

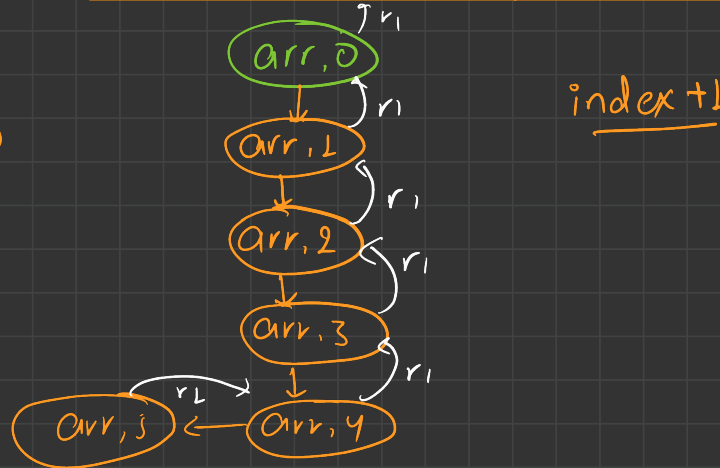
## Linear Search

arr →  
x = 9

ui = 9

<u>0</u>	1	2	3	4	5	6
2	7	4	3	<u>11</u>	<u>9</u>	13

x = 21



```

bool LinearSearch (int arr[], int x, int index, int N) {
    if (index == N) {
        return 0;
    }
    if (arr[index] == x) {
        return 1;
    }
    return LinearSearch(arr, x, index + 1, N);
}

```

A call stack diagram illustrating the recursive calls for `LinearSearch(arr, 9, 0, 7)`. The calls are shown in a vertical sequence, each in an oval, with arrows indicating the return flow from the bottom call to the top call.

- `LS(9, 0, 7)`
- `LS(9, 1, 7)`
- `LS(9, 2, 7)`
- `LS(9, 3, 7)`
- `LS(9, 4, 7)`
- `LS(9, 5, 7)` (This call returns 1, indicated by a curved arrow pointing back to the previous call.)

## Binary Search

0	1	2	3	4	5
2	6	8	10	14	26

→ arr  $x = 10$

start = mid + 1  
end = mid - 1

↑ mid = 2  
↑ start = 3  
↑ end = 5

$$\frac{\text{start} + \text{end}}{2} = \frac{0 + 5}{2}$$

$$\text{mid} = \frac{0 + 5}{2} = 2$$

$$\frac{3 + 5}{2} = 4$$

```

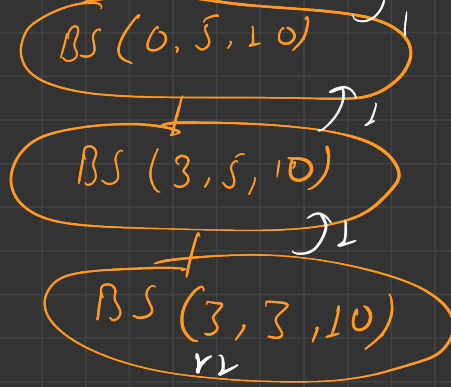
BinarySearch (int arr, int start, int end, int x) {
    if (start > end) {
        return 0;
    }
    int mid = start + (end - start) / 2;
    if (arr[mid] == x) {
        return 1;
    }
    else if (arr[mid] < x)
        return BinarySearch (arr, mid + 1, end, x);
    else {
        return BinarySearch (arr, start, mid - 1, x);
    }
}

```

```

BinarySearch (int arr, int start, int end, int x) {
    if (start > end) {
        return 0;
    }
    int mid = start + (end - start) / 2;
    if (arr[mid] == x) {
        return 1;
    }
    else if (arr[mid] < x) {
        return BinarySearch (arr, mid + 1, end, x);
    }
    else {
        return BinarySearch (arr, start, mid - 1, x);
    }
}

```



start = 0  
 end = 5  
 n = 10  
 mid = 2.5  
 =

$$mid = start + \frac{(end - start)}{2}$$

20 10 10 7 6 4 2 1

n byte









































