# Largest Element in Array

Ip: n = 5
arr[] = < 1, 8, 90, 7, 5 >

Op: 90

Ip: n = 1
arr[] = < 5 >

Op: 5

Ip: n = 7
arr[] = < 1, 2, 0, 5, 3, 2, 4 >

Op: 5

First Approach

I/p: n = 5

arr[] = { 1, 8, 90, 7, 5 }

× n = arr.size();

```
for(int i=0; i<n; i++)
{
    ans = max(ans, arr[i]);
}
cout << ans << endl;
```

90

```
int ans = INT_MIN;
```

↳ -ve

ans = ~~8~~ 90

Recursive Solution

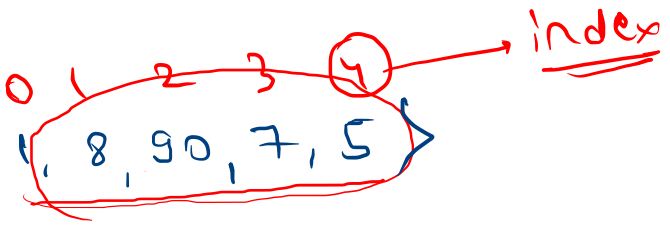
I/p: n = 5

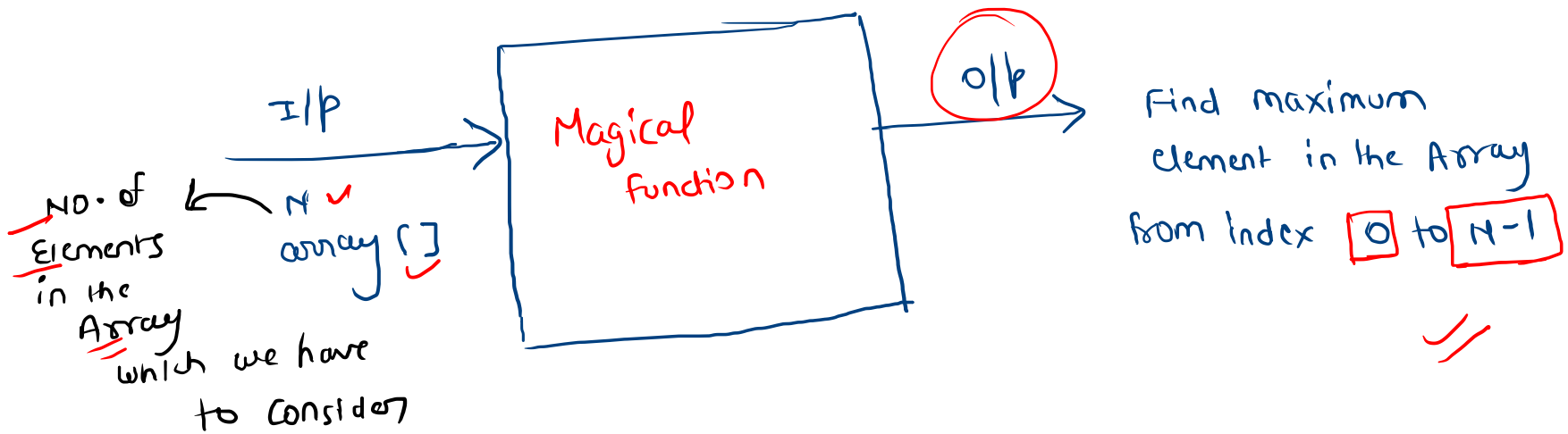
arr[] = { 1, 8, 90, 7, 5 }

Recursive Solution

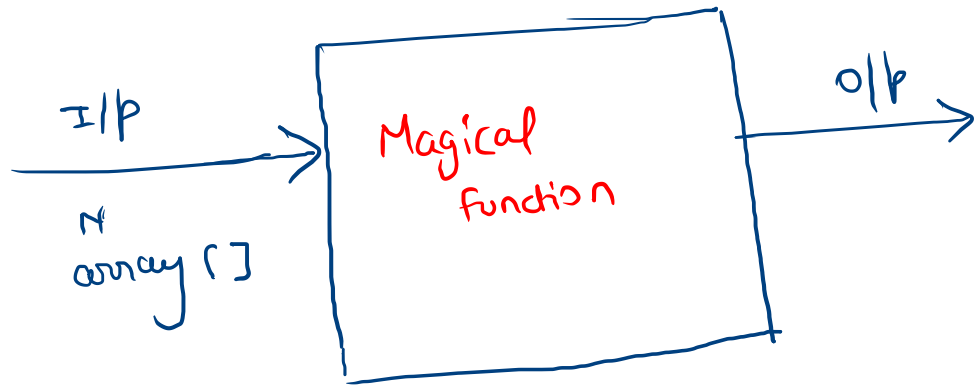
I/p: n = 5

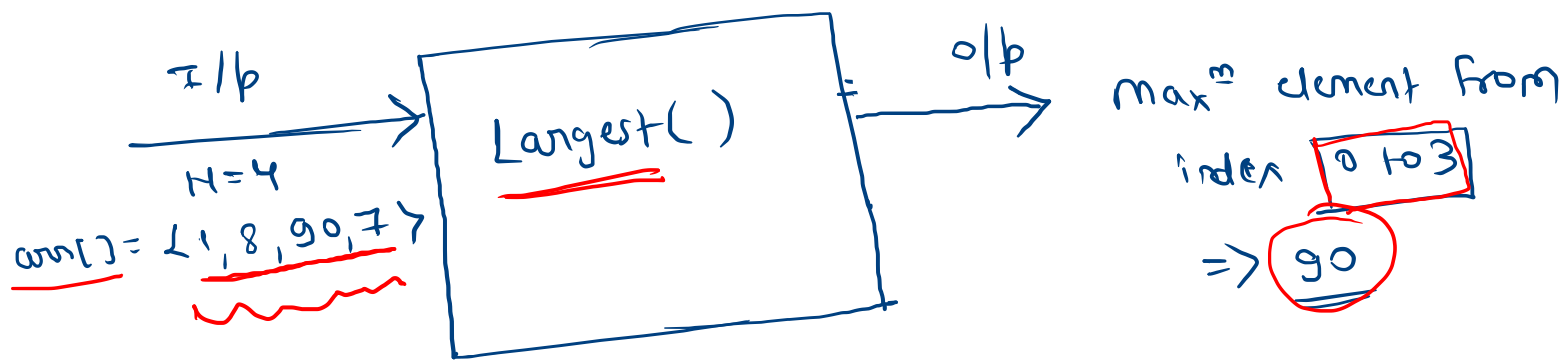
arr[] = { 1, 8, 90, 7, 5 }



Recursive Solution

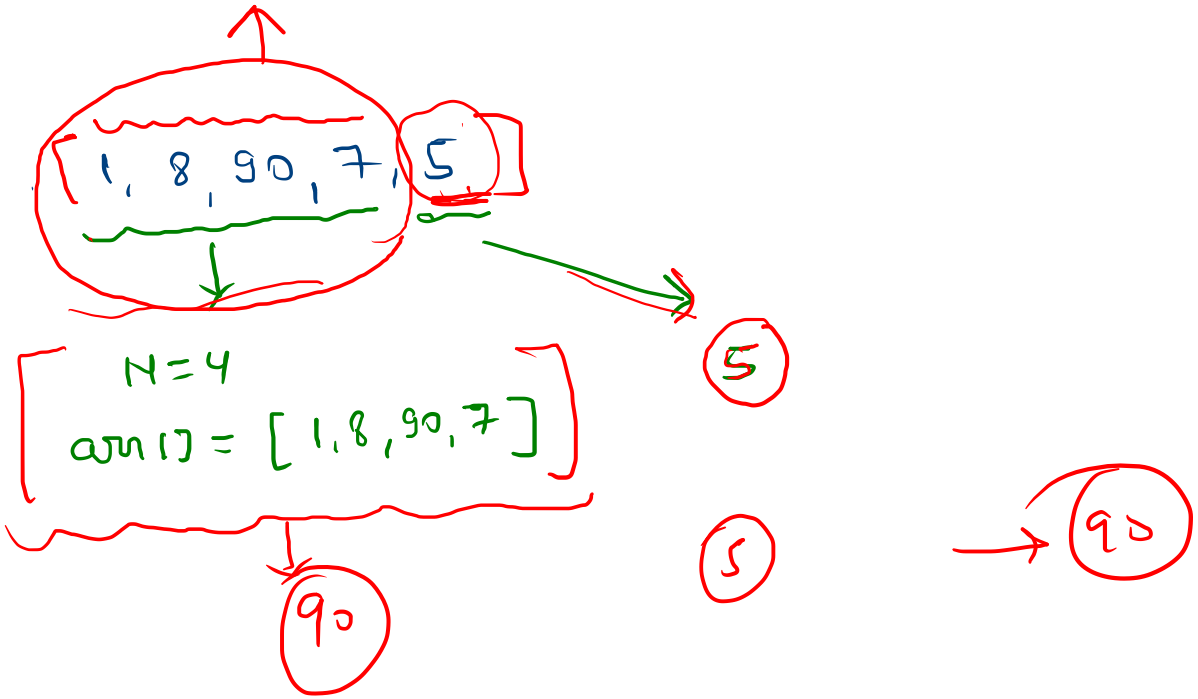


Logical part

I/p: n = 5

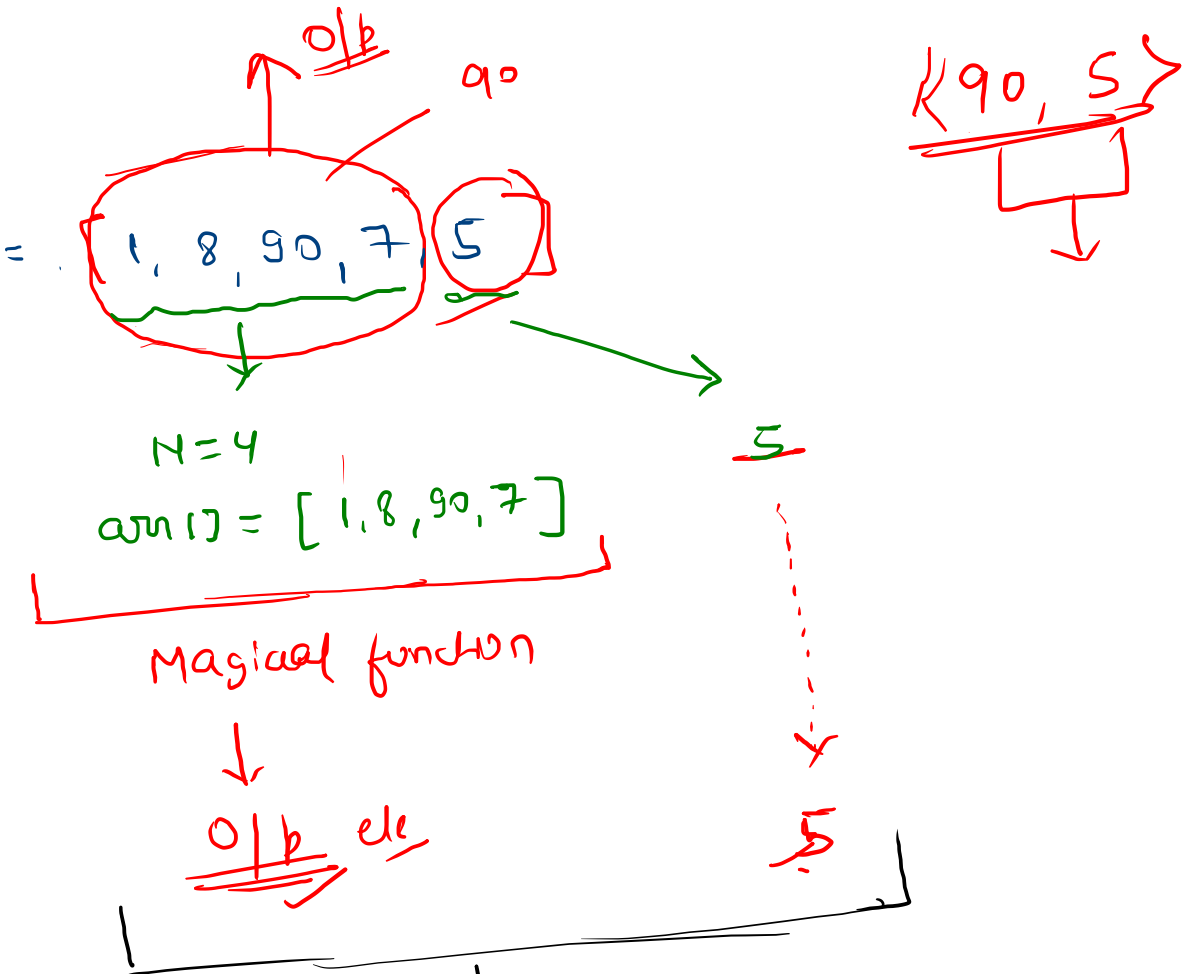
arr[] = [1, 8, 90, 7, 5]



Logical part

I/p: n = 5

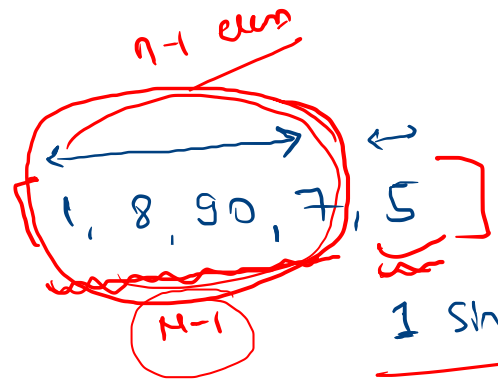
arr[] = 1, 8, 90, 7, 5



Logical part

I/p: n = 5

arr[] = 1, 8, 90, 7, 5



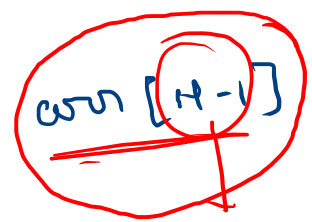

n = 5

n-1 = 4

Element

⇓

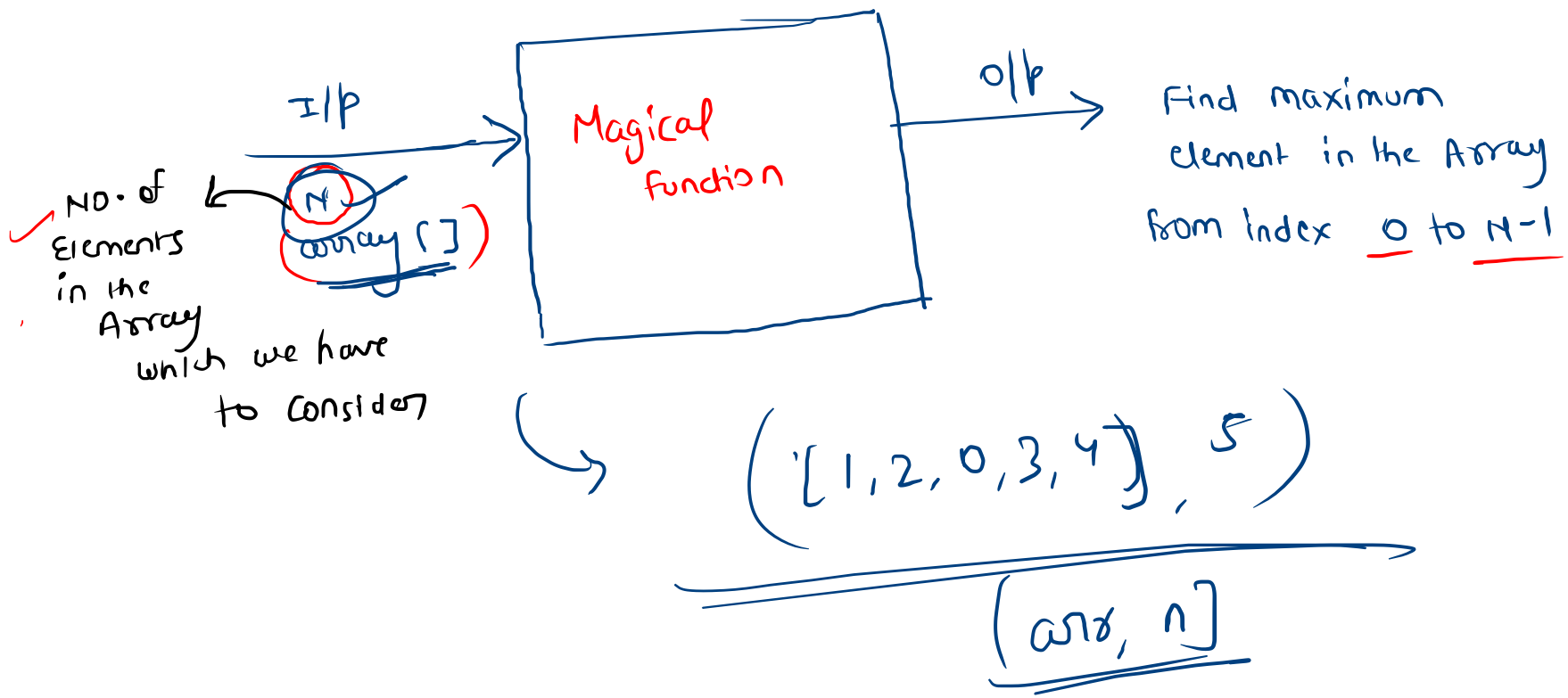
largest ( arr, n-1 ) , arr[n-1]




o/p

Largest ( [1, 2, 0, 3, 4], 3 )

2



Recursive Solution

```
int Largest ( int arr[], int N )
{
    // Base Condition
    return max ( Largest ( arr, (N-1) ), arr[N-1] );
}
```

Recursive Solution

```
int Largest ( int arr[], int N )
{
    // Base Condition
    return max ( Largest ( arr, N-1 ), arr[N-1] );
}
```

Number of elements

index of elements

Base Condition

Smallest Valid Input → check output

```
int Largest ( int arr[] , int n )
```

n → Variable

n = 0, 1, 2, 3 . . .

x ↓ ↓

Smallest Valid input

```
if ( n == 1 ) return arr[0];
```

arr = [5]

n = 1 → arr[0]

Recursive Solution

```
int Largest ( int arr[], int N )
{
    if ( n == 1 ) return arr[0];
    return max ( Largest ( arr, N-1 ), arr[N-1] );
}
```

max

Logical doubt

↓

we are passing whole Array

Largest ( int arr[], int n )