10/10/2023

RECURSION CLASS2

✅Program 01: Climbing stairs (Leetcode-70)

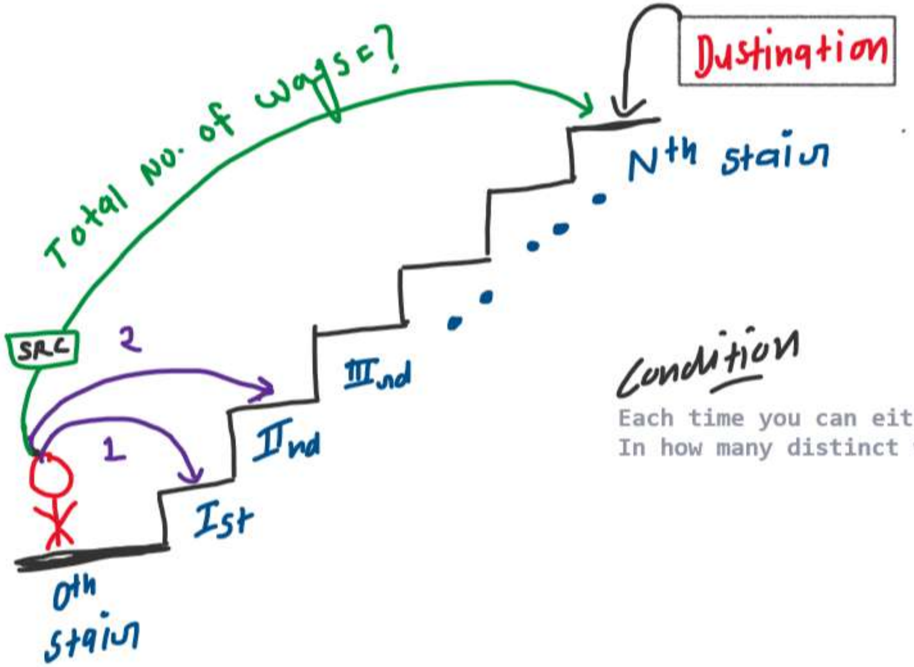
Ex

N=1
Output=1

N=0
Output=1

N=2
Output=2

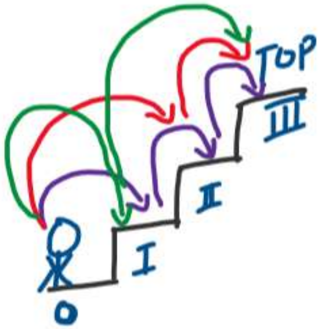
N=3
Output=3



Condition

Each time you can either climb 1 or 2 steps.
In how many distinct ways can you climb to the top?

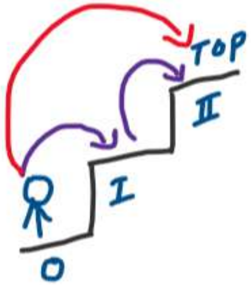
Ex N=3



1 1 1
2 1
1 2

Total 3 ways

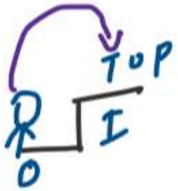
Ex N=2



1 1
2

Total 2 ways

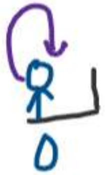
Ex N=1



1

Total 1 way

Ex N=0



1

Total 1 way

Nth stair → Total No of ways

$f(N)$ = Total No of ways to nth stairs

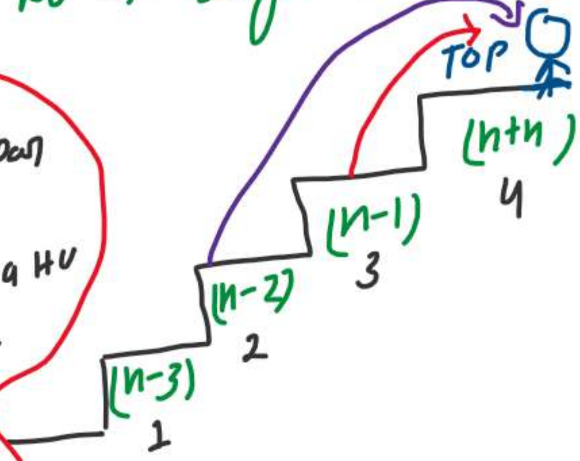
way1

JAB MI stair NO. 2 par tha TO MI Two stair EK SATH par kar Rha Hu 4th stair Tak jane ke liye TO US TARIKE KO MI (n-2) Bol Rha Hu

(+) OR

JAB MI 3rd stair par hu TO MI one stair EK Time par chal Raha Hu 4th stair tak jane ke liye TO US way KO MI (N-1) Bol Rha Hu = Total way

way2



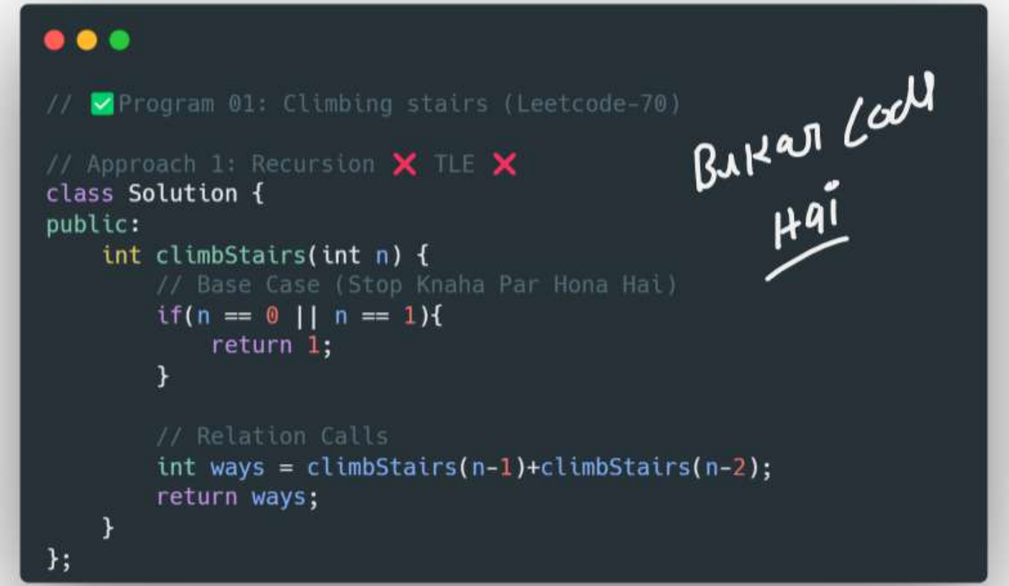
$\boxed{N=4}$

Relation

$$\boxed{f(N) = f(N-1) + f(n-2)}$$

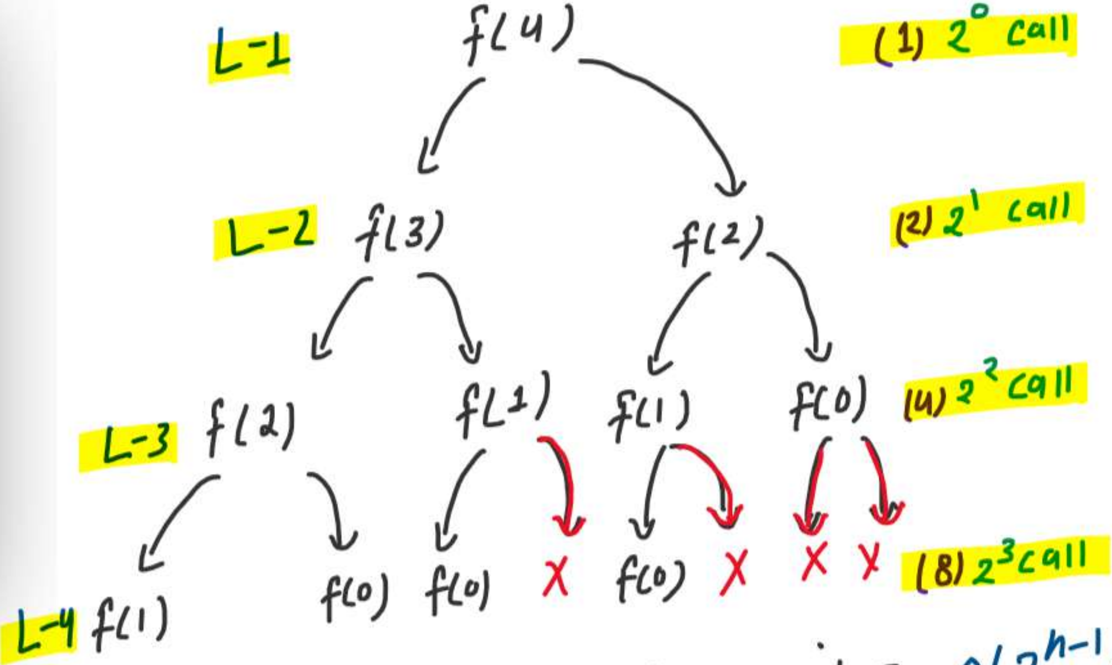
Base case

```
if(n==0 || n==1)
    return 1
```



```
// ✅Program 01: Climbing stairs (Leetcode-70)

// Approach 1: Recursion ❌ TLE ❌
class Solution {
public:
    int climbStairs(int n) {
        // Base Case (Stop Knaha Par Hona Hai)
        if(n == 0 || n == 1){
            return 1;
        }

        // Relation Calls
        int ways = climbStairs(n-1)+climbStairs(n-2);
        return ways;
    }
};
```

Bakar Code Hai



Time Complexity = $O(2^{n-1})$

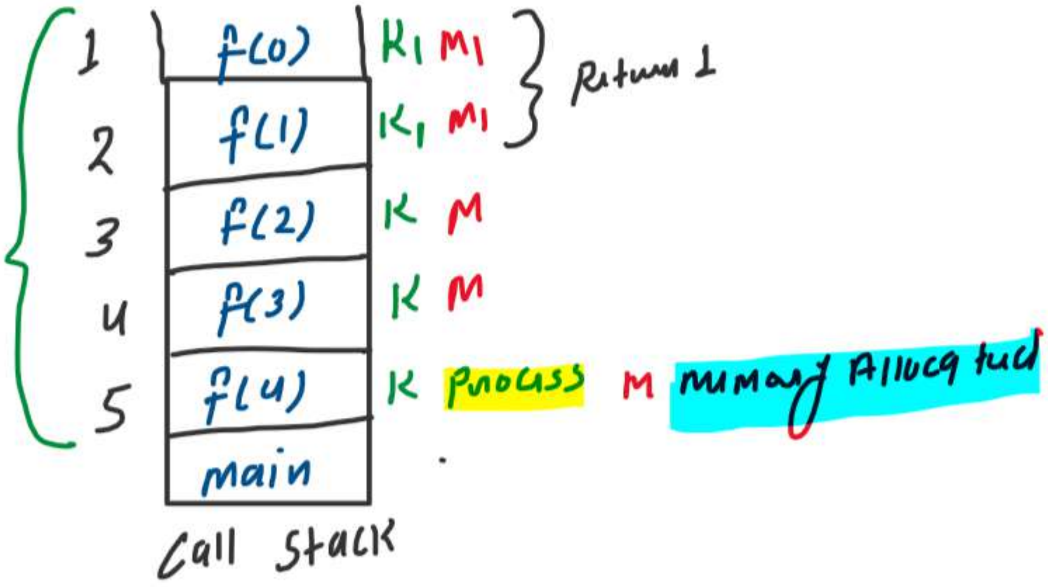
$= O(2^n)$

Space Compuxity

$\boxed{N=4}$

$O(N+1)$ = total ENTRY 5

S.C = O(N)



✅Program 02: Print array

arr	10	20	30	40	50
	0	1	2	3	4

Initial Index

index = 0

size = 5

Relation

f(arr ,index+1 , size)

Base case

if (index >= size)
return

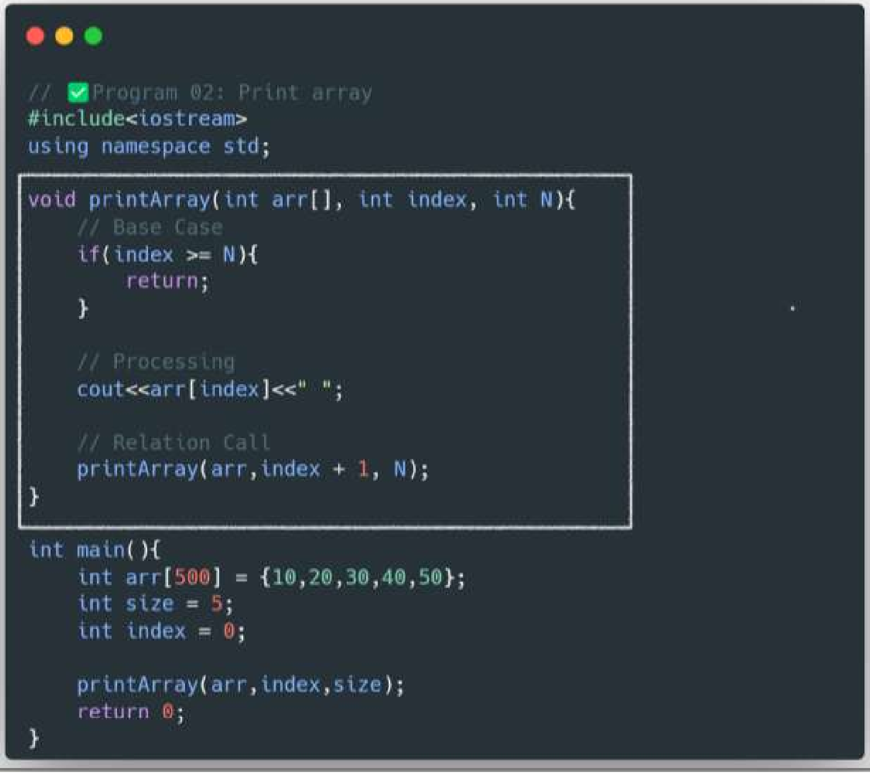
Relation

f(arr ,index+1 , size)

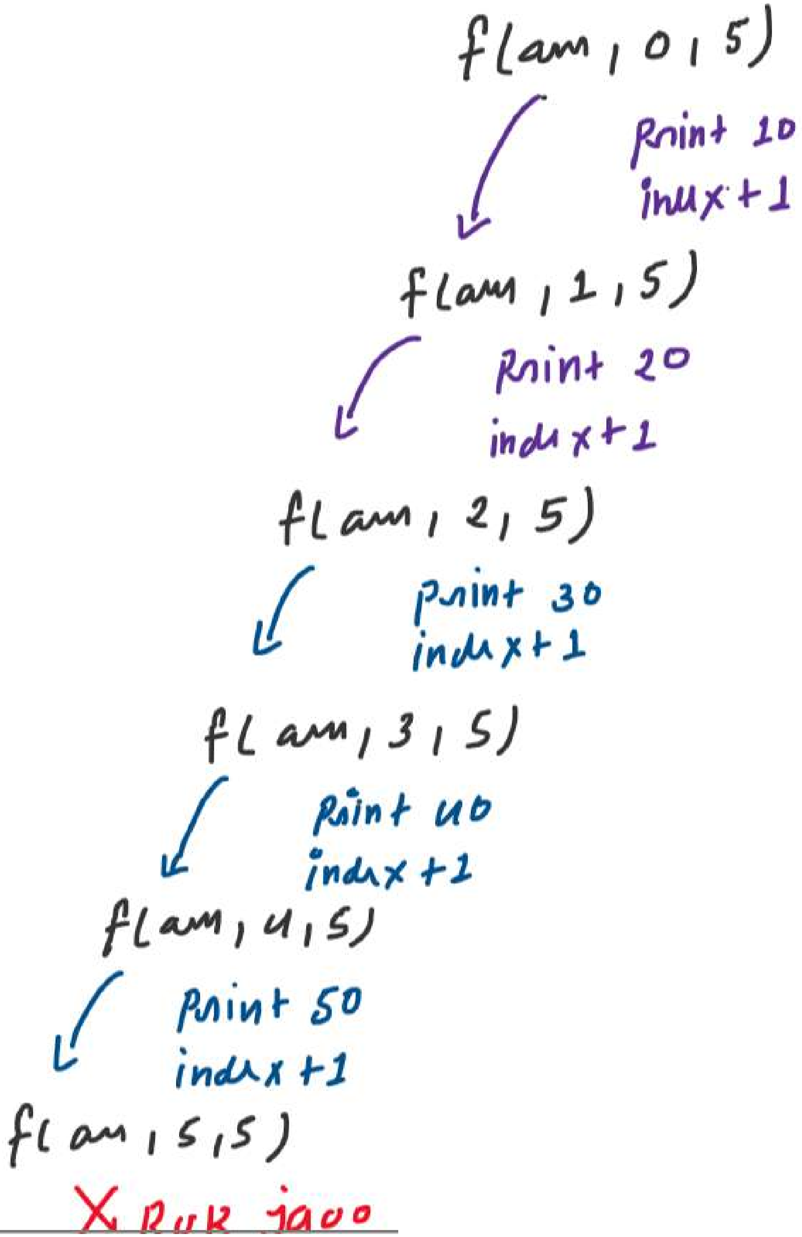
Base case

if (index >= size)

return



```
// ✅Program 02: Print array
#include<iostream>
using namespace std;

void printArray(int arr[], int index, int N){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    cout<<arr[index]<<" ";

    // Relation Call
    printArray(arr,index + 1, N);
}

int main(){
    int arr[500] = {10,20,30,40,50};
    int size = 5;
    int index = 0;

    printArray(arr,index,size);
    return 0;
}
```



f(arr, 0, 5)

Print 10
index+1

f(arr, 1, 5)

Print 20
index+1

f(arr, 2, 5)

print 30
index+1

f(arr, 3, 5)

Print 40
index+1

f(arr, 4, 5)

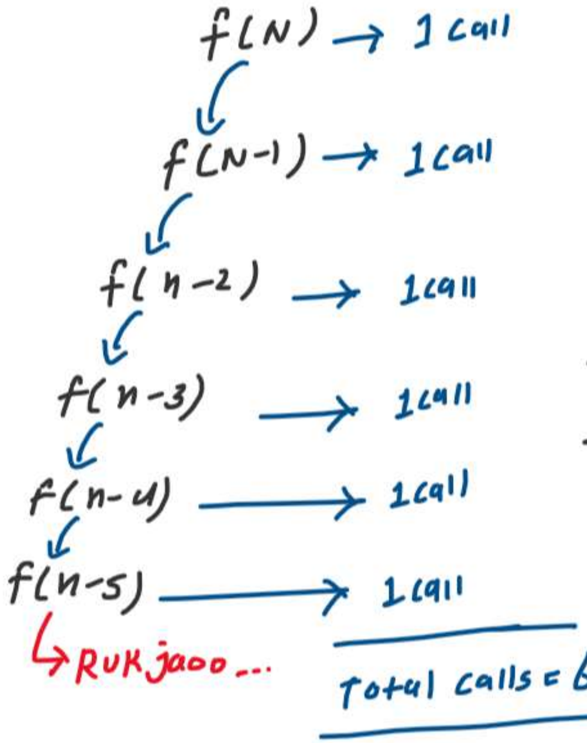
Print 50
index+1

f(arr, 5, 5)

X Ruk jaoo

Time Complexity

$\boxed{N=5}$
↳ size of Array



T.C. = 5+1

= N+1

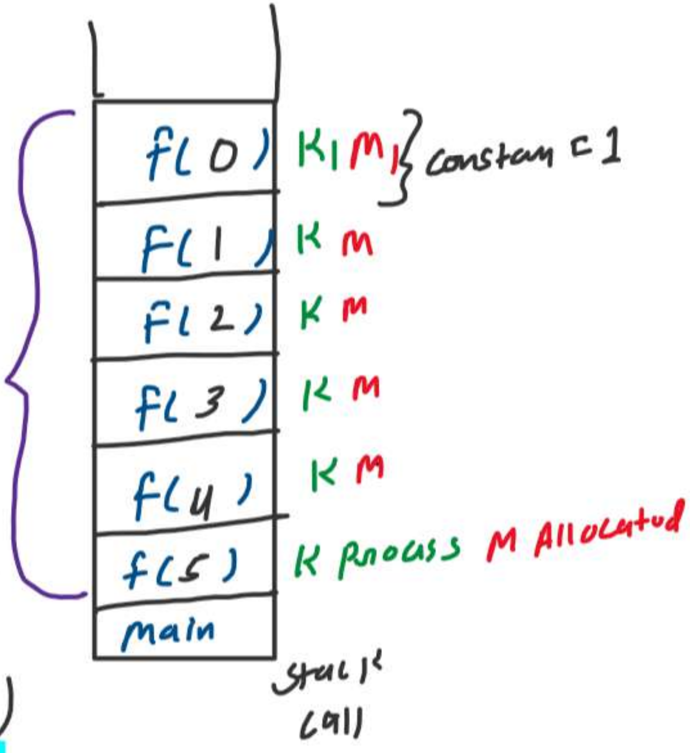
= O(N+1)

= O(N)

Space Complexity

$\boxed{n=5}$

Total ENTRY = 6



S.C. = 5space+1

= N+1

= O(N+1)

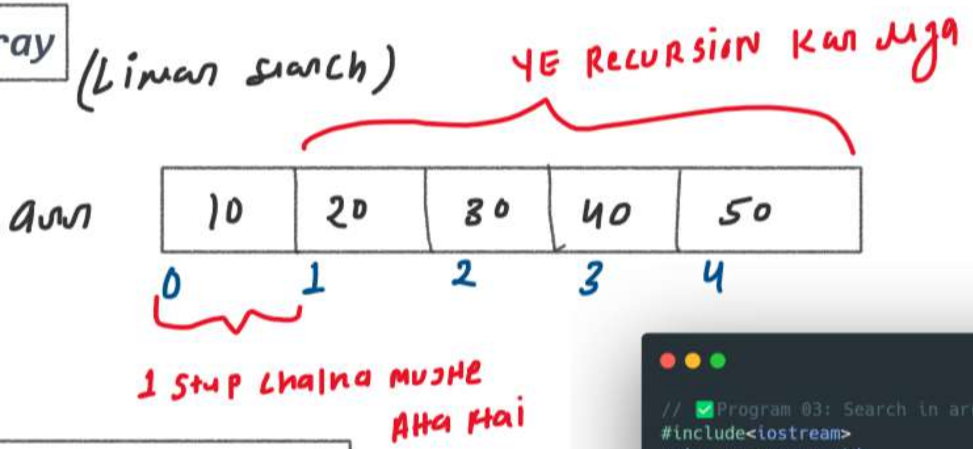
= O(N)

✅Program 03: Search in array

(Linear search)

Target = 50 , 100

Output = True , False

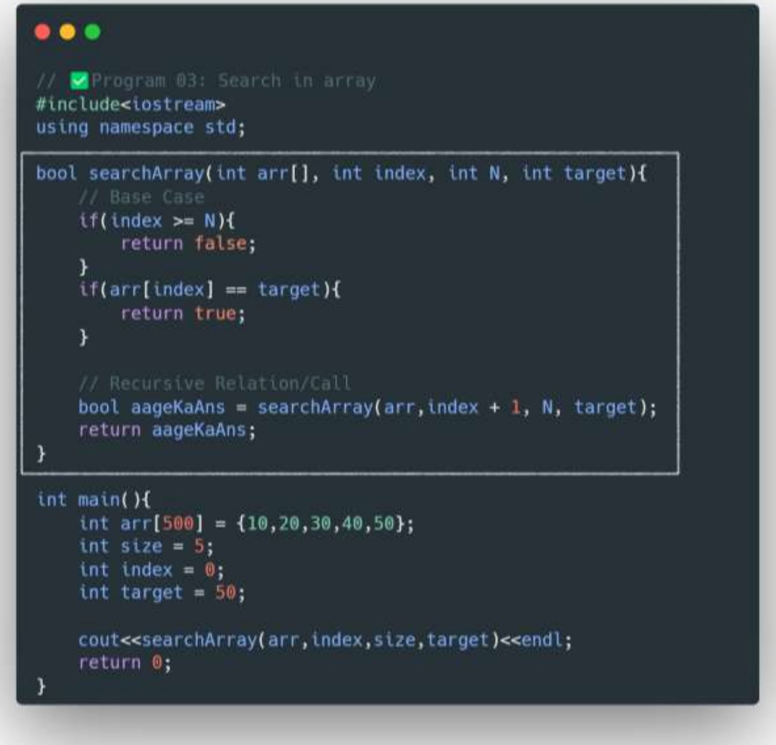


Relation

$$\boxed{f(arr, index+1, size, Target)}$$

Base case

```
if(arr[index] == Target)
    ↳ return True

if( index >= size)
    ↳ return false
```



```cpp
// ✅Program 03: Search in array
#include<iostream>
using namespace std;

bool searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return false;
    }
    if(arr[index] == target){
        return true;
    }

    // Recursive Relation/Call
    bool aageKaAns = searchArray(arr,index + 1, N, target);
    return aageKaAns;
}

int main(){
    int arr[500] = {10,20,30,40,50};
    int size = 5;
    int index = 0;
    int target = 50;

    cout<<searchArray(arr,index,size,target)<<endl;
    return 0;
}
```

Time Complexity

$= O(N+1)$

$\boxed{\text{Time Comp} = O(N)}$

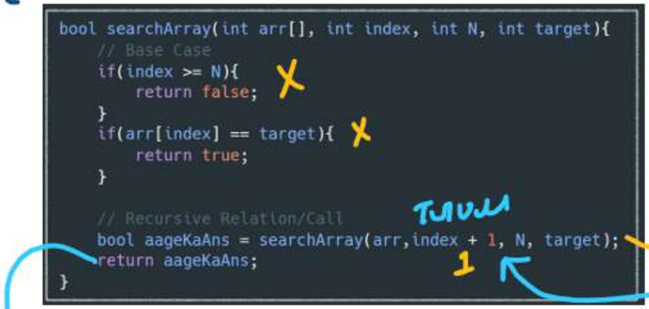
Like as Print Array

Space Complexity

$\boxed{SoCo = O(N)}$

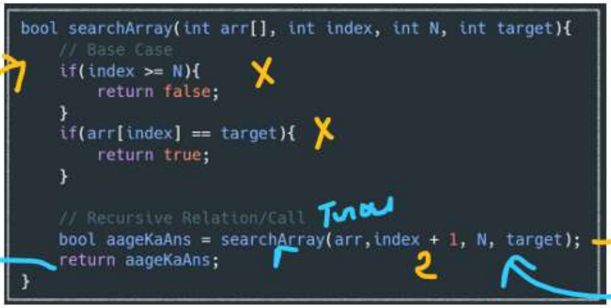
like as

f(arr, 0, 3, 30)



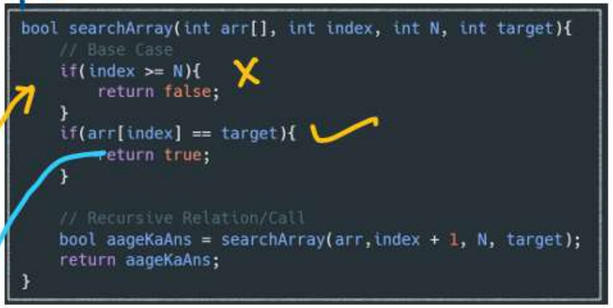
```
bool searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return false;
    }
    if(arr[index] == target){
        return true;
    }

    // Recursive Relation/Call
    bool aageKaAns = searchArray(arr,index + 1, N, target);
    return aageKaAns;
}
```

f(arr, 1, 3, 30)



```
bool searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return false;
    }
    if(arr[index] == target){
        return true;
    }

    // Recursive Relation/Call
    bool aageKaAns = searchArray(arr,index + 1, N, target);
    return aageKaAns;
}
```

f(arr, 2, 3, 30)



```
bool searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return false;
    }
    if(arr[index] == target){
        return true;
    }

    // Recursive Relation/Call
    bool aageKaAns = searchArray(arr,index + 1, N, target);
    return aageKaAns;
}
```

$\boxed{\text{TRUE}}$ ANS

f(arr, 0, 3, 30)

Array ↑ arr; Initial Index value ↑ 0; Size ↑ 3; Target ↑ 30

arr	10	20	30
	0	1	2

Target = 30

✅Program 04: Minimum in array

$\boxed{mini = 5}$

arr

20	30	10	5	11
0	1	2	3	4

BASE case

```
if (index >= size)
    return
```

Processing

```
mini = min( arr[index], mini)
```

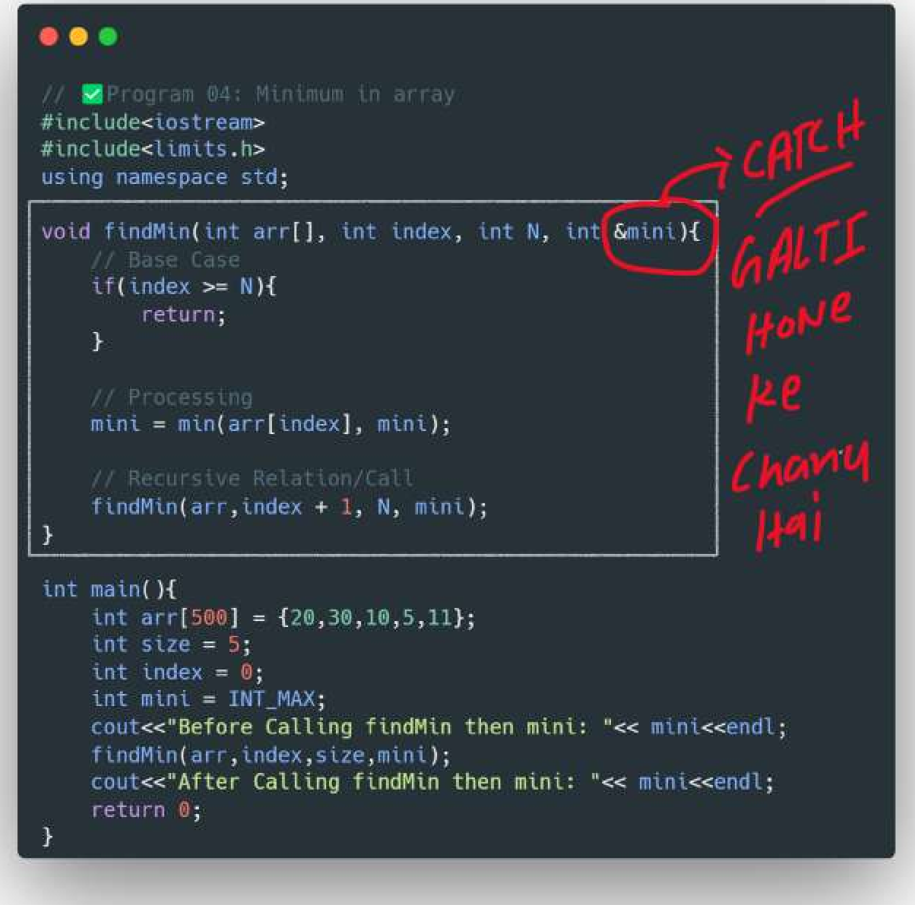
Relation

```
f(arr, index+1, size, mini)
```

{where mini = ~~INT_MAX~~}

index = 0	20
index = 1	20
index = 2	10
index = 3	5
index = 4	5

Index = 5 X Stop → BASE CASE



```
// ✅Program 04: Minimum in array
#include<iostream>
#include<limits.h>
using namespace std;

void findMin(int arr[], int index, int N, int &mini){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    mini = min(arr[index], mini);

    // Recursive Relation/Call
    findMin(arr,index + 1, N, mini);
}

int main(){
    int arr[500] = {20,30,10,5,11};
    int size = 5;
    int index = 0;
    int mini = INT_MAX;
    cout<<"Before Calling findMin then mini: "<< mini<<endl;
    findMin(arr,index,size,mini);
    cout<<"After Calling findMin then mini: "<< mini<<endl;
    return 0;
}
```

CATCH

GALTI Hone ke Chany Hai

Worst CASE me TC

O(N)

Space Complexity

O(N)

✅Program 05: Arrays even element store in vector

Base case

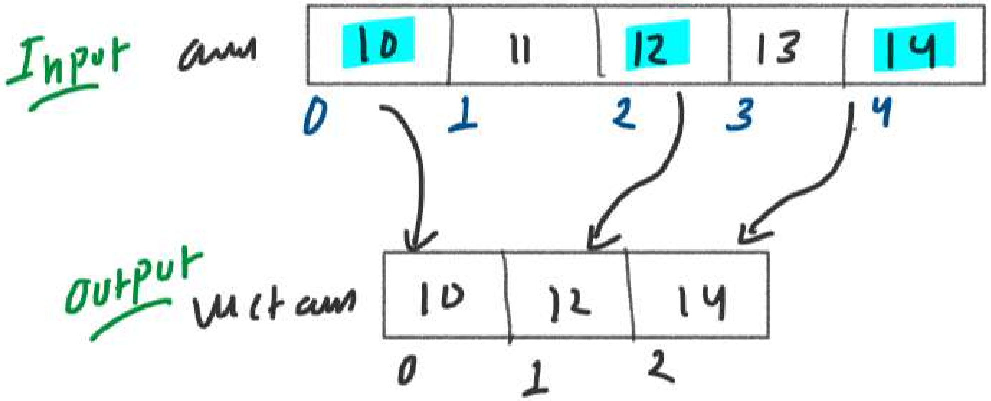
```
if(index >= size)
    return
```

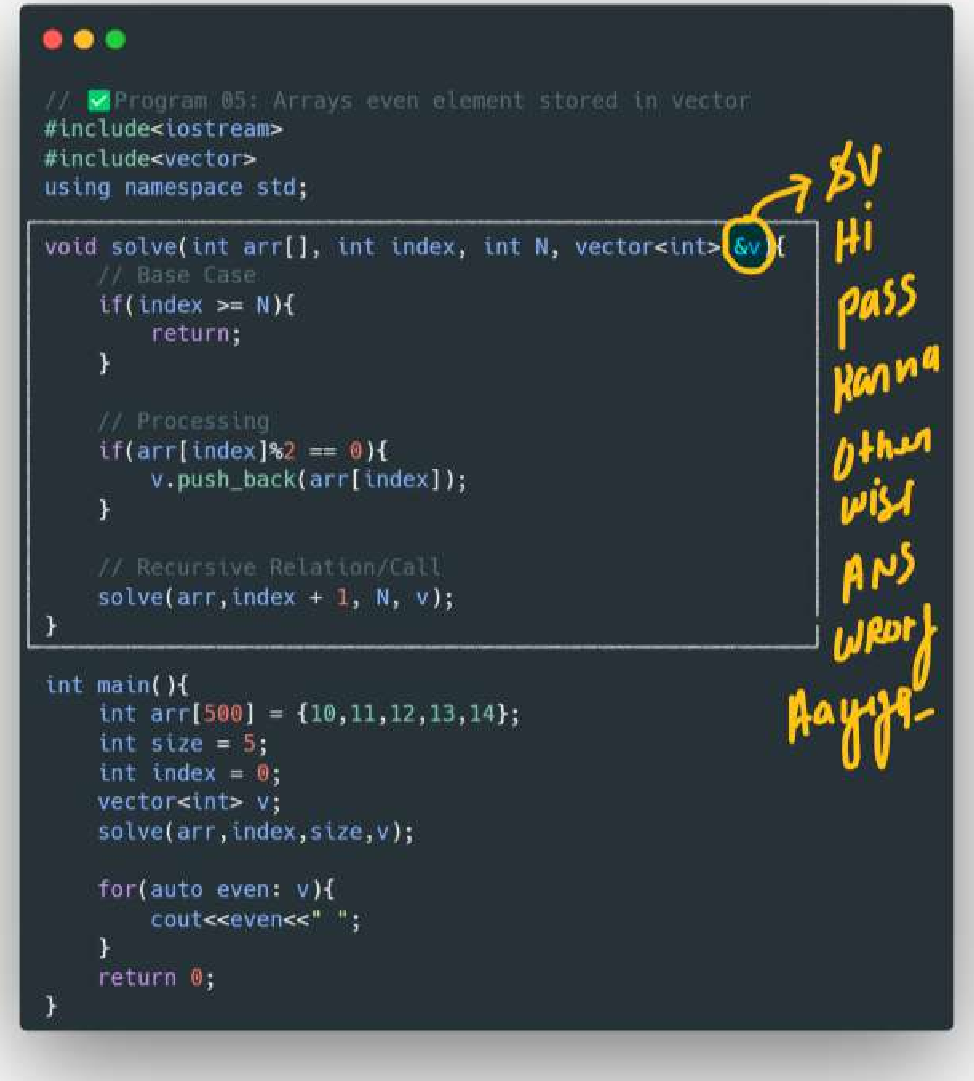
Process

```
if(arr[index] % 2 == 0)
    v.push_back(arr[index])
```

Relation

```
f(arr, index+1 ; size, V)
```





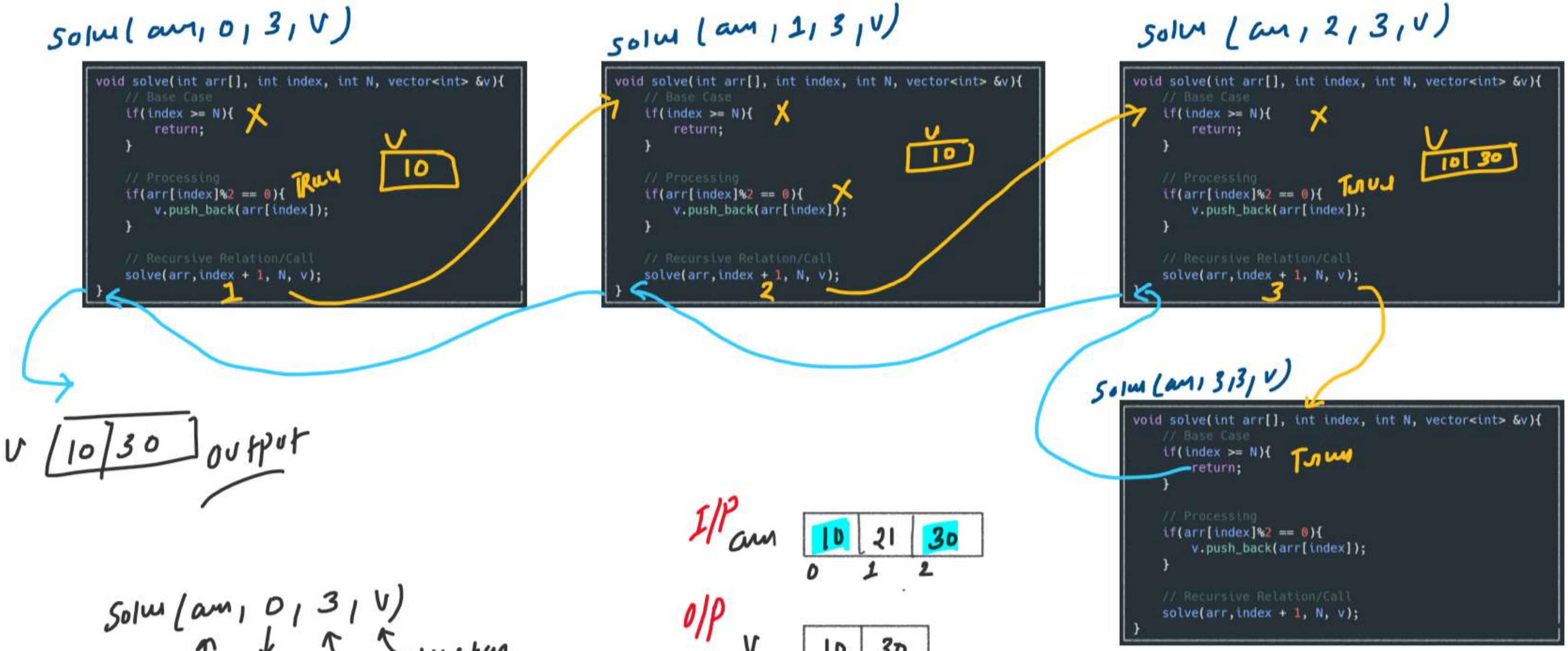
```cpp
// ✅Program 05: Arrays even element stored in vector
#include<iostream>
#include<vector>
using namespace std;

void solve(int arr[], int index, int N, vector<int> &v){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    if(arr[index]%2 == 0){
        v.push_back(arr[index]);
    }

    // Recursive Relation/Call
    solve(arr,index + 1, N, v);
}

int main(){
    int arr[500] = {10,11,12,13,14};
    int size = 5;
    int index = 0;
    vector<int> v;
    solve(arr,index,size,v);

    for(auto even: v){
        cout<<even<<" ";
    }
    return 0;
}
```

&V Hi pass karna otherwise ANS wrong Aayega

Upper Bound case me

T.C = O(N)

S.C = O(N)



solve(arr, 0, 3, v)

```
void solve(int arr[], int index, int N, vector<int> &v){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    if(arr[index]%2 == 0){
        v.push_back(arr[index]);
    }

    // Recursive Relation/Call
    solve(arr,index + 1, N, v);
}
```

solve(arr, 1, 3, v)

```
void solve(int arr[], int index, int N, vector<int> &v){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    if(arr[index]%2 == 0){
        v.push_back(arr[index]);
    }

    // Recursive Relation/Call
    solve(arr,index + 1, N, v);
}
```

solve(arr, 2, 3, v)

```
void solve(int arr[], int index, int N, vector<int> &v){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    if(arr[index]%2 == 0){
        v.push_back(arr[index]);
    }

    // Recursive Relation/Call
    solve(arr,index + 1, N, v);
}
```

solve(arr, 3, 3, v)

```
void solve(int arr[], int index, int N, vector<int> &v){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    if(arr[index]%2 == 0){
        v.push_back(arr[index]);
    }

    // Recursive Relation/Call
    solve(arr,index + 1, N, v);
}
```

solve(arr, 0, 3, v)

arr → array, 0 → index, 3 → size, v → vector

I/P arr

10	21	30
0	1	2

O/P v

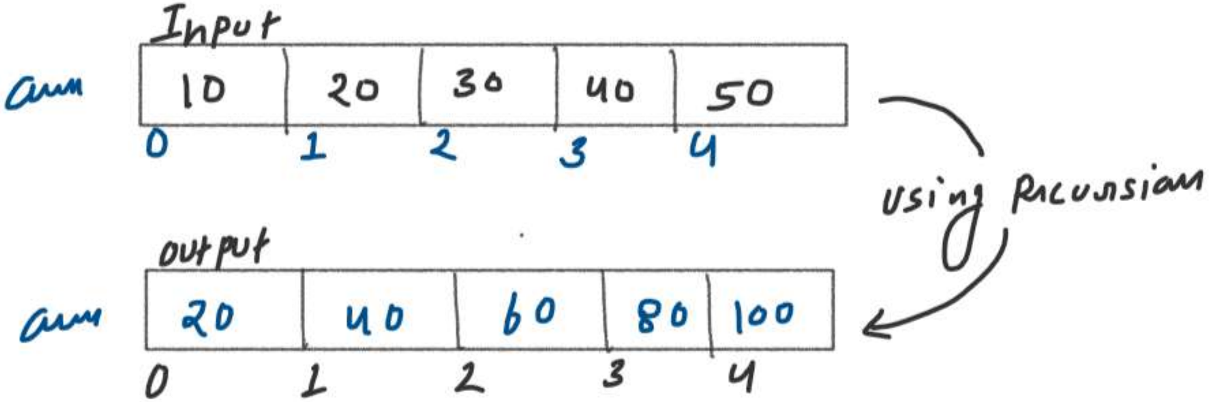
10	30
0	1

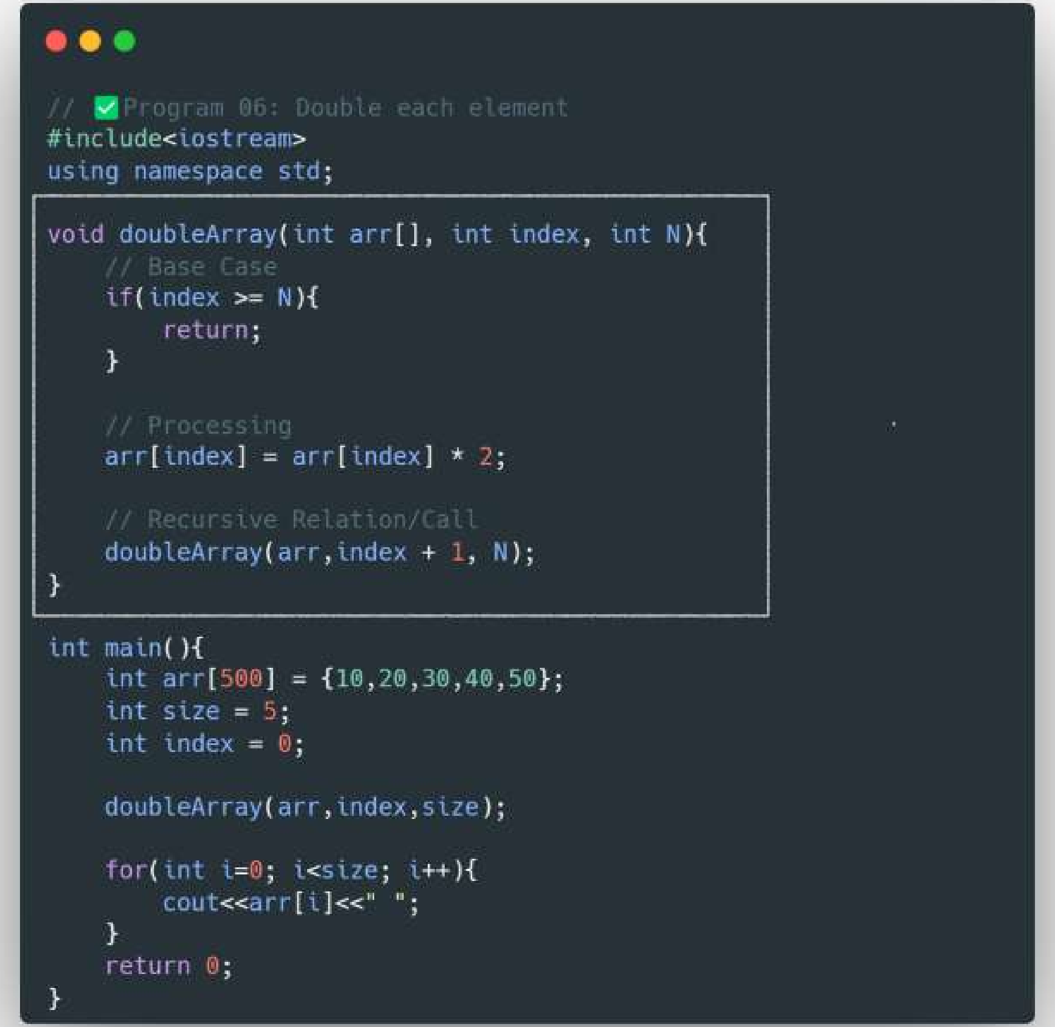
✅Program 06: Double each element

```
if (index >= size)
    return
```

```
arr[index] = arr[index] *2
```

```
f(arr, Index+1 ; size)
```





```cpp
// ✅Program 06: Double each element
#include<iostream>
using namespace std;

void doubleArray(int arr[], int index, int N){
    // Base Case
    if(index >= N){
        return;
    }

    // Processing
    arr[index] = arr[index] * 2;

    // Recursive Relation/Call
    doubleArray(arr,index + 1, N);
}

int main(){
    int arr[500] = {10,20,30,40,50};
    int size = 5;
    int index = 0;

    doubleArray(arr,index,size);

    for(int i=0; i<size; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

T.C. = O(N)

S.C. = O(N)

✅Program 07: Find in array

Linear search

arr	10	20	30	40	50
	0	1	2	3	4

target = 50 , 100

output = 4 , -1

Base case

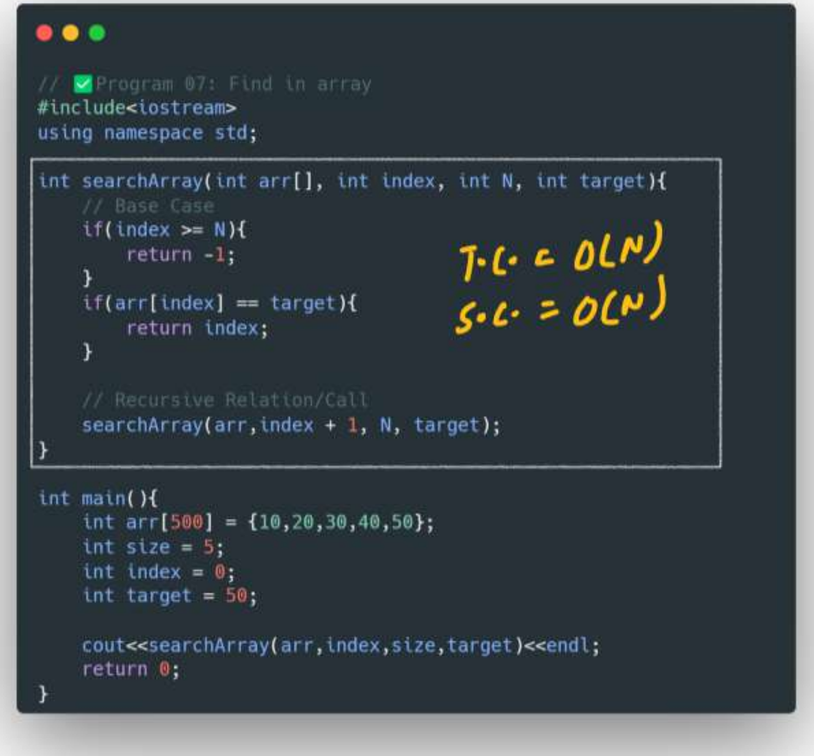
```
if(index >= size)
    return -1;
```

Processing

```
arr[index] == target
    return Index
```

Relation

```
f(arr, index+1, size, target)
```



```
// ✅Program 07: Find in array
#include<iostream>
using namespace std;

int searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return -1;
    }
    if(arr[index] == target){
        return index;
    }

    // Recursive Relation/Call
    searchArray(arr,index + 1, N, target);
}

int main(){
    int arr[500] = {10,20,30,40,50};
    int size = 5;
    int index = 0;
    int target = 50;

    cout<<searchArray(arr,index,size,target)<<endl;
    return 0;
}
```

T.C. = O(N)

S.C. = O(N)

✅Program 08: Print index of all occurrence of target

Target 10

Output 0,2,3

arr

10	20	10	10	30	40
0	1	2	3	4	5

Base case

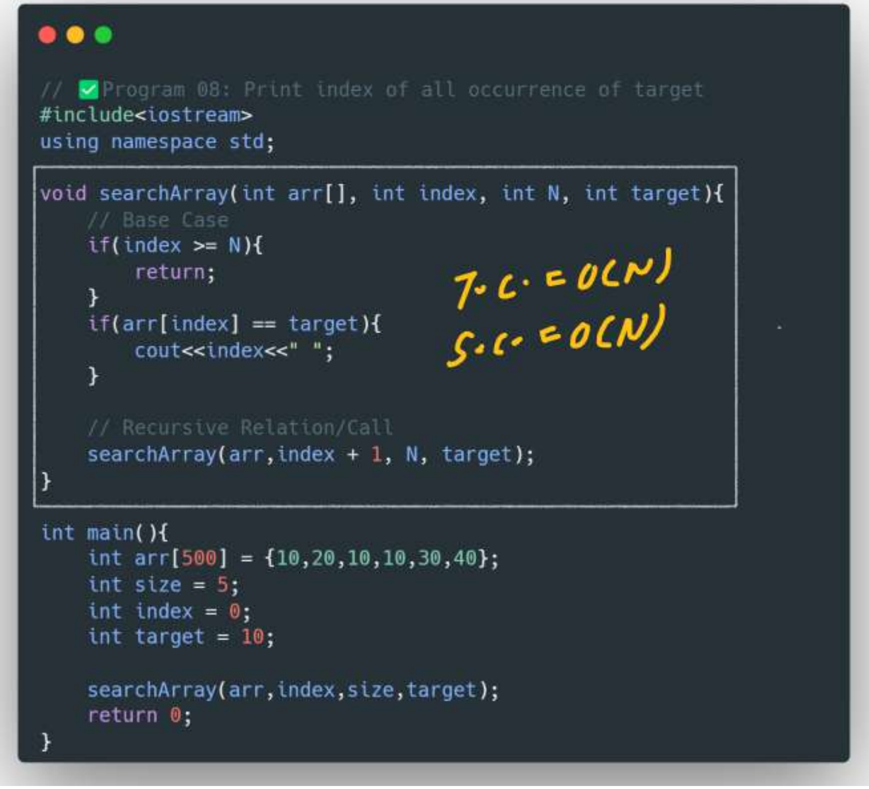
```
if( index >= size)
    return
```

Process

```
if (arr[index] == target)
    cout<< index <<" ";
```

Relation

```
f(arr, index+1, size, target)
```

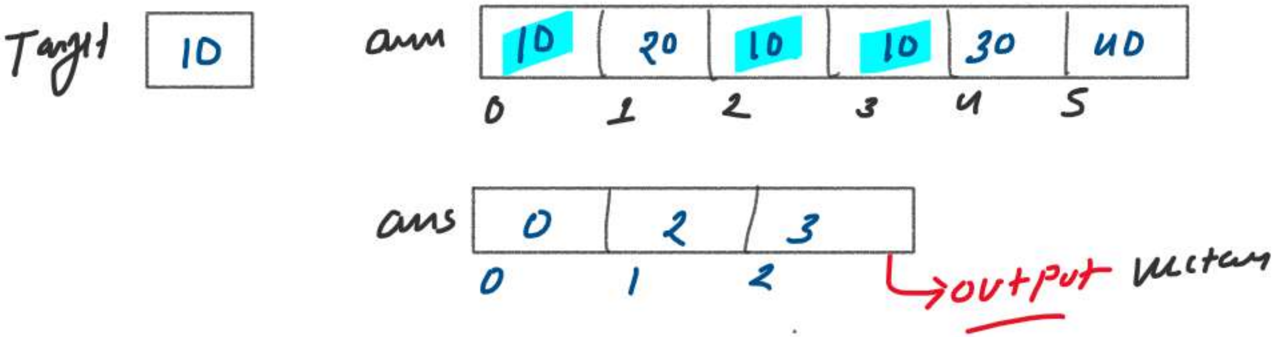


```
// ✅Program 08: Print index of all occurrence of target
#include<iostream>
using namespace std;

void searchArray(int arr[], int index, int N, int target){
    // Base Case
    if(index >= N){
        return;
    }
    if(arr[index] == target){
        cout<<index<<" ";
    }

    // Recursive Relation/Call
    searchArray(arr,index + 1, N, target);
}

int main(){
    int arr[500] = {10,20,10,10,30,40};
    int size = 5;
    int index = 0;
    int target = 10;

    searchArray(arr,index,size,target);
    return 0;
}
```

T.C. = O(N)

S.C. = O(N)

✅Program 09: Return vector with all occurrence of target


Target 10

arr

10	20	10	10	30	40
0	1	2	3	4	5

ans

0	2	3
0	1	2

↳output vector

```
vector <int> ans;
```

BASE case

```
if (index >= size)
    return ans
```

Processing

```
if( arr[index] == target)
      ans.push_back( Index)
```

Relation

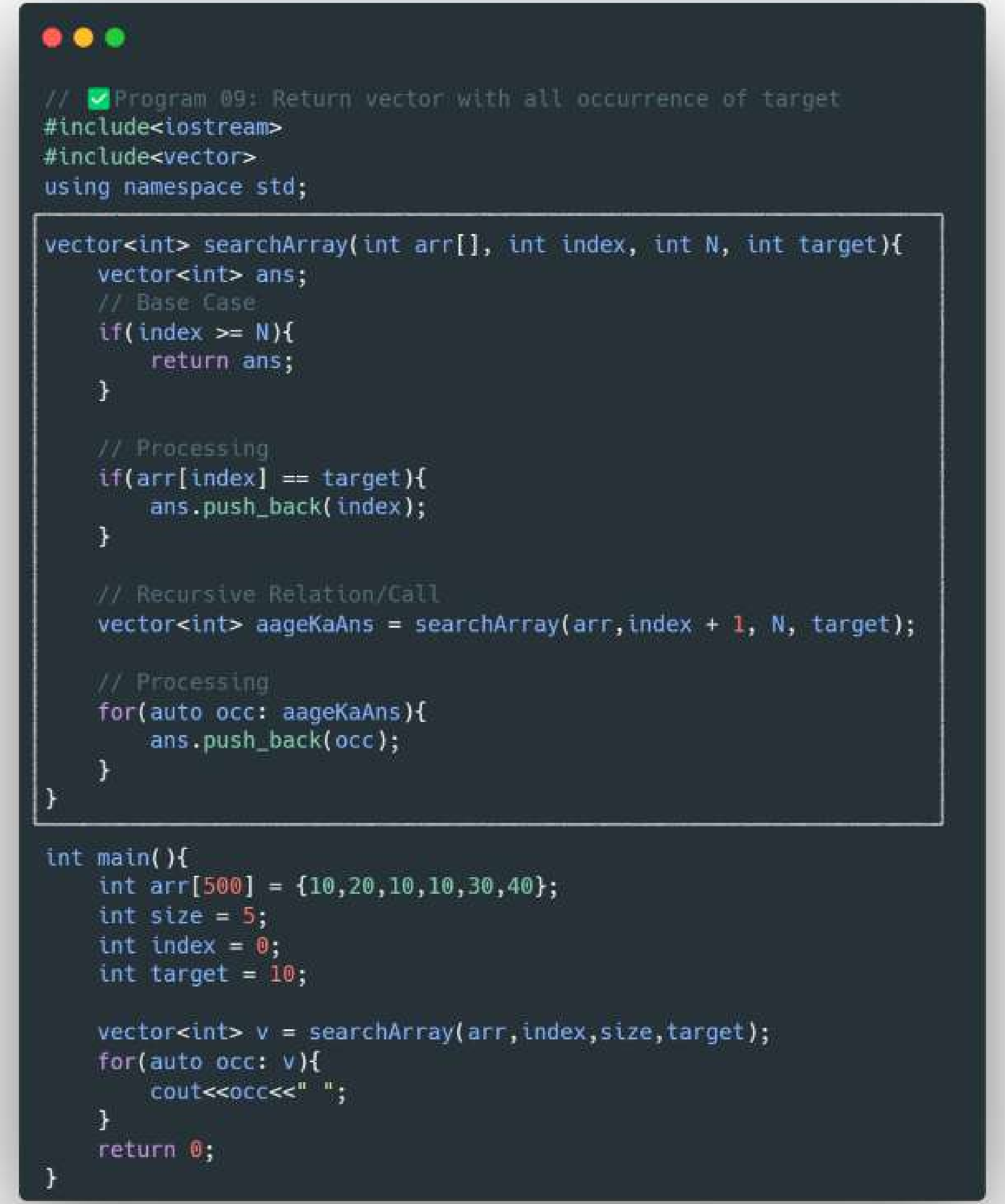
```
vector <int> AAgeAns = f(arr, index+1, size, Target)
```

Processing

```
for ( Auto  occ : AAgeAns ){
      Ans.push_back(occ);
}
```

T.C = O(N)

S.C. = O(N)



```
// ✅Program 09: Return vector with all occurrence of target
#include<iostream>
#include<vector>
using namespace std;

vector<int> searchArray(int arr[], int index, int N, int target){
    vector<int> ans;
    // Base Case
    if(index >= N){
        return ans;
    }

    // Processing
    if(arr[index] == target){
        ans.push_back(index);
    }

    // Recursive Relation/Call
    vector<int> aageKaAns = searchArray(arr,index + 1, N, target);

    // Processing
    for(auto occ: aageKaAns){
        ans.push_back(occ);
    }
}

int main(){
    int arr[500] = {10,20,10,10,30,40};
    int size = 5;
    int index = 0;
    int target = 10;

    vector<int> v = searchArray(arr,index,size,target);
    for(auto occ: v){
        cout<<occ<<" ";
    }
    return 0;
}
```

✅Program 10: Print digits of number

I/P value = 4215

O/P vector V

0	1	2	3
4	2	1	5

BASE case

```
if( value==0 )
    return
```

Processing

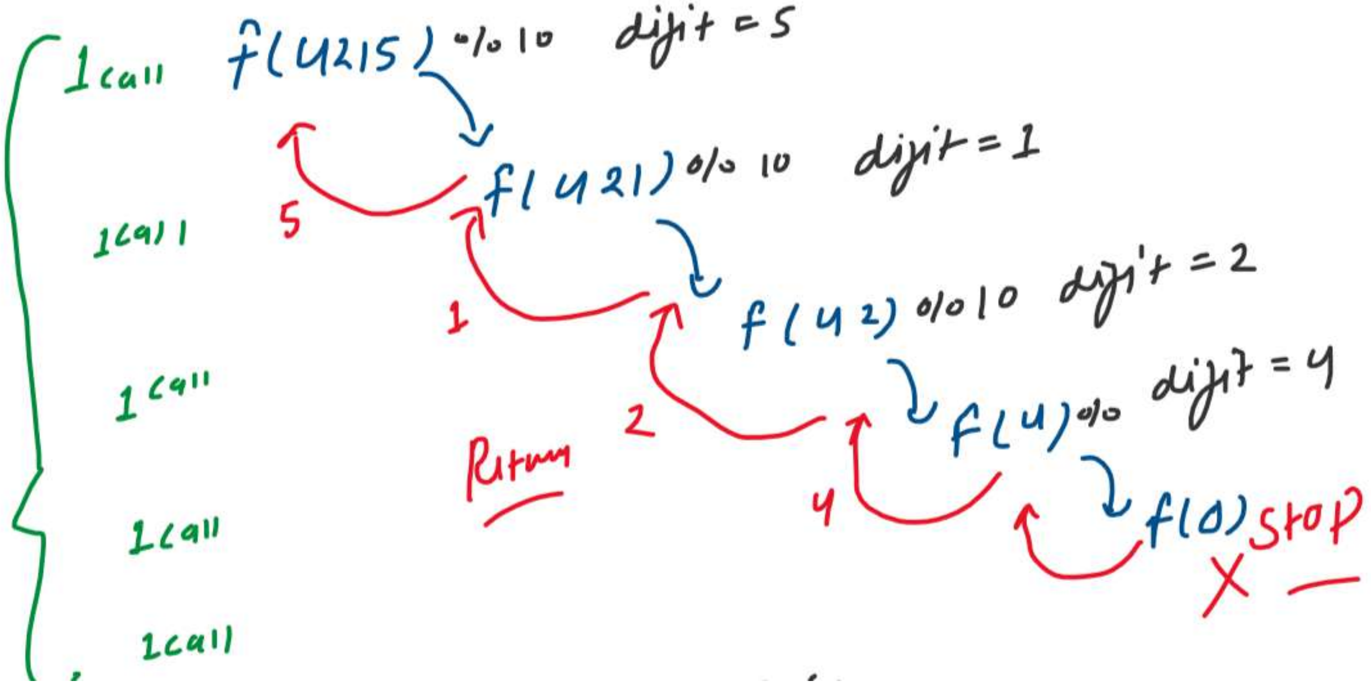
```
digit = value%10
value = value/10
```

Relation

```
f(val, v)
```

Processing

```
v.push_back(digit)
```



BASE case

```
if( value==0 )
    return
```

Processing

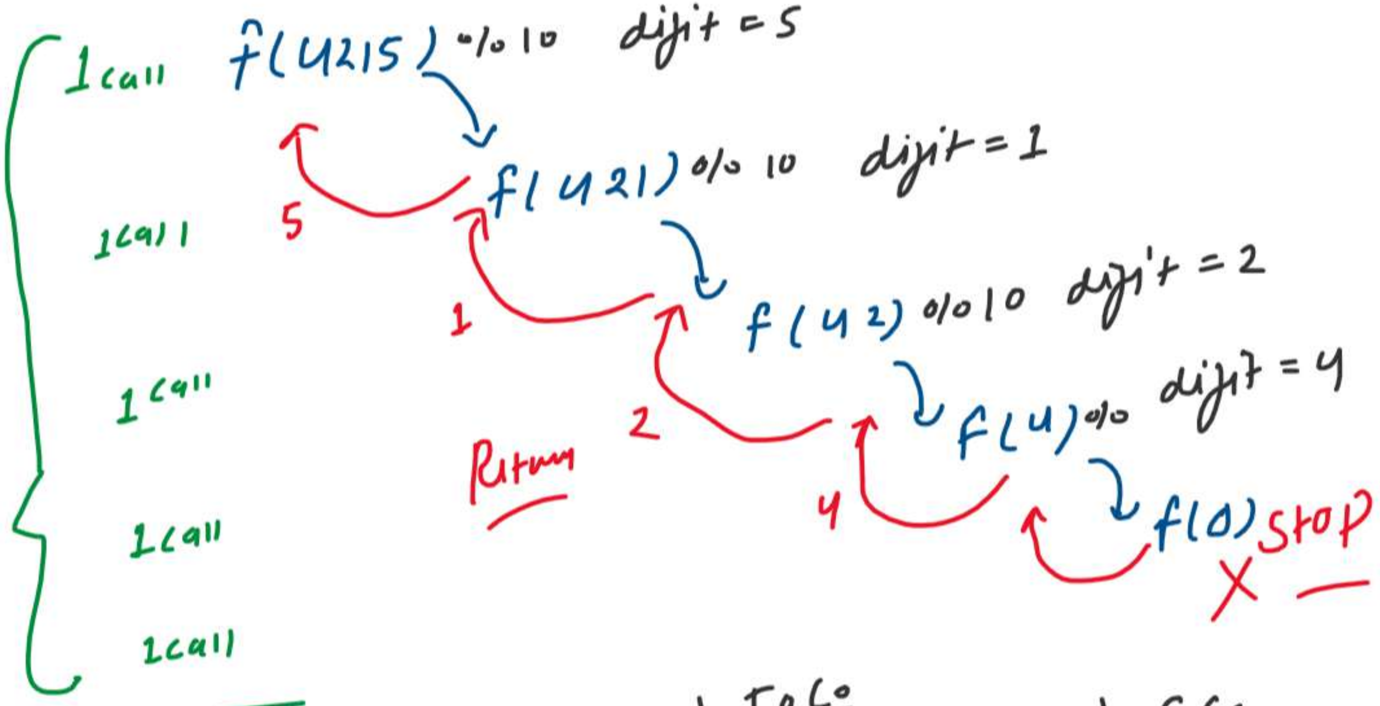
```
digit = value%10
value = value/10
```

Relation

```
f(val, v)
```

Processing

```
v.push_back(digit)
```



Total calls = 5 = 4+1 = N+1

N = No of digits of value

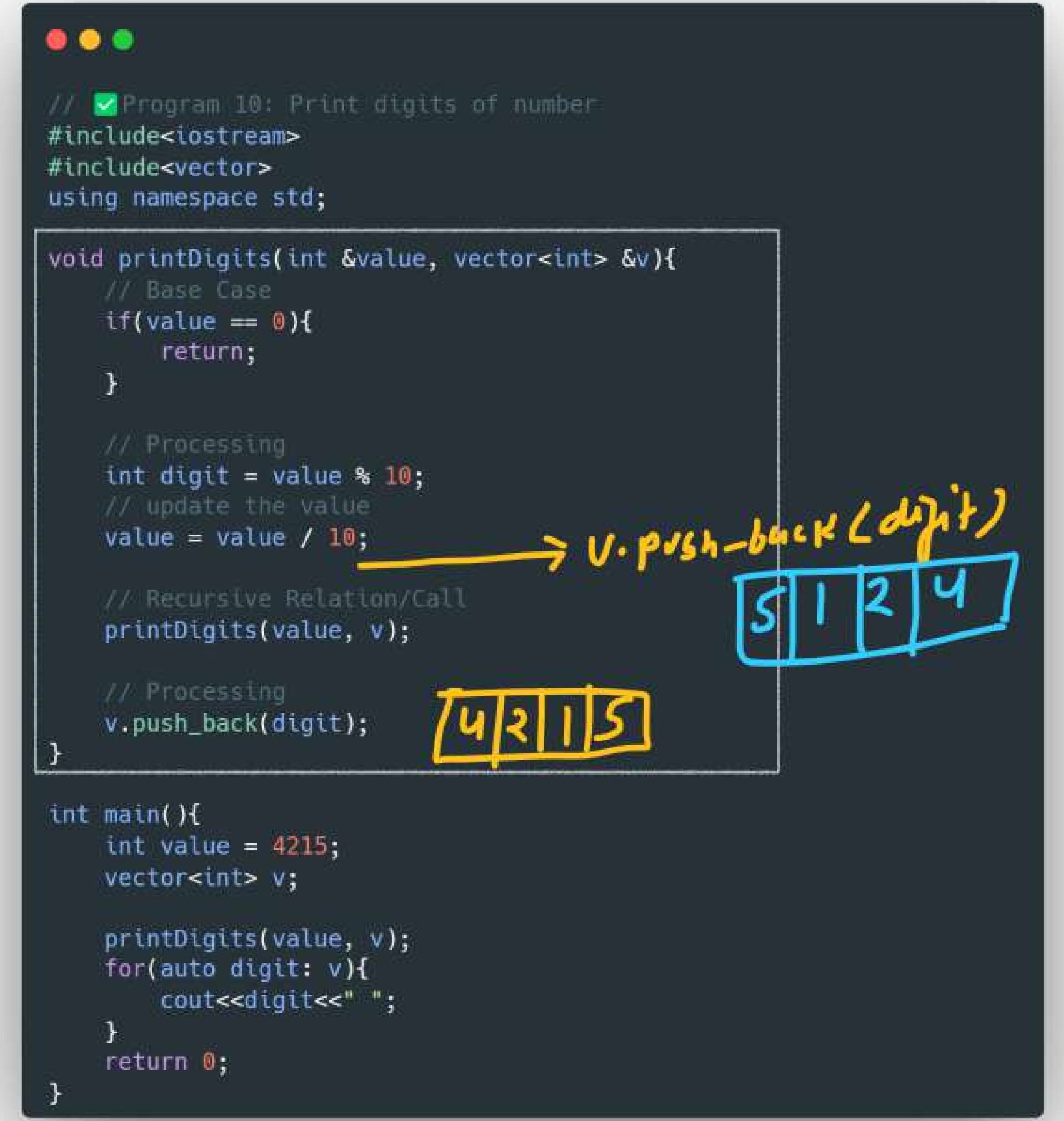
T.C.

$\Rightarrow O(N+1)$

$\Rightarrow O(N)$

S.C.

$\Rightarrow O(N)$



```
// ✅Program 10: Print digits of number
#include<iostream>
#include<vector>
using namespace std;

void printDigits(int &value, vector<int> &v){
    // Base Case
    if(value == 0){
        return;
    }

    // Processing
    int digit = value % 10;
    // update the value
    value = value / 10;

    // Recursive Relation/Call
    printDigits(value, v);

    // Processing
    v.push_back(digit);
}

int main(){
    int value = 4215;
    vector<int> v;

    printDigits(value, v);
    for(auto digit: v){
        cout<<digit<<" ";
    }
    return 0;
}
```