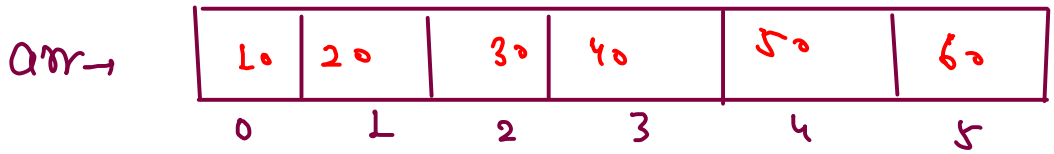


traversal → (1) arr

→ (2) index

Merging

Expectation → disp(arr,0) → 10 20 30 40 50 60

faith → disp(arr,1) → 20 30 40 50 60

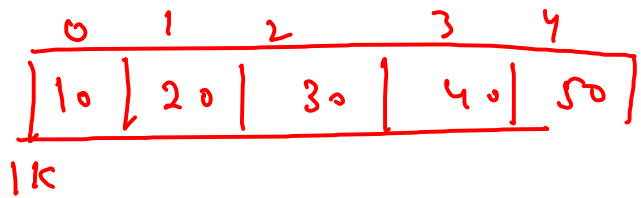
Recursion

disp(arr,0) → 10 20 30 40 50 60

disp(arr,0) → Syso(arr[0]) disp(arr,1)

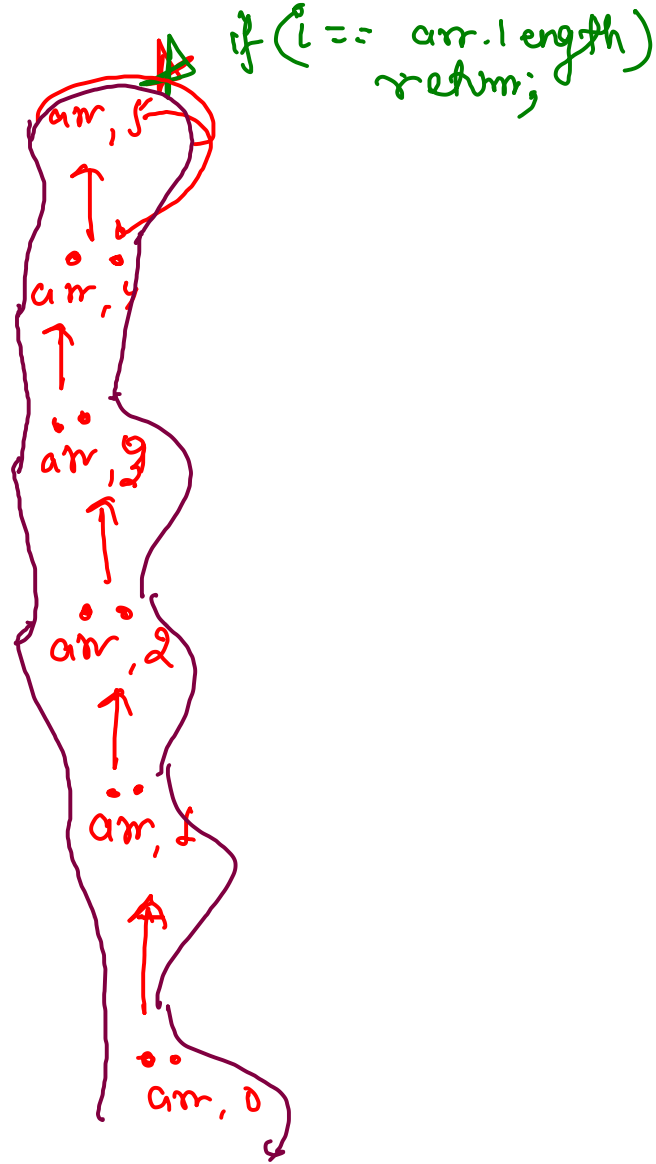
general term disp(arr,i) → Syso(arr[i])

display(arr, i+1);




```java
public static void display(int[] arr, int i) {
    // my work
    System.out.println(arr[i]);
    // faith
    display(arr, i + 1);
}
```

Pre Area



console

10
20
30
40
50

Display Array Reverse -

arr = 10 20 30 40 50

print → 50 40 30 20 10

display(arr, 0);

Expectation → dispRev(arr, 0) → 50 40 30 20 10

faith → dispRev(arr, 1) → 50 40 30 20.

dispRev(arr, 0) → 50 40 30 20 10

dispRev(arr, 0) → dispRev(arr, 1) Syso(arr[0])

Merging.

displayRev(arr, i) → display(Rev(arr, i+1);

Syso(arr[i]);

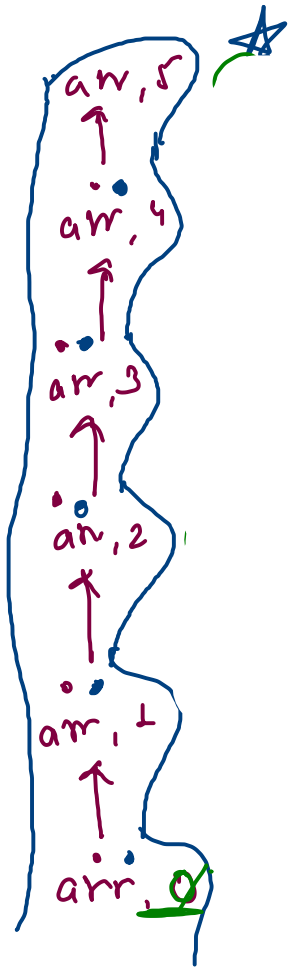
displayRev (low level)

```java
public static void displayReverse(int[] arr, int i) {
    // faith
1.  displayReverse(arr, i + 1);
    // my work
2.  System.out.println(arr[i]);
}
```

} Post Area

arr → 10, 20, 30, 40, 50

0 1 2 3 4

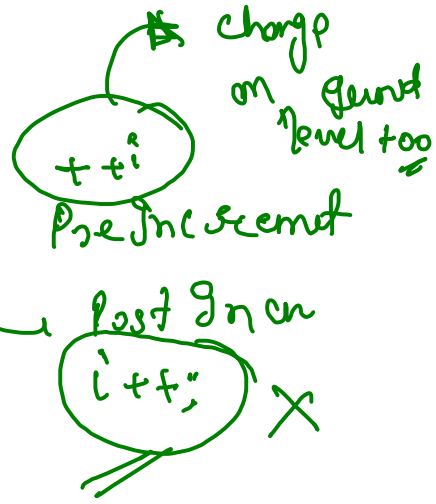


if(i == arr.length) return

Console -

50
40
30
20
10

* NOTE



Max from an Array.

Expectation: max(arr, 0) ⟶ 0 to length-1 } Max value

faith max(arr, 1) ⟶ 1 to length-1 } R.res.

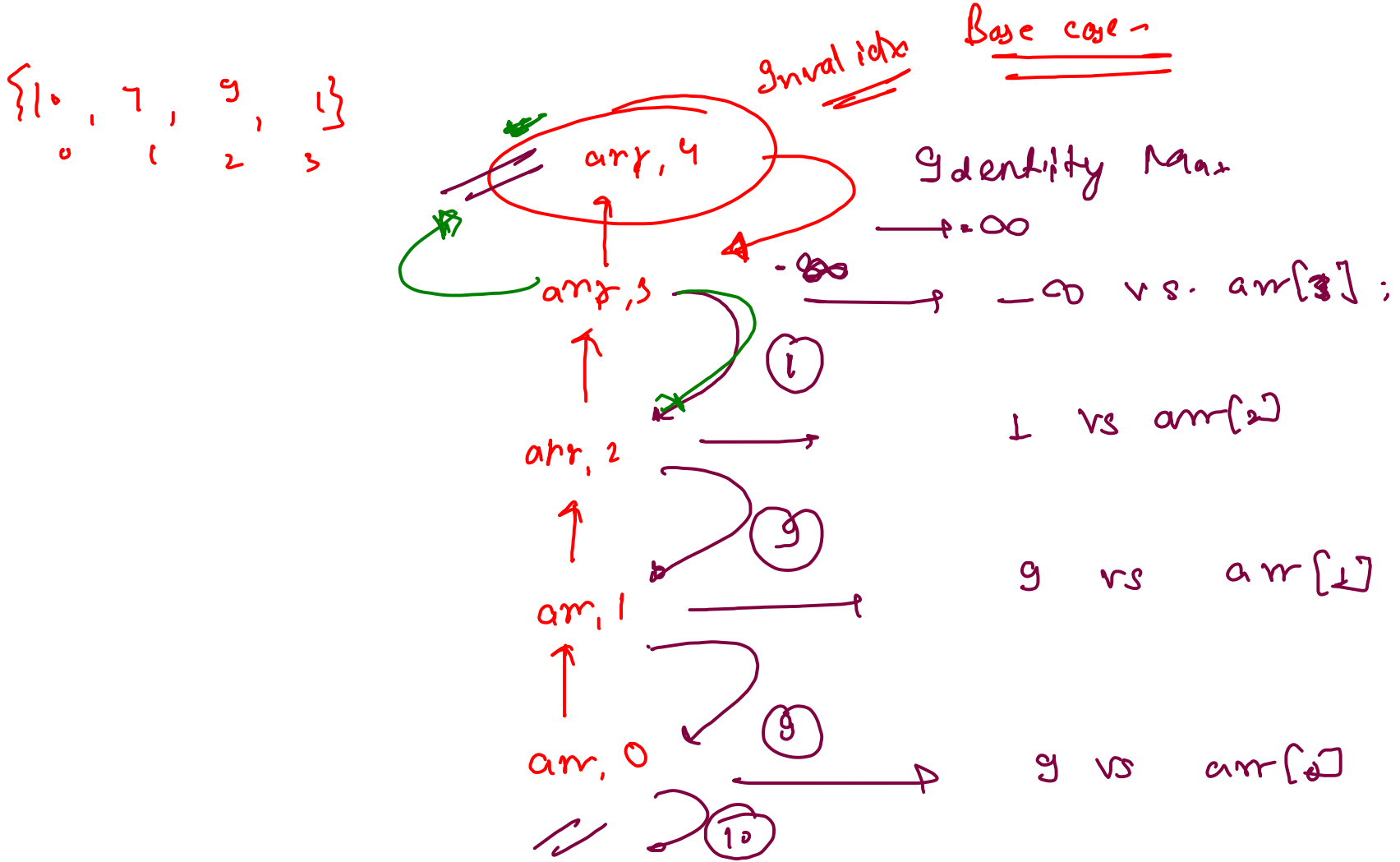
Merging → max(arr, 0) = 0 to length-1

max(arr, 0) = res = max(arr, 1);

ret. Math.max(res vs arr[0]);

max(arr, i) ⟶ res = max(arr, i+1);

return Math.max(res, arr[i]);


{10, 7, 9, 1}
0 1 2 3
Base case -
Invalid idx
arr, 4
Identity Max
-∞
arr, 3
-∞ vs. arr[3];
1
arr, 2
1 vs arr[2]
9
arr, 1
9 vs arr[1]
arr, 0
9
9 vs arr[0]
10

find: arr → 10 20 30 40 50 60

data = 50

Return - if data present → True.

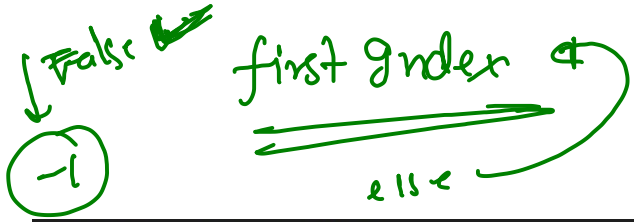
otherwise, → False

Expectation - find (arr, 0, data) → True / False

faith - Find (arr, 1, data) → True / False } Faith on Recursion.

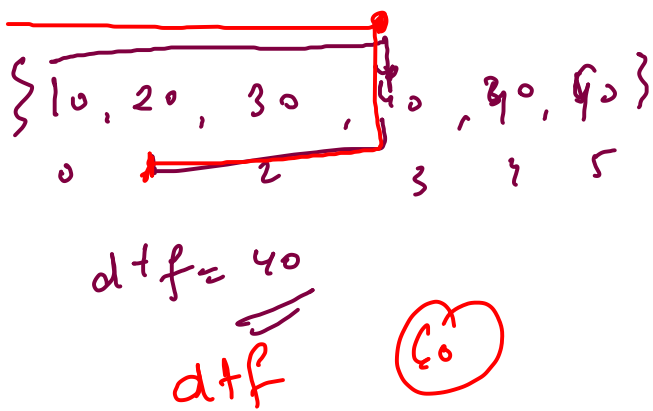
Meaning → fin(arr, 0, data) → // check yourself
if (arr[0] == data)
return True

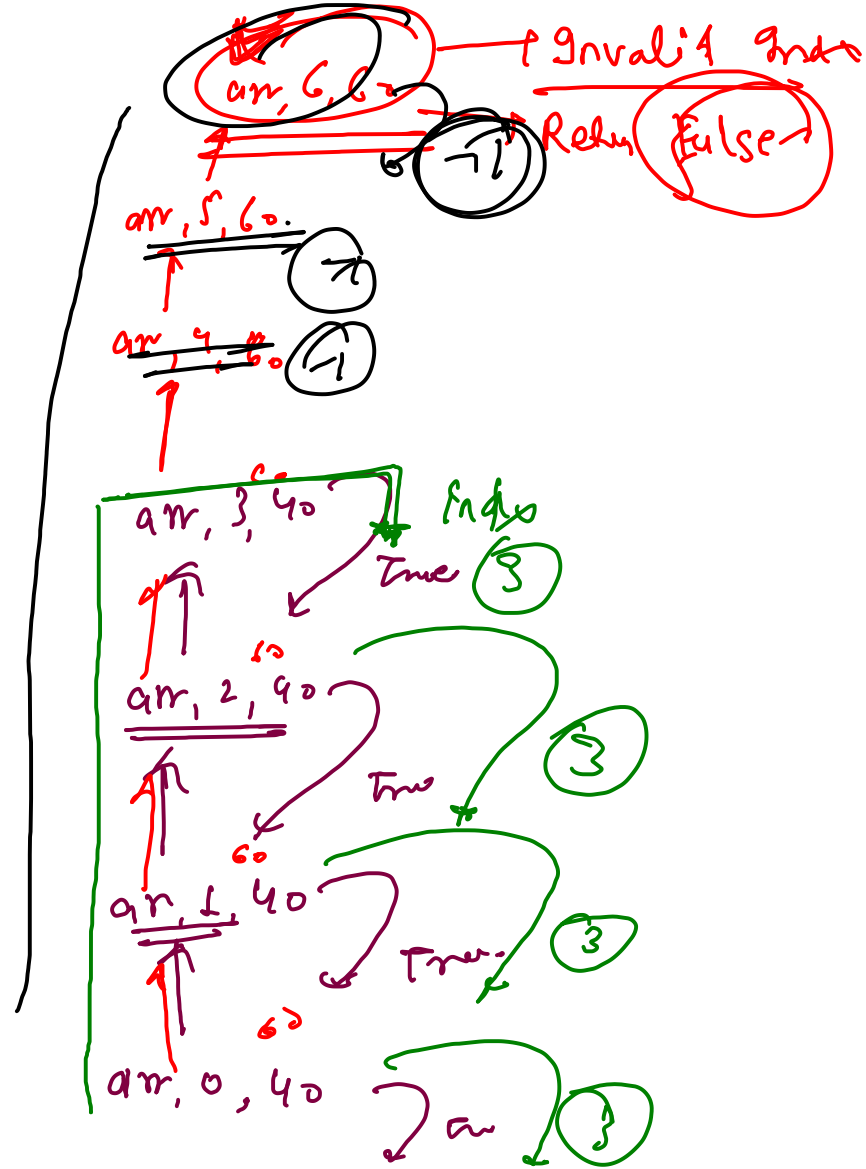
return find (arr, 1, data);




```java
// dtf => data to find
public static boolean find(int[] arr, int indx, int dtf) {
    // check yourself
1.  if(arr[indx] == dtf)
        return true;

2.  // otherwise, rres is ans
    return find(arr, indx + 1, dtf);
}
```





1. Recursion
2. Tns
3. DP.
4. – Stack & Queue.
5. – LinkedList
6. → Trees
7. → Graphs
8. → Heaps & Hashmaps

implement

implementation

DSA

class
obj
cn

Generic Trees
Binary Trees
BST

graphs

maps