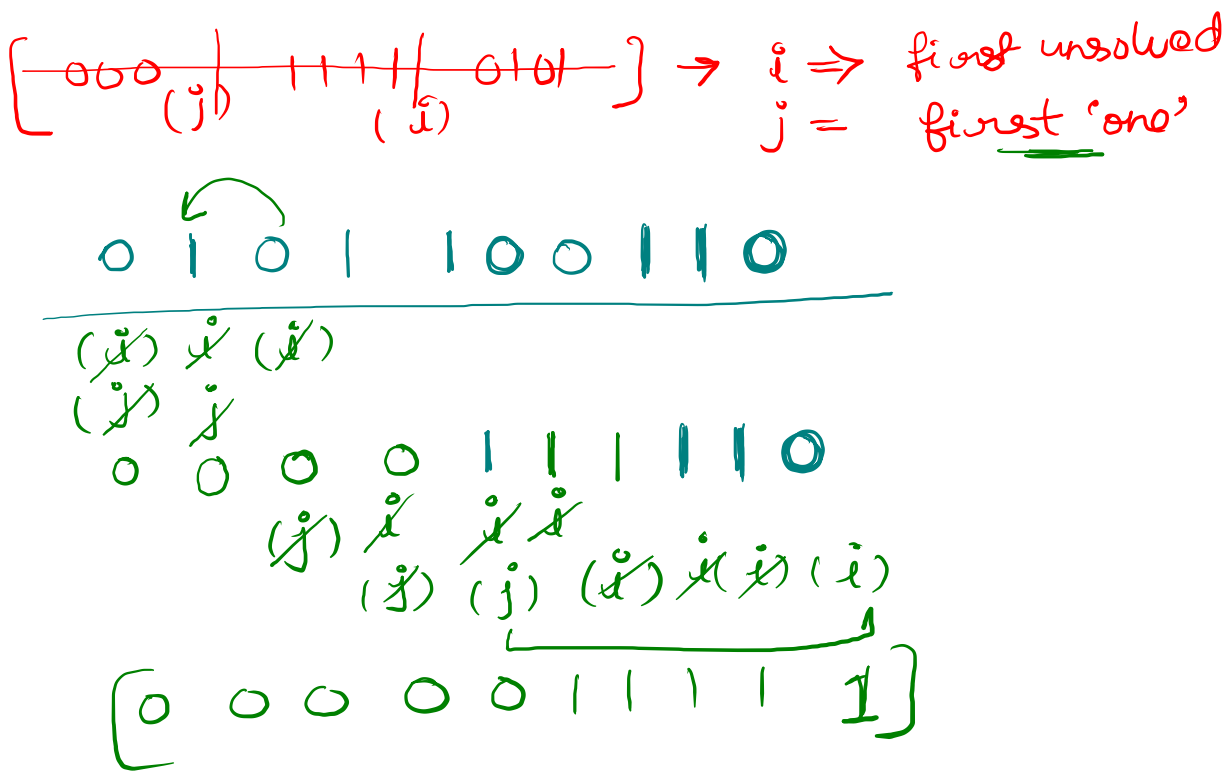


i → first unsolved

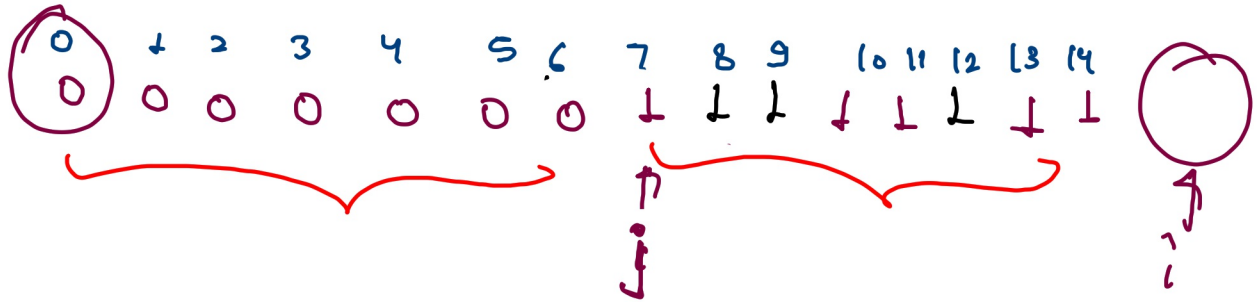
j → first 1

prime (odd)	Not (Even)
arr[i] = 1	arr[i] = 0
// Increment in segment of 1.	swap(arr, i, j);
i++;	i++;
	j++;

Segregate Parity




```
while( i<arr.len)
{
  if( arr[i] == 0)
  { swap(arr,i,j)
    i++; j++;
  }
  else
  {
    i++;
  }
}
```



i → First Unsolved i

j → first 1

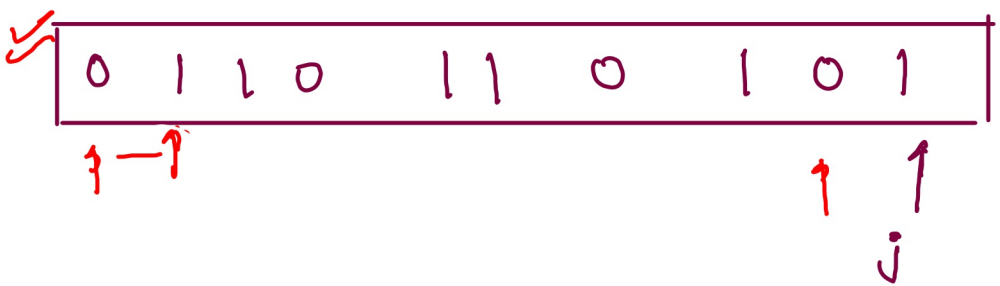
arr[i] == 0
i++

arr[i] == 1
swap(i, j)
j--;

```
if(arr[i] == 1){
    i++; //Increment in
         segment of 1
} else {
    swap(arr, i, j);
    i++;
    j++;
}
```

i → first unsolved

j = last unsolved



Segregate Odd Even → Odd - Even but we can't maintain the order of even element.

order of odd & closest nhi change

sequence of odd Element does not change

4 7 6 9 3 8 10 11 14 16 15 20

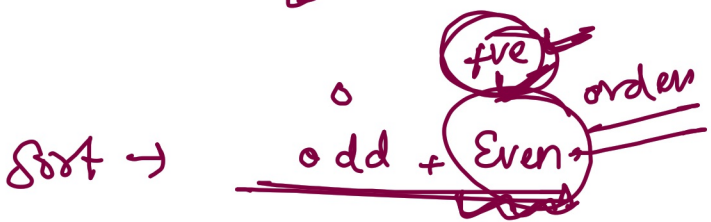
order maintain }

→ 7 9 3 11 13 4 6 8 10 14 16 20

odd Even

E_1 O_1 E_2 E_3 O_2 O_3 E_4 O_4

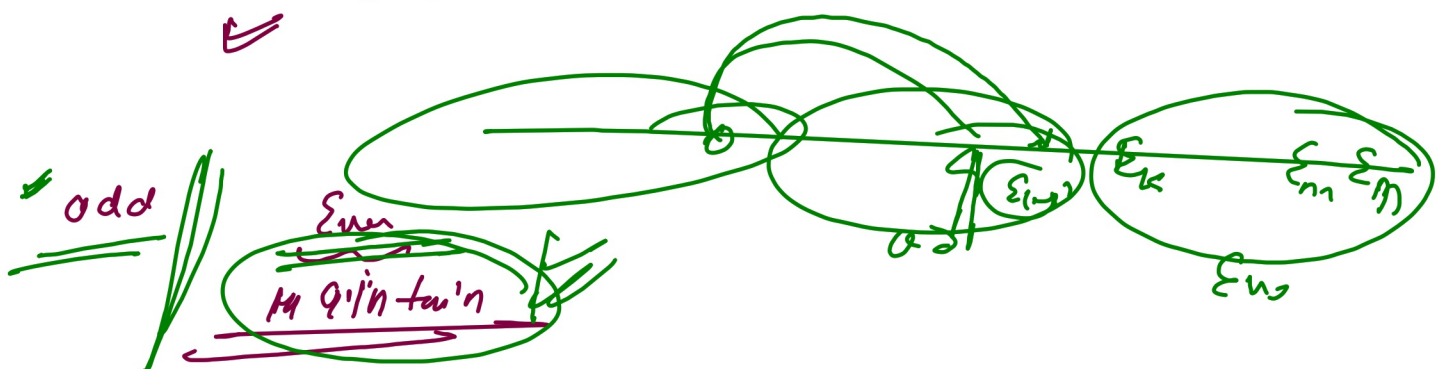
10 1 14 8 5 3 18 7



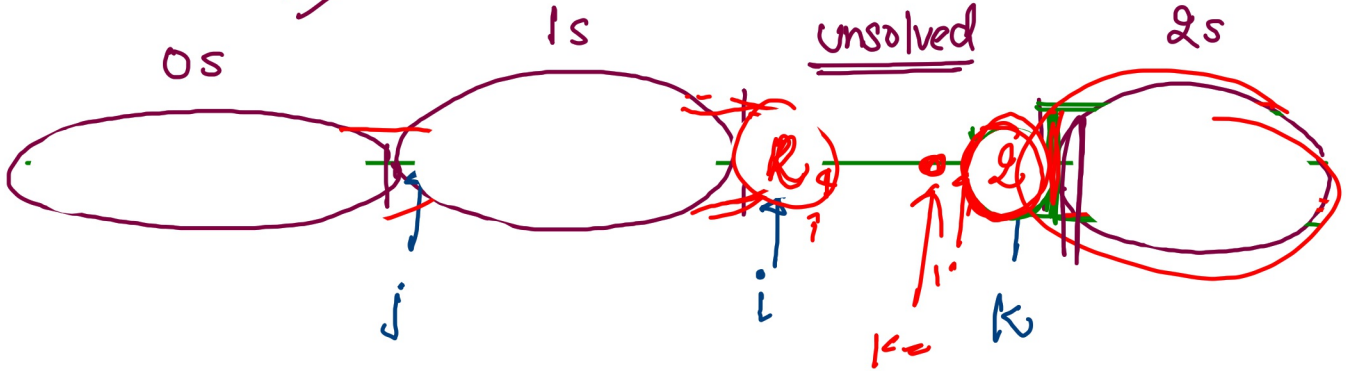
Even may be arrange in Random order

1 5 3 7 | { 10 14 8 18 }

order Maintain



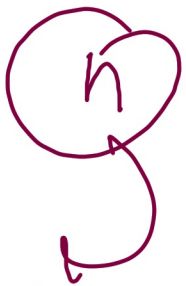
Sort 0 1 2 (Dutch Flag)



i ⇒ first unsolved
j ⇒ First 1
k ⇒ Last unsolved

} decision factor → 'i'

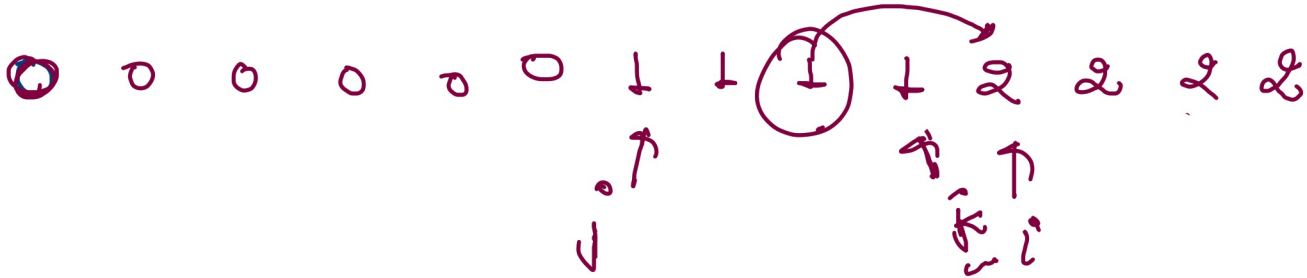
Algorithm



Quick Sort

arr[i] == 1	arr[i] == 0	arr[i] == 2

i++;
 	
// Increment the Region 1
	Swap(i,j):
	i++;
	j++;
		Swap(i,k):→ (arr[k] == 2)
		k--;



i = 0

j = 0

k = arr.length - 1

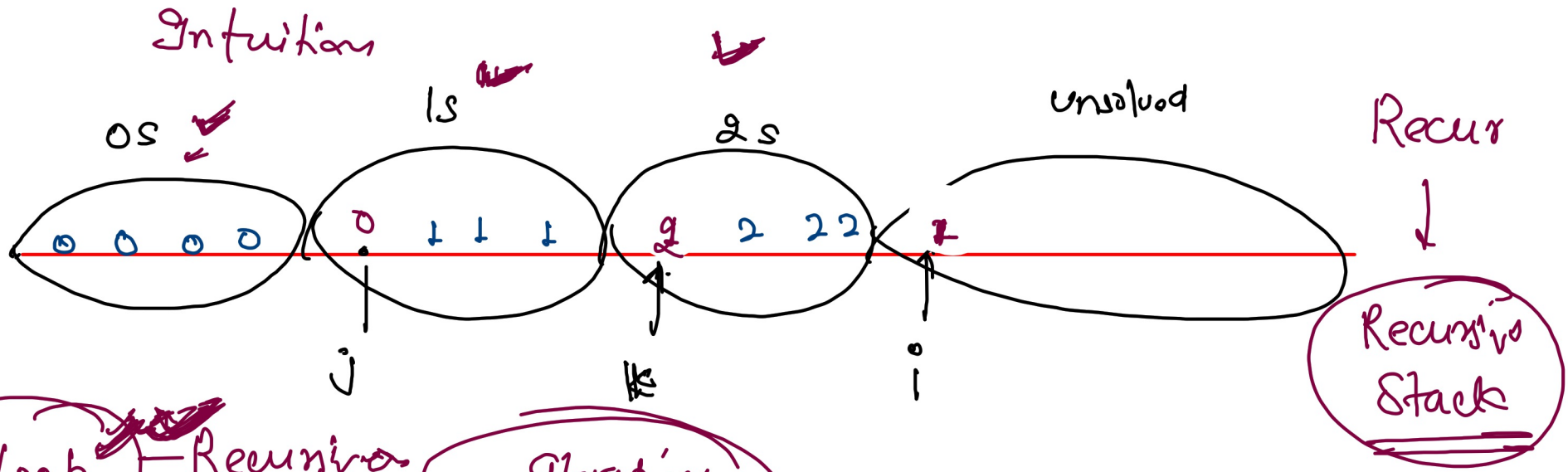
k > i → Stopping.

i == k

Both are unvalued

arr[i] == 0	arr[i] == 1	arr[i] == 2.

swap(i, j);
i++;
j++;
	i++;	swap(i, k);
		k--;



arr[i] == 0

swap(i, j)

swap(i, k)

i++;
j++;
k++;

arr[i] == 1

swap(i, k);

i++;

k++.

arr[i] == 2.

i++;

Solve Polynomial

$f(x, n) \rightarrow 1 \cdot x^n + 2 \cdot x^{n-1} + 3 \cdot x^{n-2} + \ldots \ldots + n \cdot x^1$

$f(x, n) \rightarrow n \cdot x^1 + (n-1) \cdot x^2 + (n-2) \cdot x^3 + \ldots \ldots + 1 \cdot x^n$

$n \cdot x$

$n--$

$x *= x$

$O(1)$

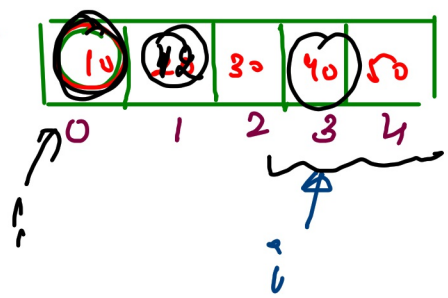
n times

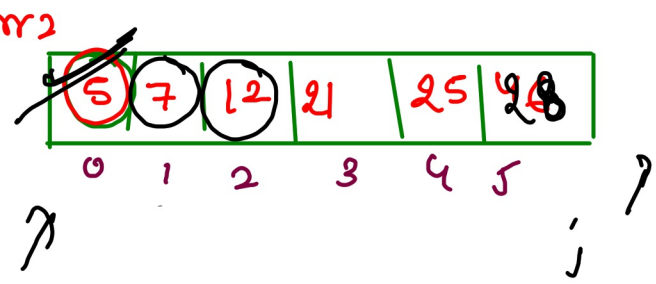
$O(n)$

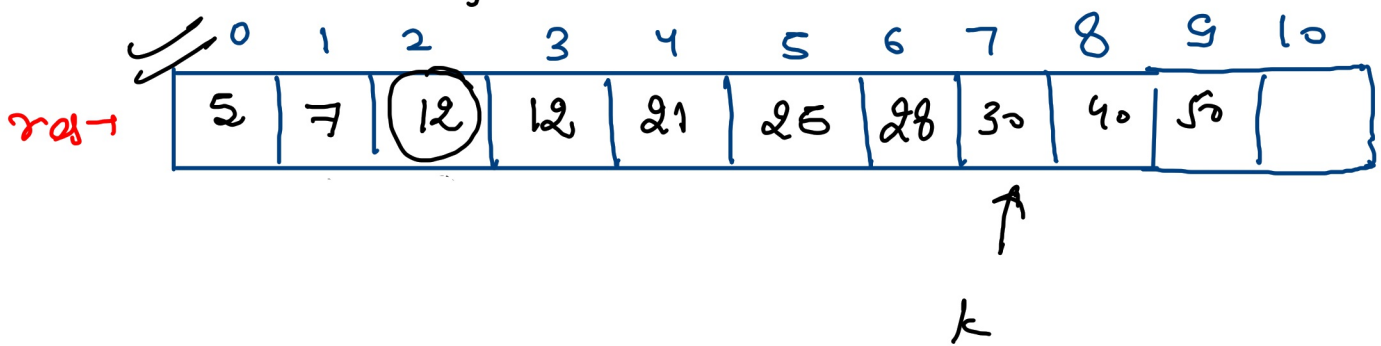
$n(\log n)$

Merge Two Sorted Arrays

```
if(arr1[i] > arr2[j]){
    res[k] = arr2[j]
    j++;
} else {
    res[k] = arr1[i];
    i++;
}
k++
```







Largest subarray having eq no. of 0's and 1's

```
int findSubArray(int arr[], int n)
{
    int sum = 0;
    int maxsize = -1, startindex = 0;
    int endindex = 0;

    // Pick a starting point as i

    for (int i = 0; i < n - 1; i++) {
        sum = (arr[i] == 0) ? -1 : 1;

        // Consider all subarrays starting from i

        for (int j = i + 1; j < n; j++) {
            if (arr[j] == 0)
                sum += -1;
            else
                sum += 1;

            // If this is a 0 sum subarray, then
            // compare it with maximum size subarray
            // calculated so far

            if (sum == 0 && maxsize < j - i + 1) {
                maxsize = j - i + 1;
                startindex = i;
            }
        }
    }
```

(maxsize < j - i + 1)

2 < 3 - 0 + 1 = 4 ✓

4 < 5 - 0 + 1 = 6

6 < 8

maxsize = -1 2 4 6 8 →

i = 0 <= 7

st = 0

j = 1 < 9

sum => 1 2 1 2 1 2 1 i = 1 no eq of sub

sum = -1 0 -1 0 -1

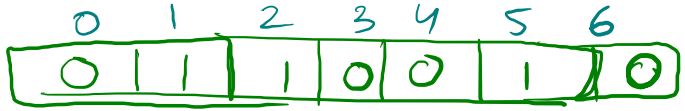
mx = 8 < (2, 4) X

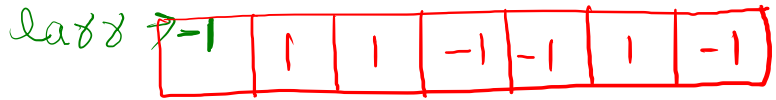
```
int largestsubarray(int[] arr){          Complexity - O(n²)
    int n = arr.length;
int sum=0, mxsize = -1, stindx = -1;

for(int i=0; i<n-1; i++){
        sum = (arr[i]==0)?-1:1;          // initialize

for(int j=i+1; j<n; j++){
            if (arr[j]==0)  sum = sum-1;
            else  sum = sum+1;
        if(sum==0 && mxsize < j-i+1){
            mxsize = j-i+1;
    }   }   stindx = i;          endindx = stindx+mxsize-1;
}
```

Method-2 use hash Map

```
// Initialize sum of elements
int sum = 0;

// Initialize result
int max_len = 0;
int ending_index = -1;
int start_index = 0;

for (int i = 0; i < n; i++) {
    arr[i] = (arr[i] == 0) ? -1 : 1;
}

// Traverse through the given array

for (int i = 0; i < n; i++) {
    // Add current element to sum

    sum += arr[i];

    // To handle sum=0 at last index

    if (sum == 0) {
        max_len = i + 1;
        ending_index = i;
    }

    // If this sum is seen before,
    // then update max_len if required
    if (hM.containsKey(sum)) {
        if (max_len < i - hM.get(sum)) {
            max_len = i - hM.get(sum);
            ending_index = i;
        }
    }
    else // Else put this sum in hash table
        hM.put(sum, i);
}

for (int i = 0; i < n; i++) {
    arr[i] = (arr[i] == -1) ? 0 : 1;
}

int end = ending_index - max_len + 1;
```

↓ starting index = 6 − 6 + 1 = 1

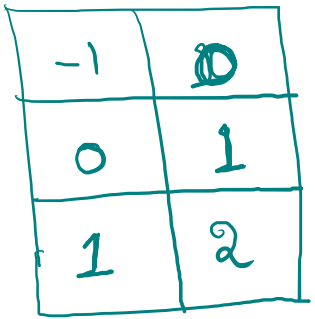




original arr change

sum = −1 0 1 0 −1 −2 −1

i = 0 1 2 3 4 5 6

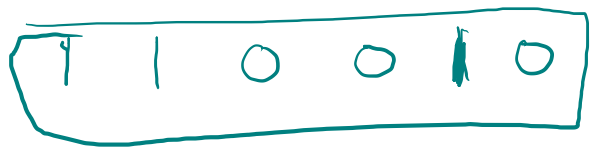


4 < 4 − 0 ✗

4 < 5 − 0 ✓

5 − 0 = 5

5 < 6 → 0



len = 6

1 → 3 0 → 3

```
int maxlen = -1, st = 0, end = 0;
HashMap<I, I> map = new HashMap<>();
```

Step 1 ; change the original array

```
for ( int i=0; i < n ; i+1)
  arr[i] = (arr[i]==0) ? : -1 : 1;
```

Step 2 :

```
for ( int i=0; i<n; i++)
{
    sum = sum + arr[i];
  if ( sum == 0)
   { maxlen = i+1;
     end = i;
   }
```

```
  if (map.containskey(sum))
{
    if( maxlen < i - map.get(sum))
{ maxlen = i - map.get(sum);
    end = i;
  }
}else
    { map.put( sum, i);
  }
st = end - maxlen + 1;
```