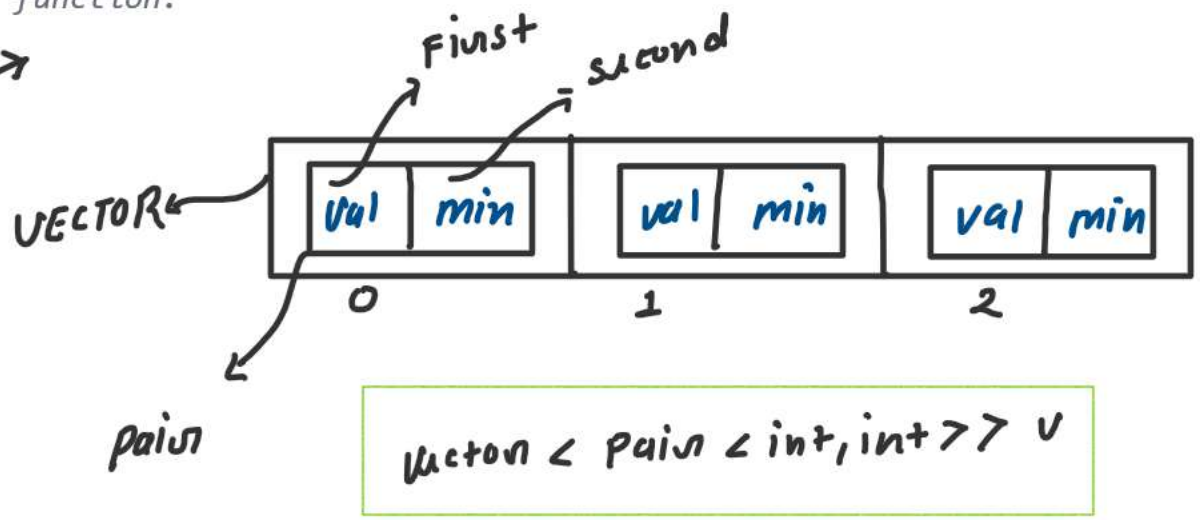
15/11/2023

STACK CLASS - 3

1. Implement a min stack (Leetcode-155)

```
Class MinStack
{
    MinStack()
    push()
    pop()
    top()
    getMin()
}
```

You must implement a solution with O(1) time complexity for each function.




```
vector < pair < int, int >> v
```

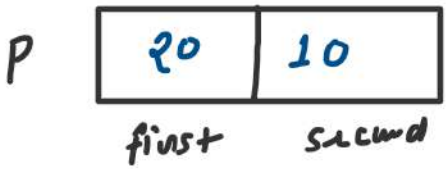
CREATE PAIR

way1

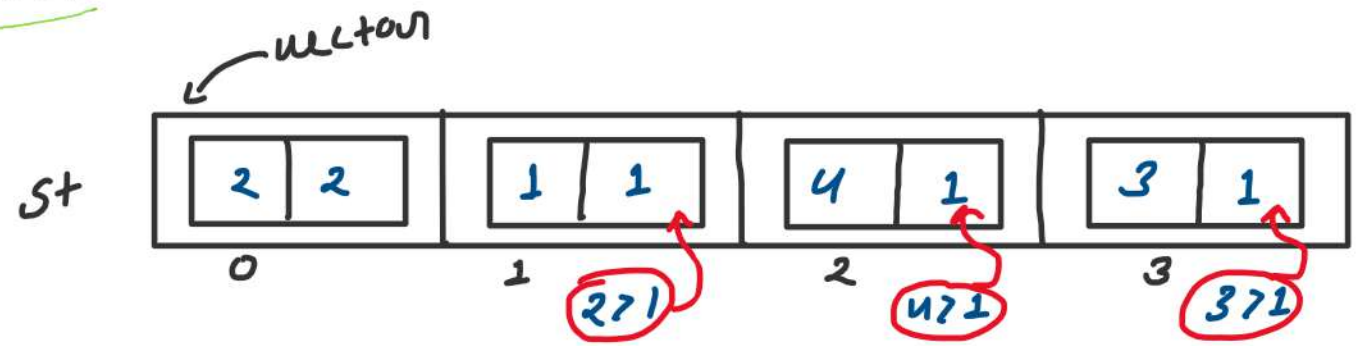
```
pair<int, int> p;
p.first = 20;
p.second = 10;
```

way2

```
pair<int, int> p = make_pair(20, 10);
```



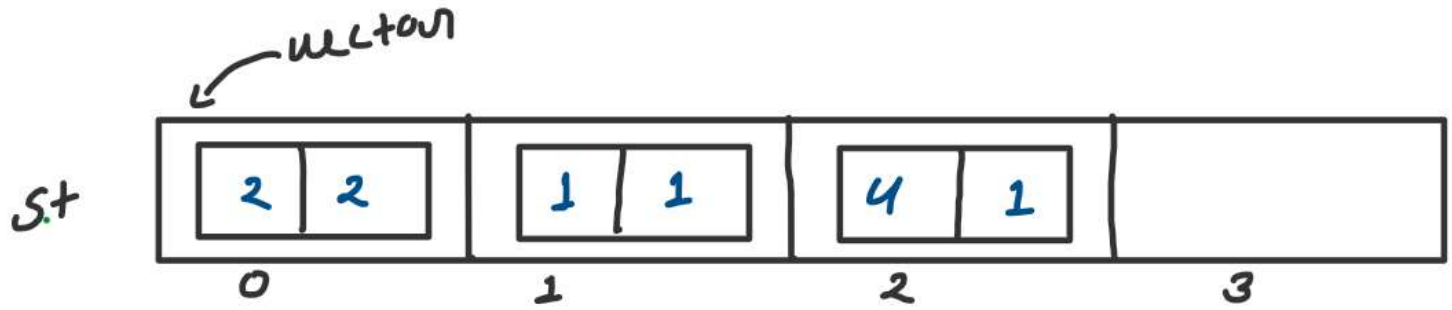
Implement push()




```
void push(int val){
    if(st.empty()){
        pair<int,int> p;
        p.first = val;
        p.second = val;
        st.push_back(p);
    }
    else{
        pair<int,int> p;
        p.first = val;
        int prevmin = st.back().second;
        p.second = min(prevmin, val);
        st.push_back(p);
    }
}
```

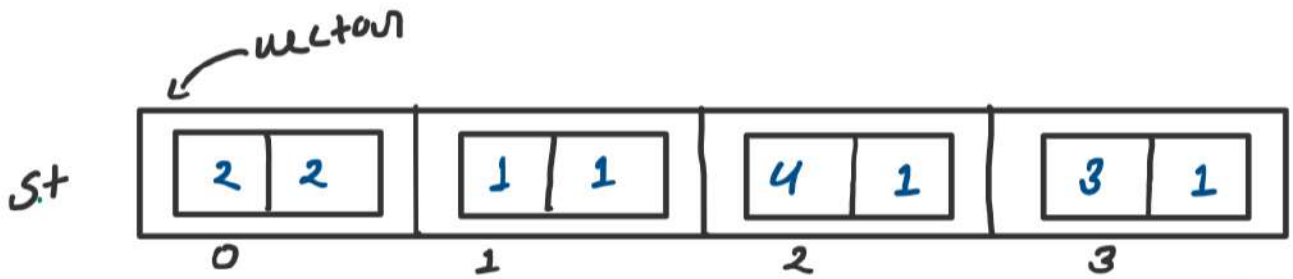
we are inserting first element in vector

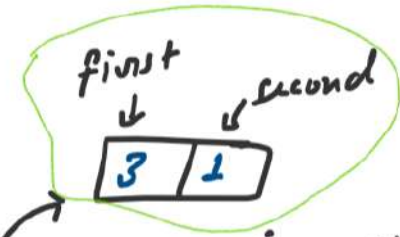
Implement pop()




```
void pop(){
    if(st.empty()){
        cout<<"UNDERFLOW";
    }
    else{
        st.pop_back();
    }
```

Implement top()

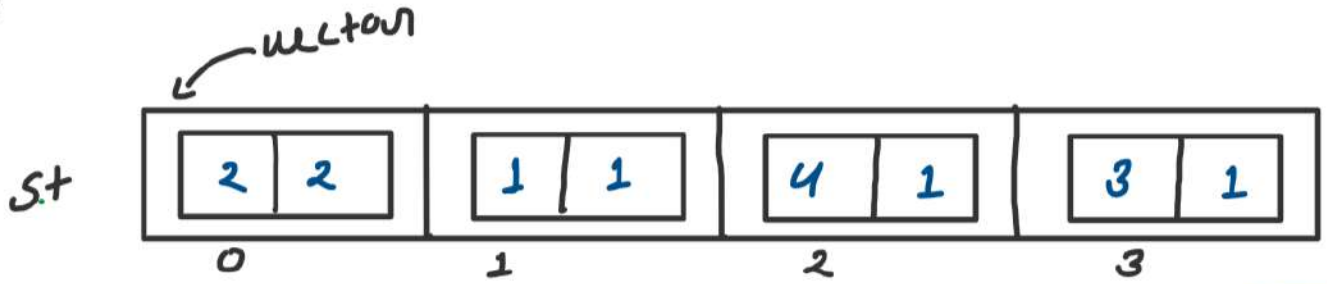


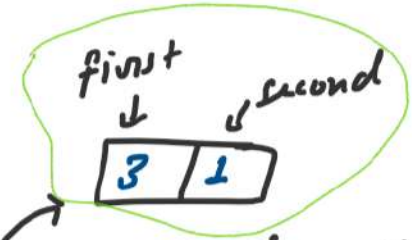



```
int top(){
    if(st.empty()){
        cout<<"UNDERFLOW";
    }
    else{
        pair<int,int> rightmostpair = st.back();
        return rightmostpair.first;
    }
}
```

Output 3

Implement getMin()






```
int getMin() {
    if (st.empty()) {
        cout << "UNDERFLOW";
    }
    else {
        pair<int,int> rightmostpair = st.back();
        return rightmostpair.second;
    }
}
```

Output 1

Complete Code

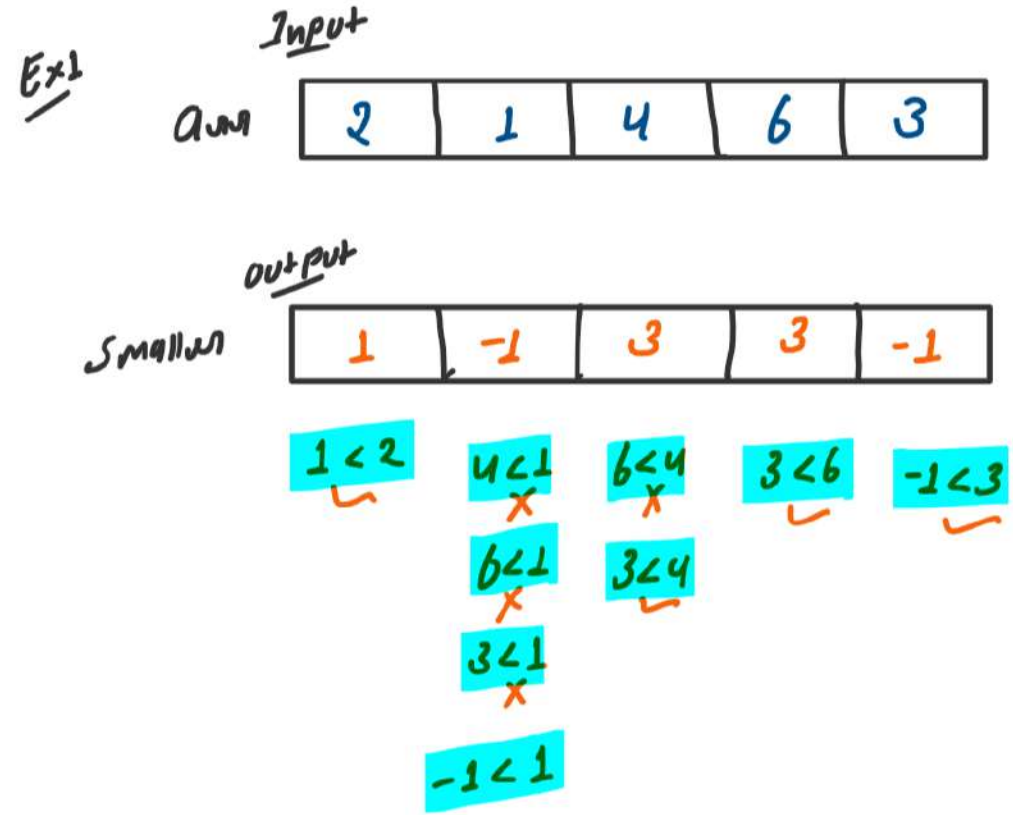
```
class MinStack {
public:
    vector< pair<int,int> > st;
    MinStack() {

    }

    void push(int val) {
        if(st.empty()){
            // It means, we are inserting first element
            pair<int,int> p;
            p.first = val;
            p.second = val;
            st.push_back(p);
        }
        else{
            pair<int,int> p;
            p.first = val;
            int puranaMin = st.back().second;
            p.second = min(puranaMin, val);
            st.push_back(p);
        }
    }

    void pop() {...}

    int top() {...}

    int getMin() {...}
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack* obj = new MinStack();
 * obj->push(val);
 * obj->pop();
 * int param_3 = obj->top();
 * int param_4 = obj->getMin();
 */
```

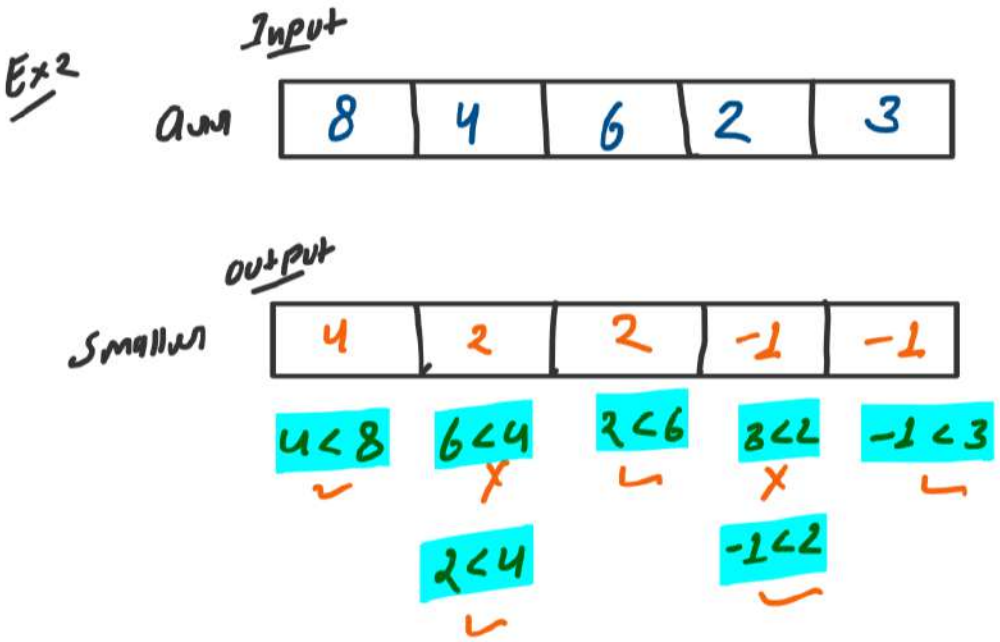
```
void pop() {
    if(st.empty()){
        return;
    }
    else{
        st.pop_back();
    }
}
```

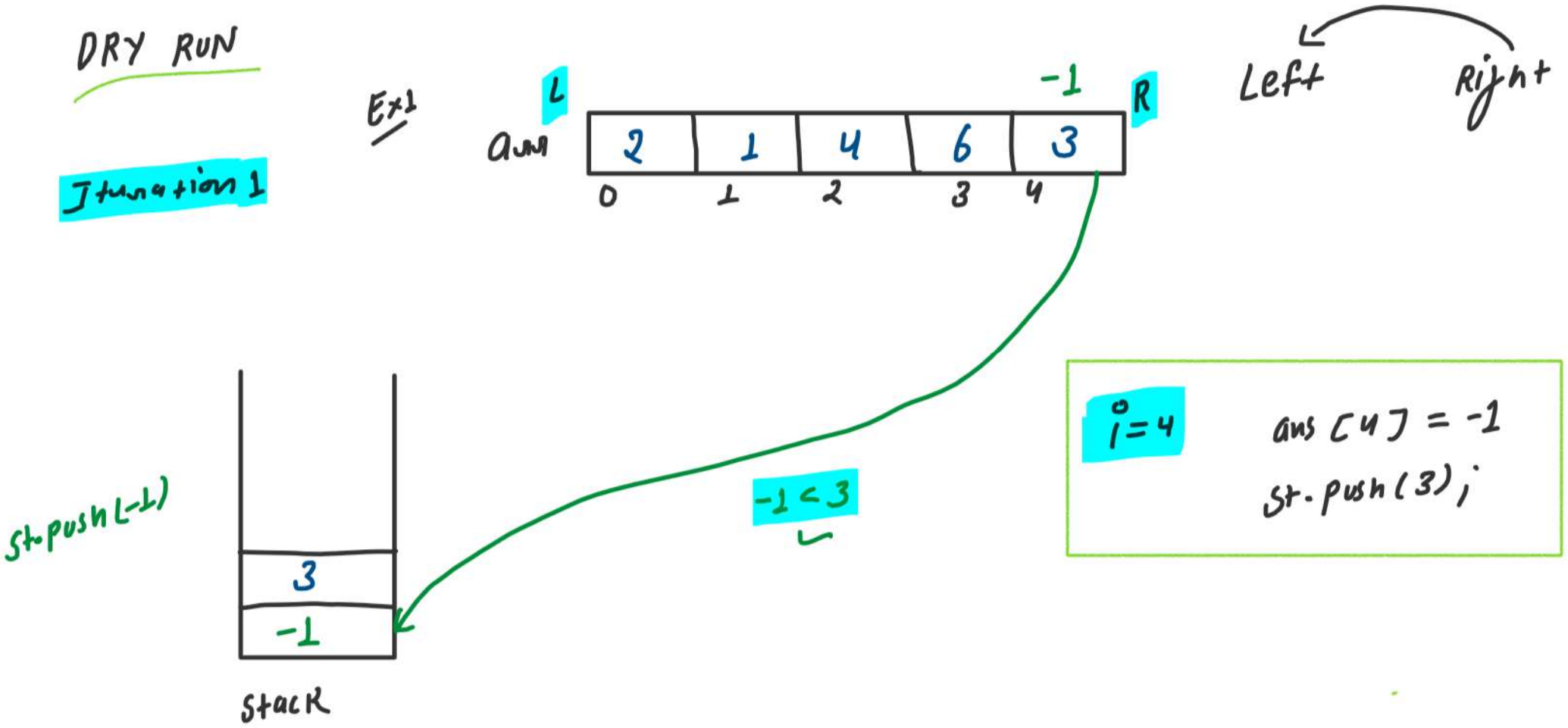
```
int top() {
    if(st.empty()){
        return NULL;
    }
    else{
        pair<int, int> rightMostPair = st.back();
        return rightMostPair.first;
    }
}
```

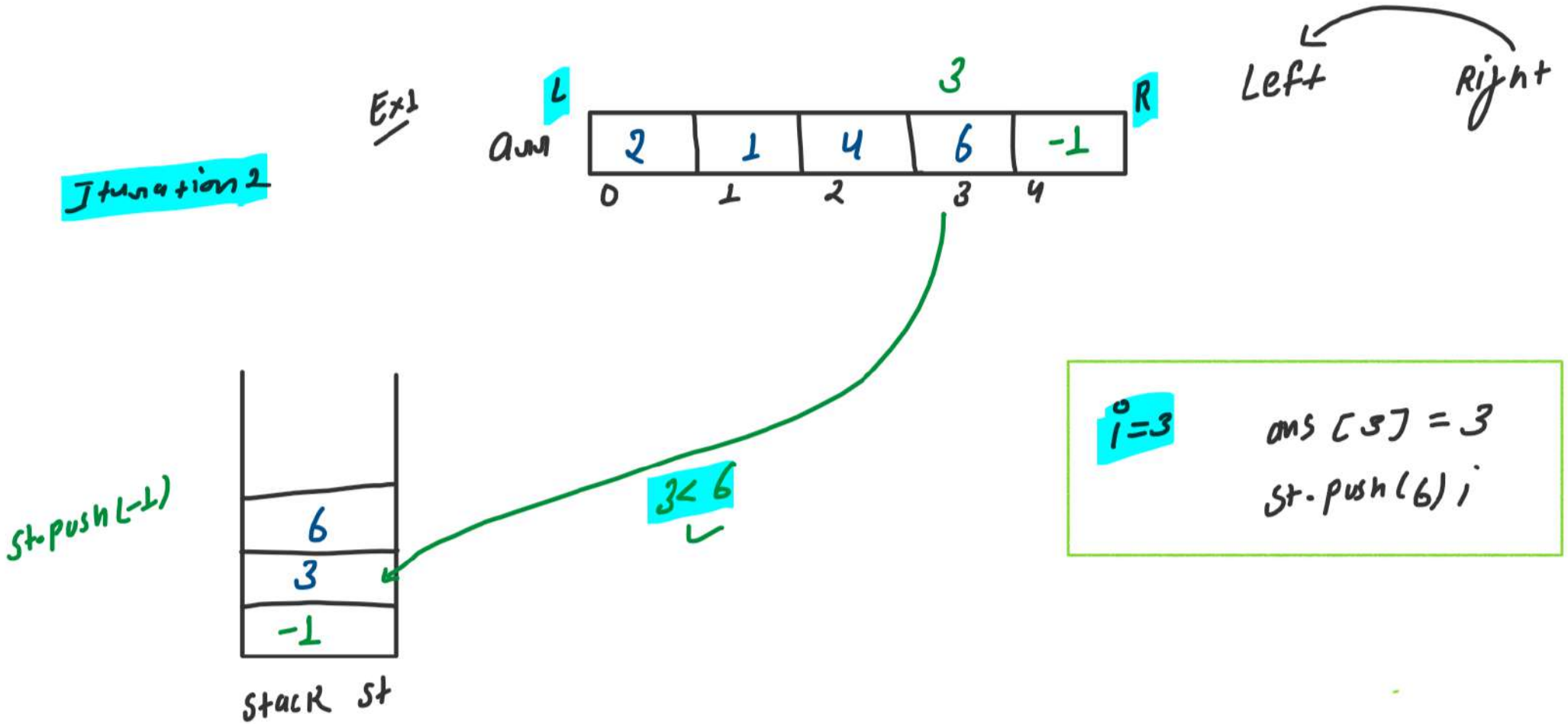
```
int getMin() {
    if(st.empty()){
        return NULL;
    }
    else{
        pair<int,int> rightMostPair = st.back();
        return rightMostPair.second;
    }
}
```

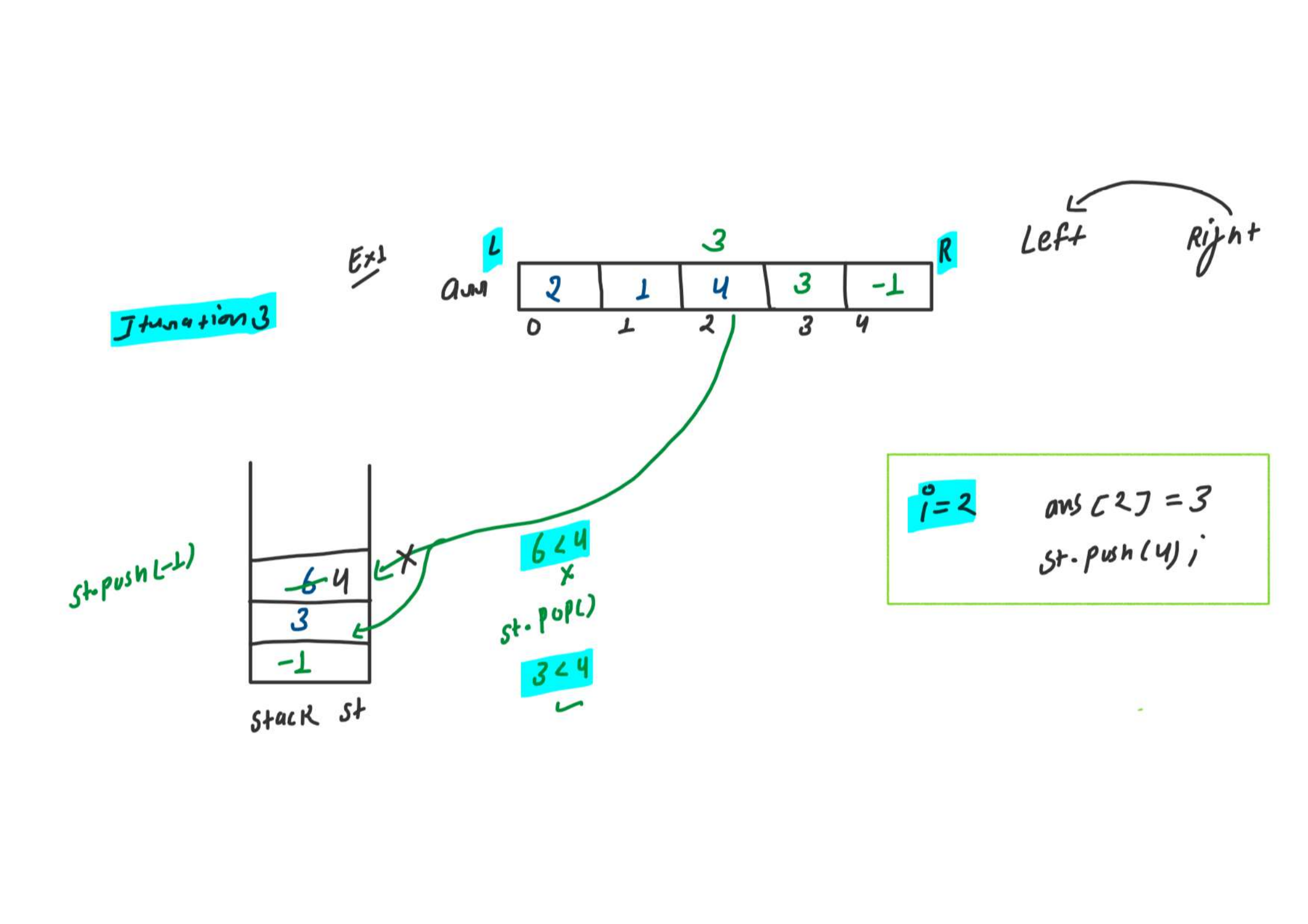
2. Next Smaller Element

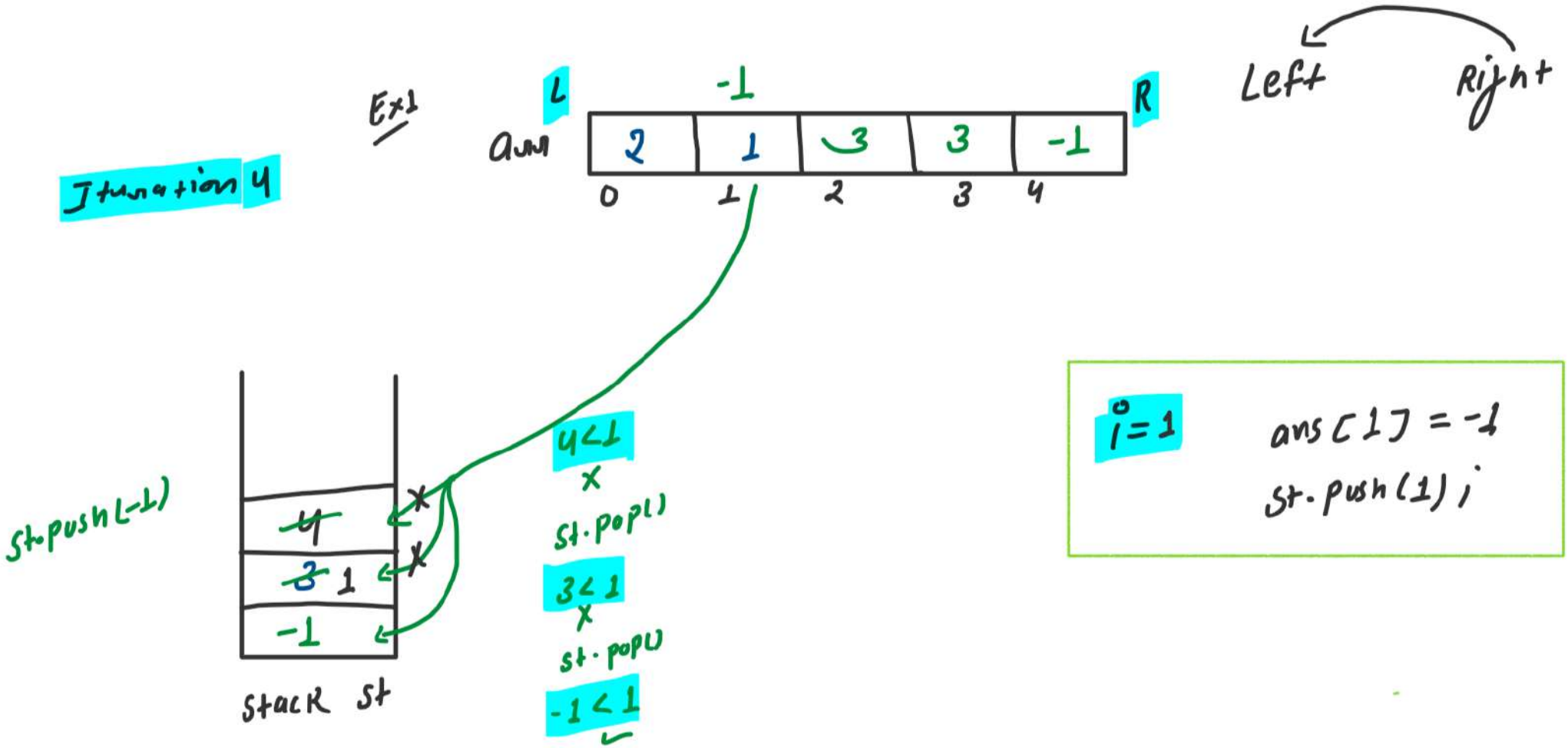


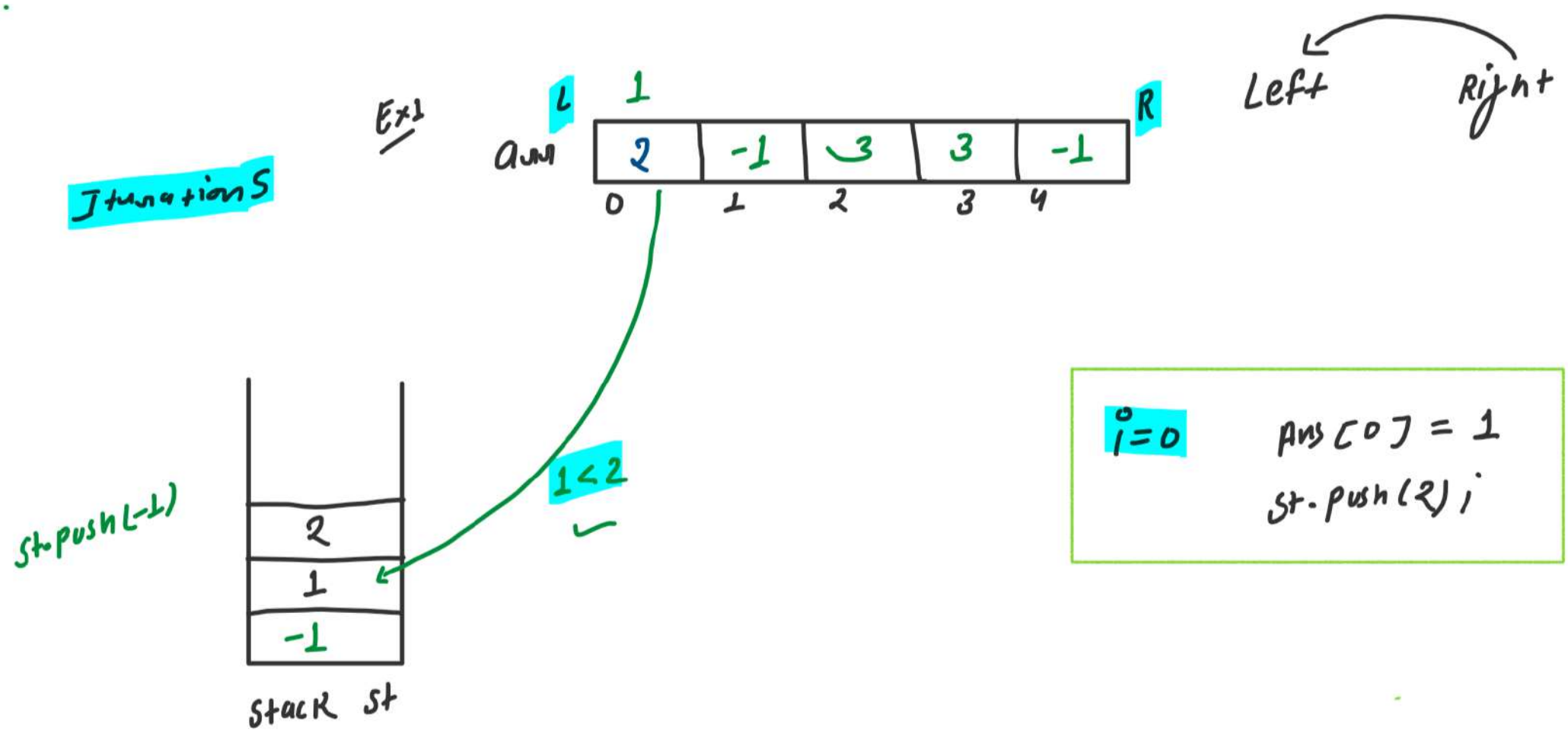



DRY RUN
Ex1
Situation 1
L
-1
R
Left
Right
arr
2
1
4
6
3
0
1
2
3
4
st.push(-1)
-1 < 3
i = 4
ans[4] = -1
st.push(3);
3
-1
stack

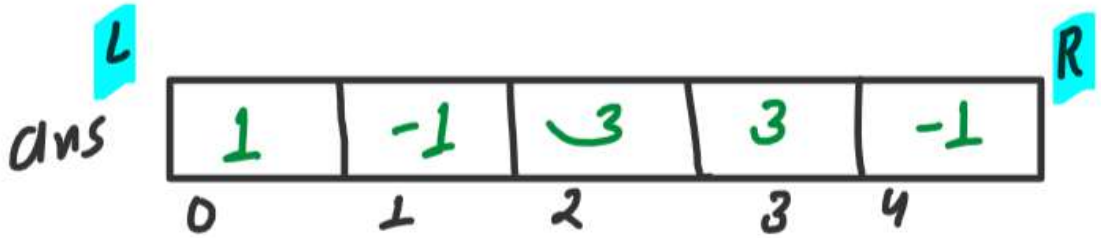

Left
Right
Ex1
arr
L
R
3
2
1
4
6
-1
0
1
2
3
4
Situation 2
st.push(-1)
6
3
-1
stack st
3< 6
i=3
ans[3] = 3
st.push(6) i


Left
Right
Ex1
Situation 3
arr
L
R
3
2
1
4
3
-1
0
1
2
3
4
stack st
st.push(-1)
6 < 4
x
st.pop()
3 < 4
i = 2
ans[2] = 3
st.push(4);


Left
Right
Ex1
L
-1
R
arr
2
1
3
3
-1
0
1
2
3
4
Situation 4
st.push(-1)
4
3
1
-1
stack st
4<1
x
st.pop()
3<1
x
st.pop()
-1<1
i=1
ans[1] = -1
st.push(1)i


Ex1
L
1
R
Arr
2
-1
3
3
-1
0
1
2
3
4
Left
Right
Situations
st.push(-1)
2
1
-1
stack st
1<2
i=0
Ans[0] = 1
st.push(2)i

Final Output



DRY RUN

Ex2

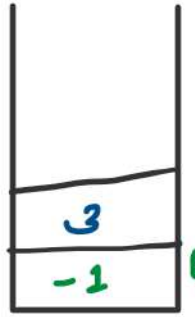
arr

8	4	6	2	3
0	1	2	3	4

-1

Situation 1

-1 < 3



st.push(-1)

Stack st

i = 4

ans[4] = -1

st.push(3) i

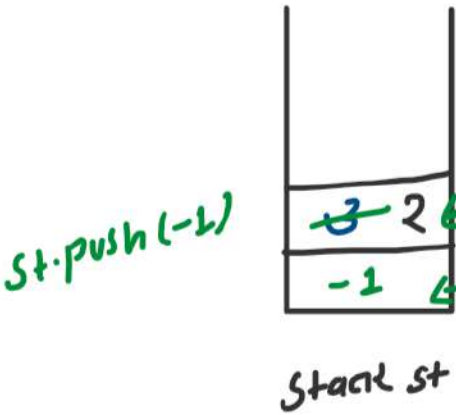
Ex2

arr

8	4	6	2	-1
0	1	2	3	4

-1

Situation2



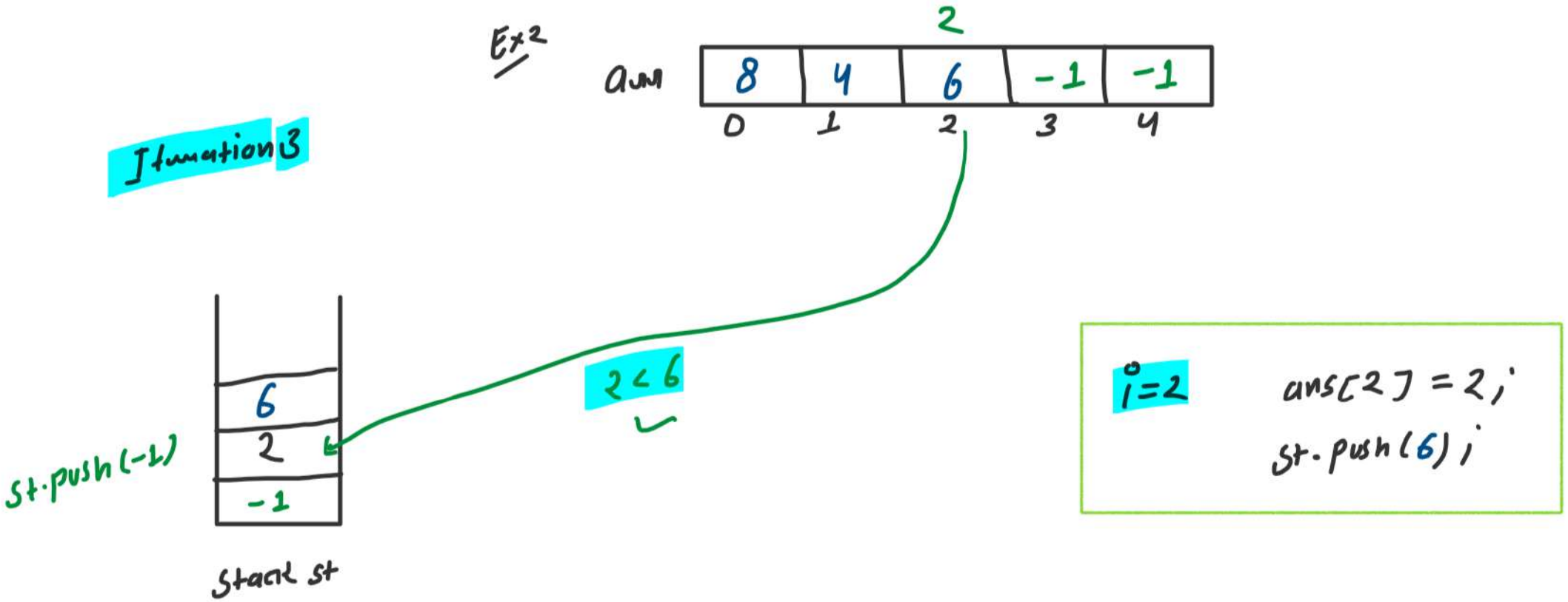
3 < 2
x
st.pop()

-1 < 2
✓

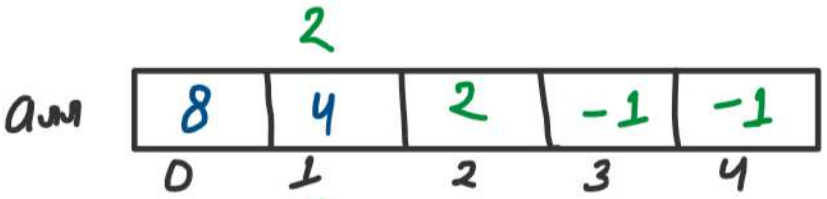
i = 3

ans[3] = -1;

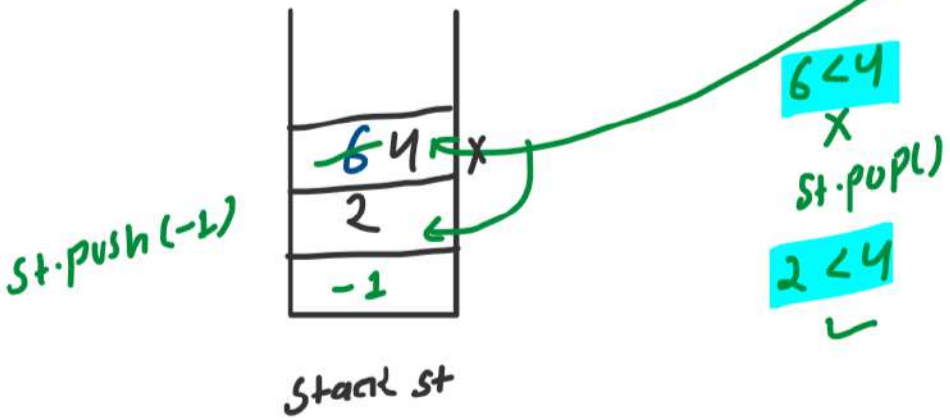
st.push(2);


Ex2
arr
8
4
6
-1
-1
0
1
2
3
4
2
Situation 3
6
2
-1
st.push(-1)
Stack st
2 < 6
i=2
ans[2] = 2;
st.push(6);

Ex2



Situation 4

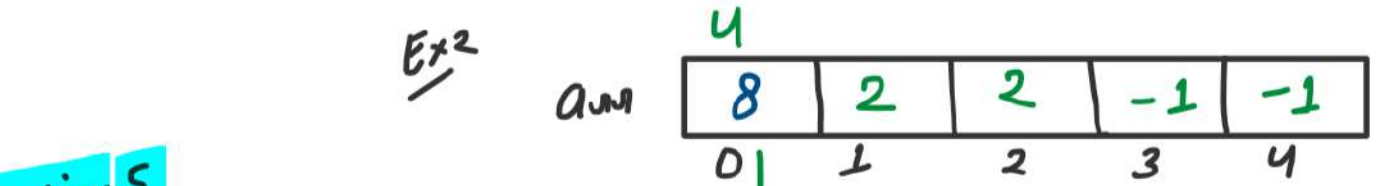


i = 1

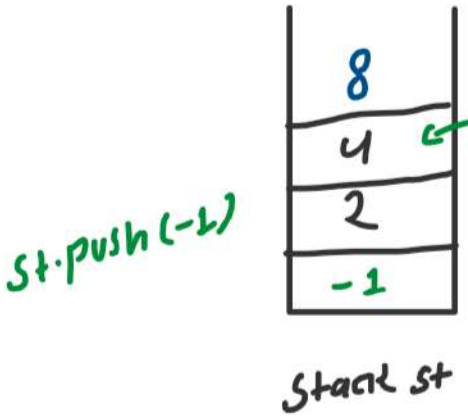
ans[1] = 2;

St.push(4);

Ex2



Situations



4<8 ✓

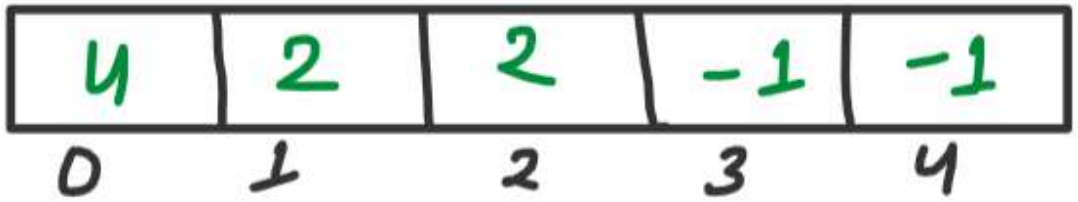
i=0

ans[0] = 4;

st.push(8);

Final Output

ans



Complete code

```cpp
// 📁 Problem 2: Next smaller element
#include<iostream>
#include<stack>
#include<vector>
using namespace std;

void nextSmallerElement(int *arr, int &size, vector<int> &ans){
    stack<int> st;
    st.push(-1);

    for(int i=size-1; i>=0; i--){
        int currElement = arr[i];

        // Find ans to currElement from stack
        while (st.top() >= currElement)
        {
            st.pop();
        }

        // Yaha tak me tabhi pahuncha hoga jab st.top() < currElement
        ans[i] = st.top();
        st.push(currElement);
    }
}

int main(){
    int arr[5] = {8, 4, 6, 2, 3};
    int size = 5;
    vector<int> ans(size);

    nextSmallerElement(arr, size, ans);

    for(auto element: ans){
        cout<< element << " ";
    }
    cout<<endl;

    return 0;
}
```

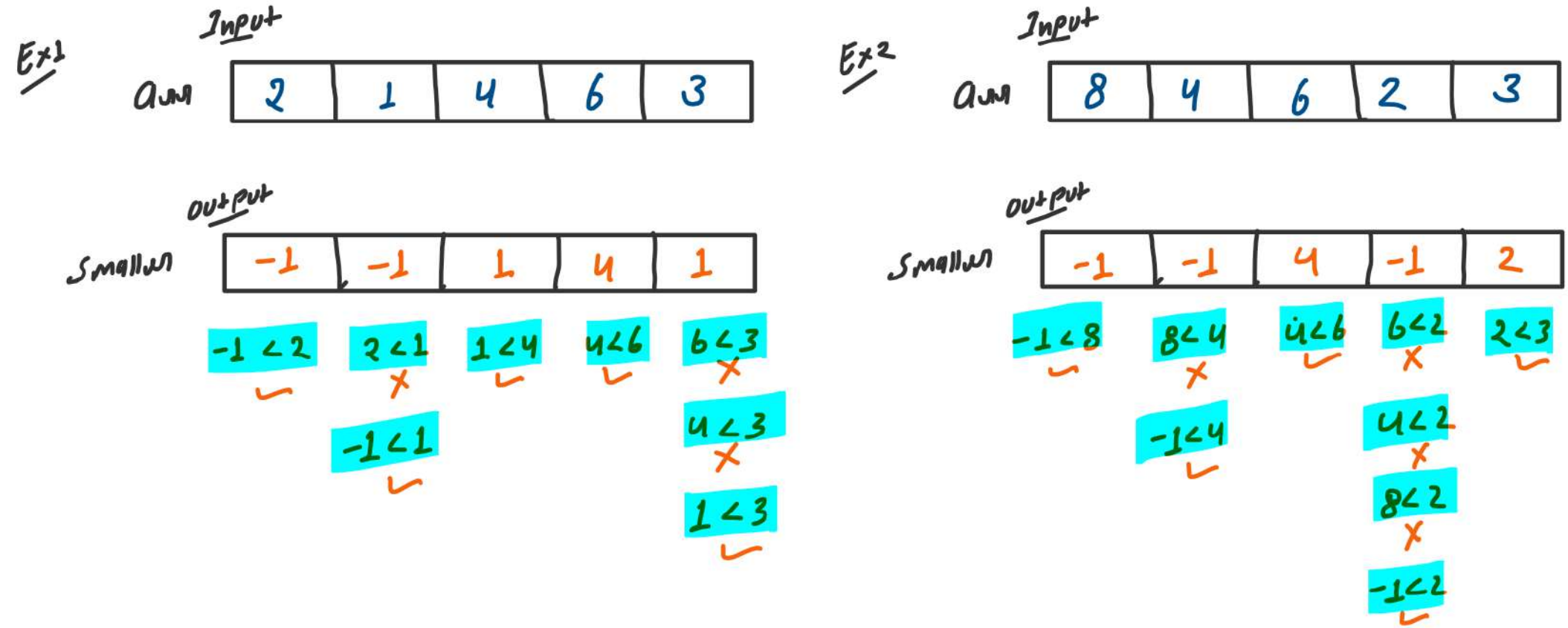
T.C. = O(N), where N is size of array.

Why use stack?

↳ To track the ans for each element of array.

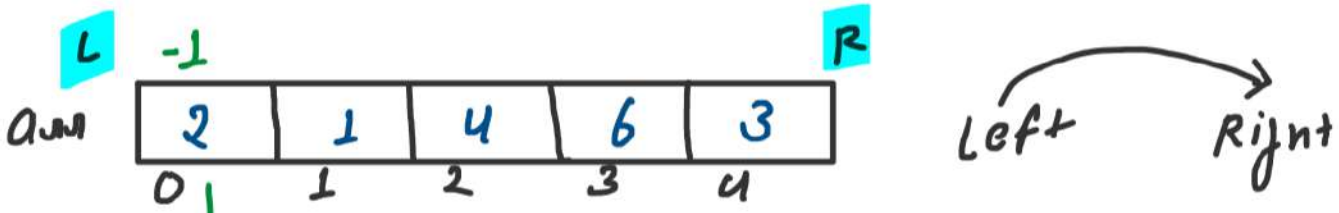
S.C. = O(N), where N is number of elements in stack.

3. Previous Smaller Element

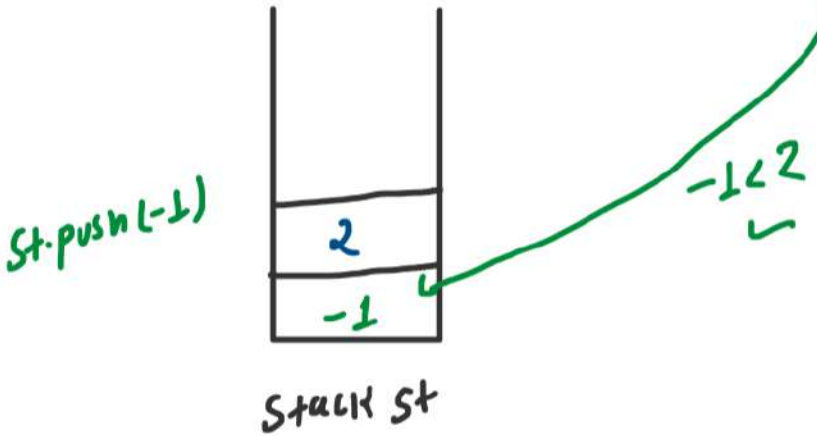


DRY RUN

Ex1



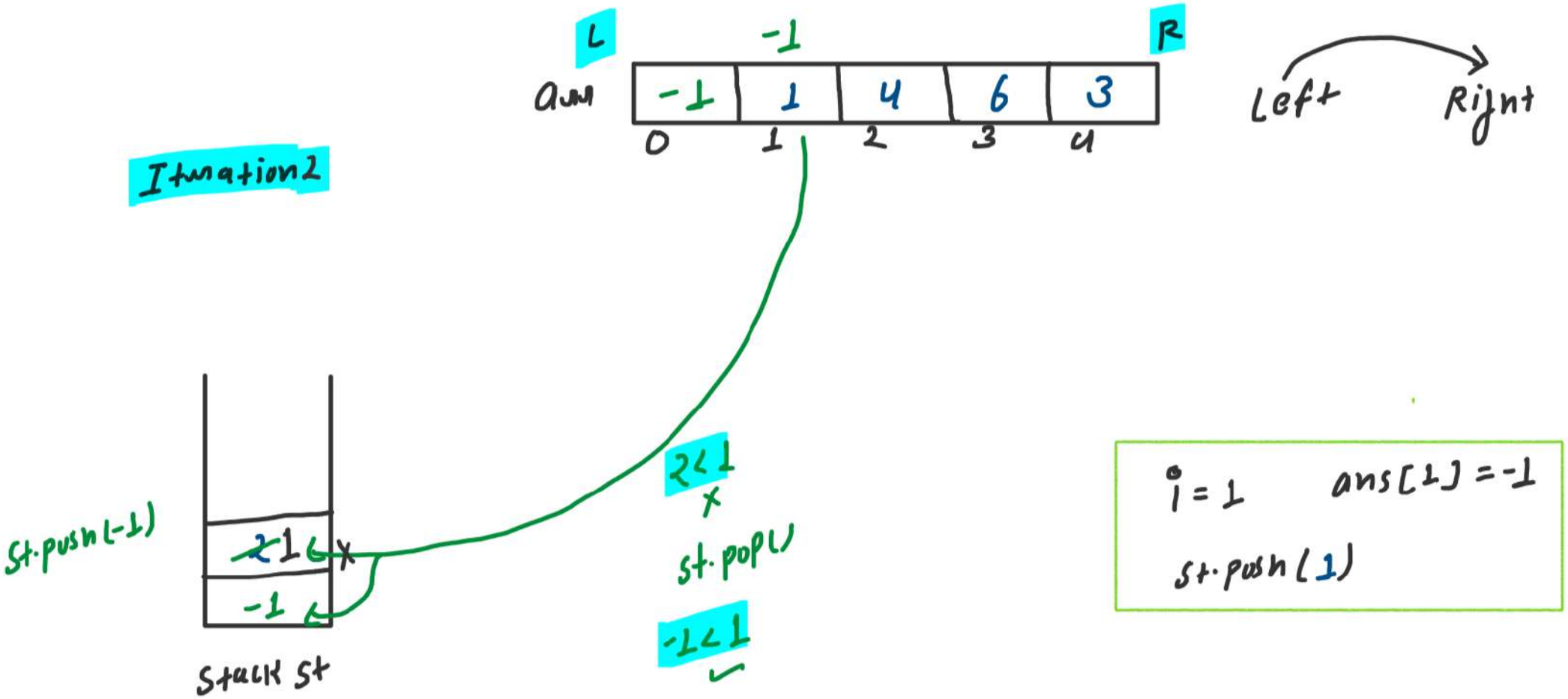
Situation 1



i = 0

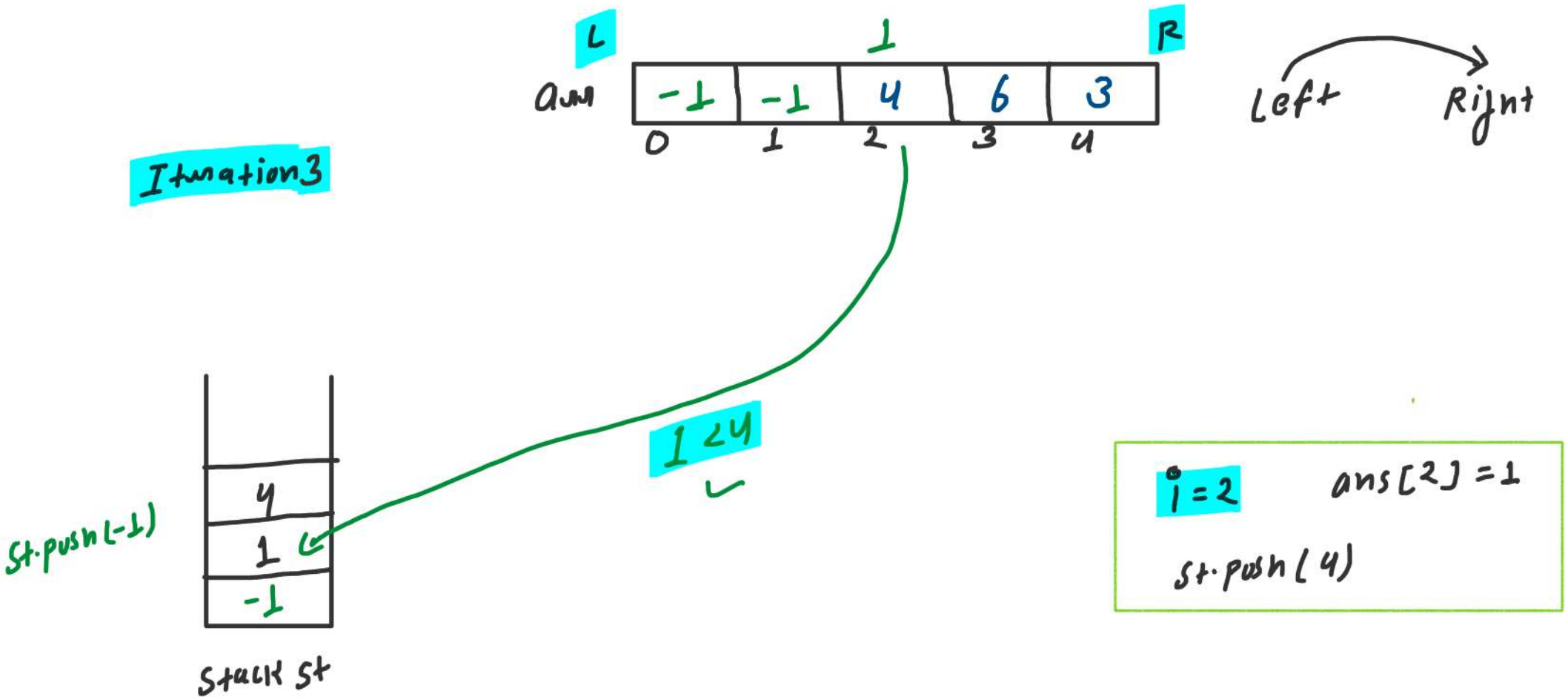
ans[0] = -1

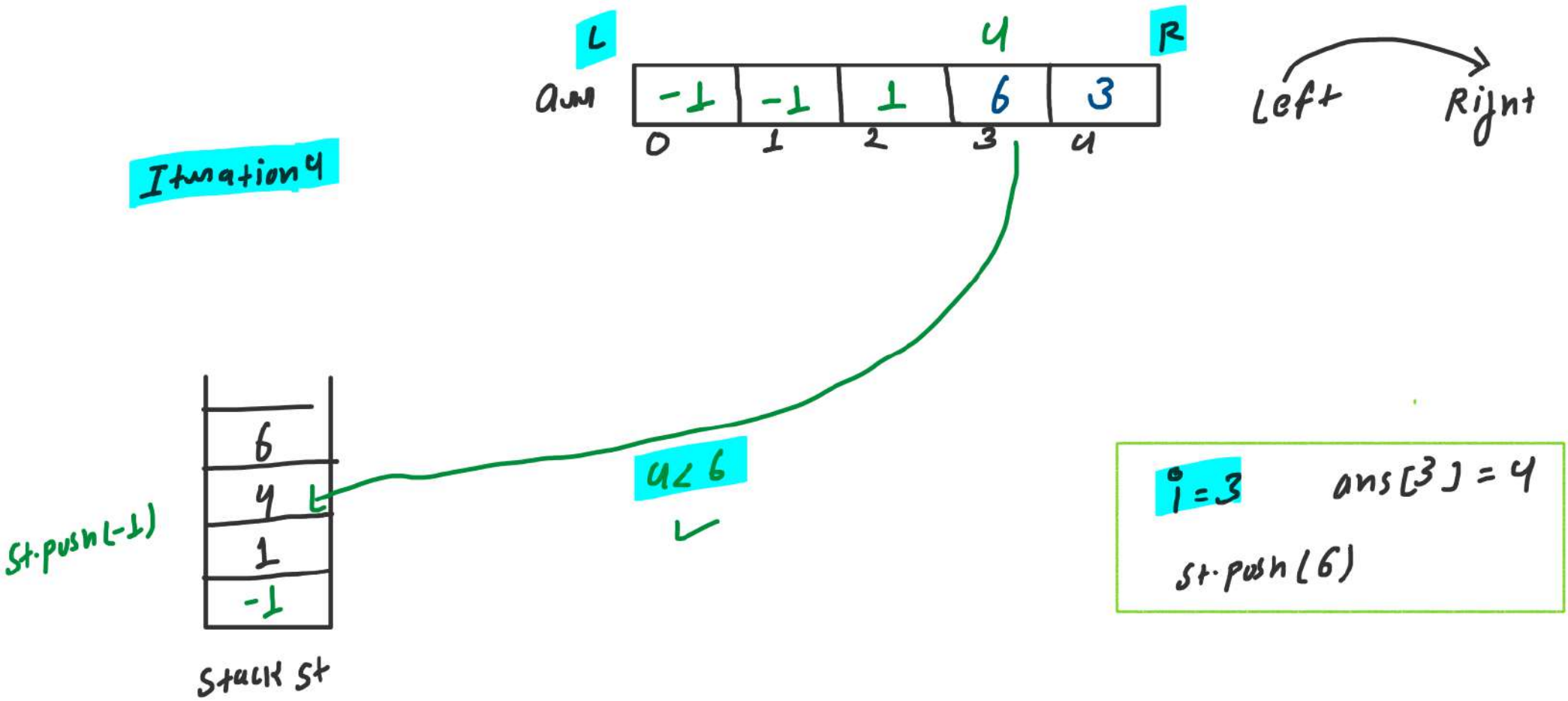
st.push(2)

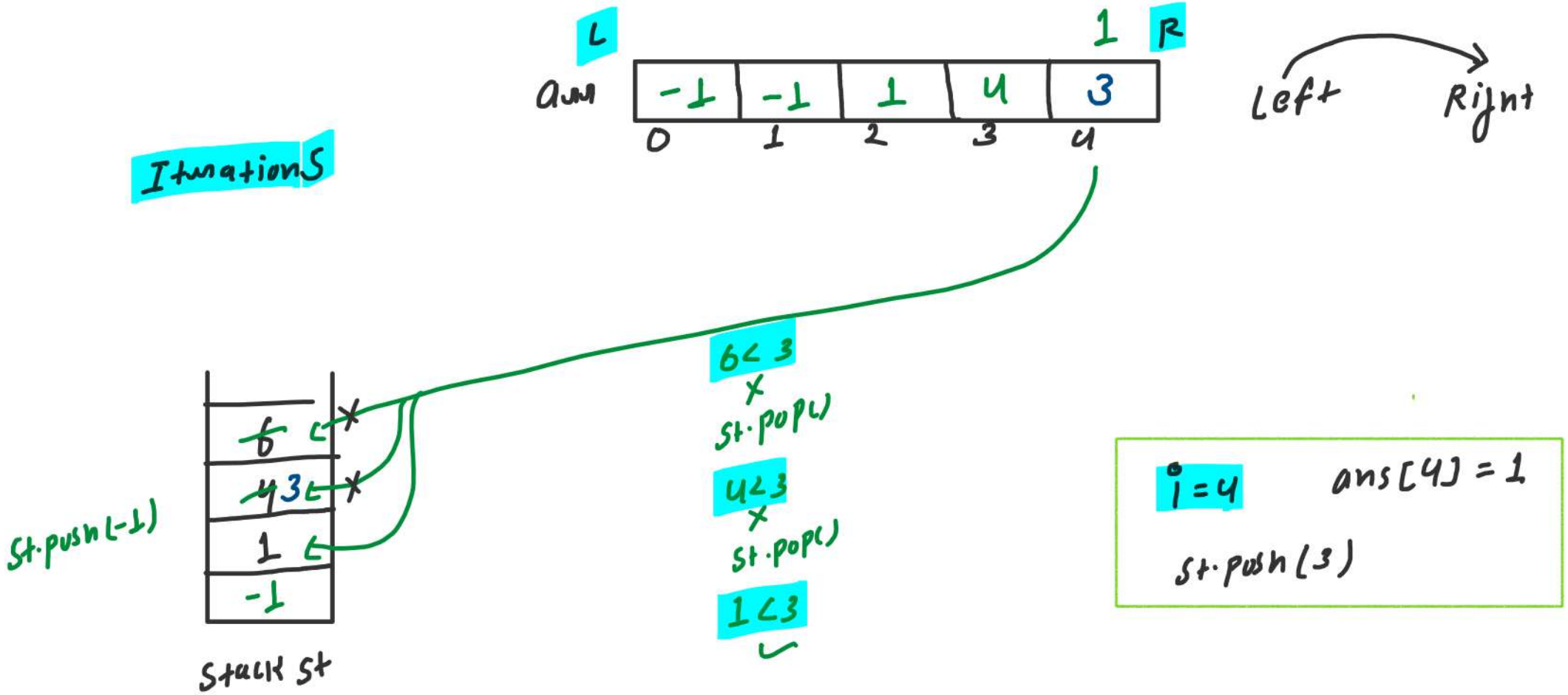


i = 1 ans[1] = -1

st.push(1)

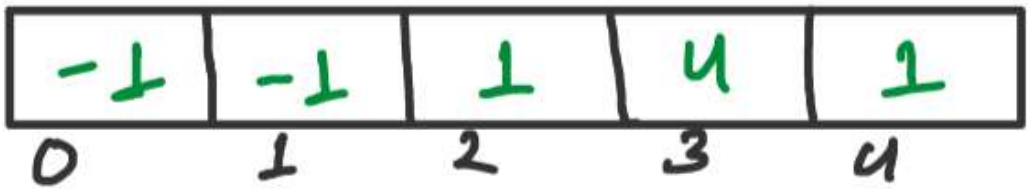

Situation3
L
1
R
ans
-1
-1
4
6
3
0
1
2
3
4
Left
Right
1 < 4
St.push(-1)
4
1
-1
Stack st
i = 2
ans[2] = 1
st.push(4)


Situation 4
L
4
R
arr
-1
-1
1
6
3
0
1
2
3
4
Left
Right
6
4
1
-1
st.push(-1)
Stack st
4 < 6
i = 3
ans[3] = 4
st.push(6)


L
1
R
arr
-1
-1
1
4
3
0
1
2
3
4
Left
Right
Situations
6 < 3
X
St.pop()
4 < 3
X
St.pop()
1 < 3
St.push(-1)
6
4 3
1
-1
Stack St
i = 4
ans[4] = 1
St.push(3)

Final Output

Ans



Complete Code

```cpp
// 📂 Problem 3: Previous smaller element
#include<iostream>
#include<stack>
#include<vector>
using namespace std;

void prevSmallerElement(int *arr, int &size, vector<int> &ans){
    stack<int> st;
    st.push(-1);

    for(int i=0; i<size; i++){
        int currElement = arr[i];

        // Find ans to currElement from stack
        while (st.top() >= currElement)
        {
            st.pop();
        }

        // Ynha tak me tabhi pahuncha hoga jab st.top() < currElement
        ans[i] = st.top();
        st.push(currElement);
    }
}

int main(){
    int arr[5] = {8, 4, 6, 2, 3};
    int size = 5;
    vector<int> ans(size);

    prevSmallerElement(arr, size, ans);

    for(auto element: ans){
        cout<< element << " ";
    }
    cout<<endl;

    return 0;
}
```

T.C. = O(N), where N is size of array.

Why use stack?

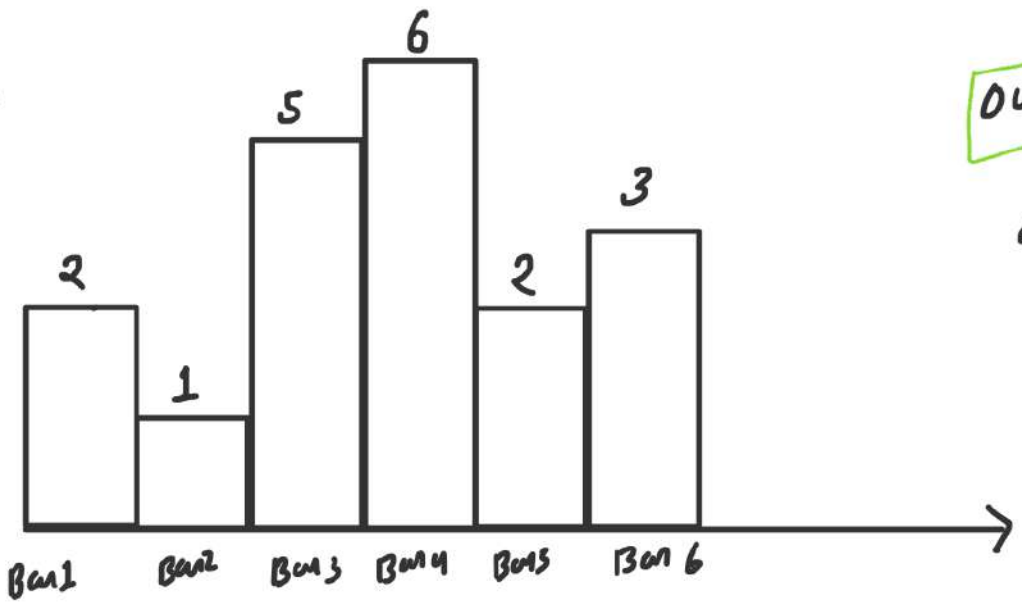
↳ To track the ans for each element of array.

S.C. = O(N), where N is number of elements in stack.

4. Largest Rectangle Area in Histogram (Leetcode-84)

Ex

Input



Output = 10

Condition

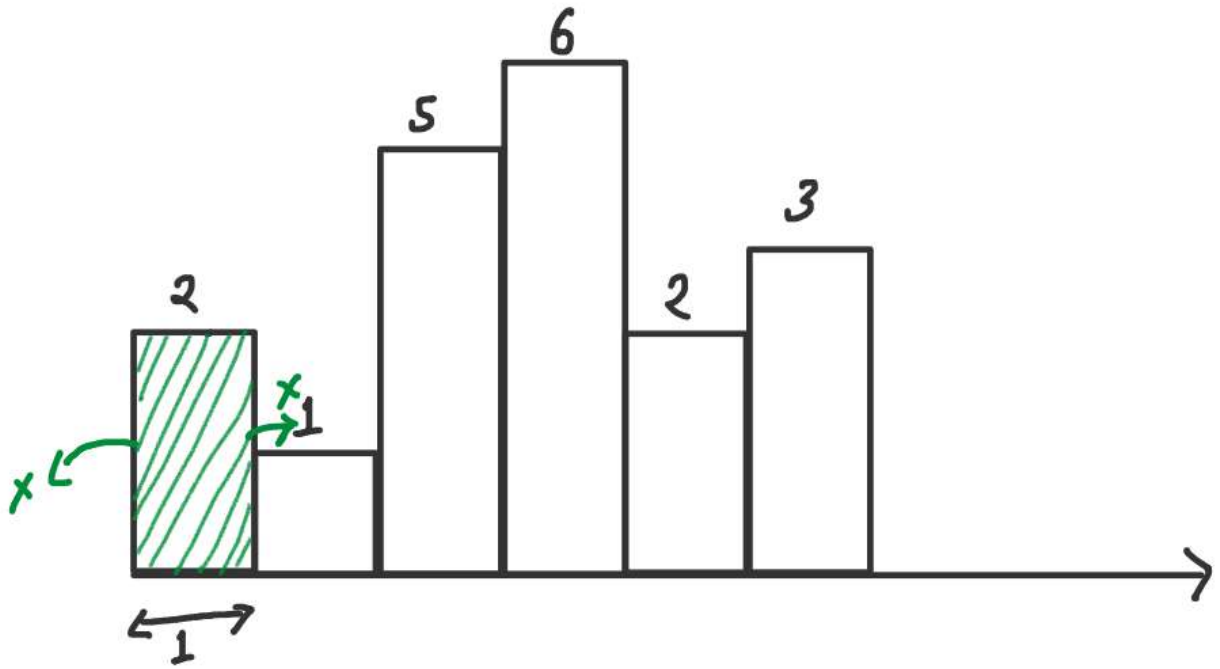
① 1 BAR APNE Right YA Left ME Tabhi Mouu Kar SKTa Hai jab USKE Adj. Bar Equal YA USSE Greater than Ho.

DRY RUN

AREA = width x hight

$A_1 = 2 \times 1$

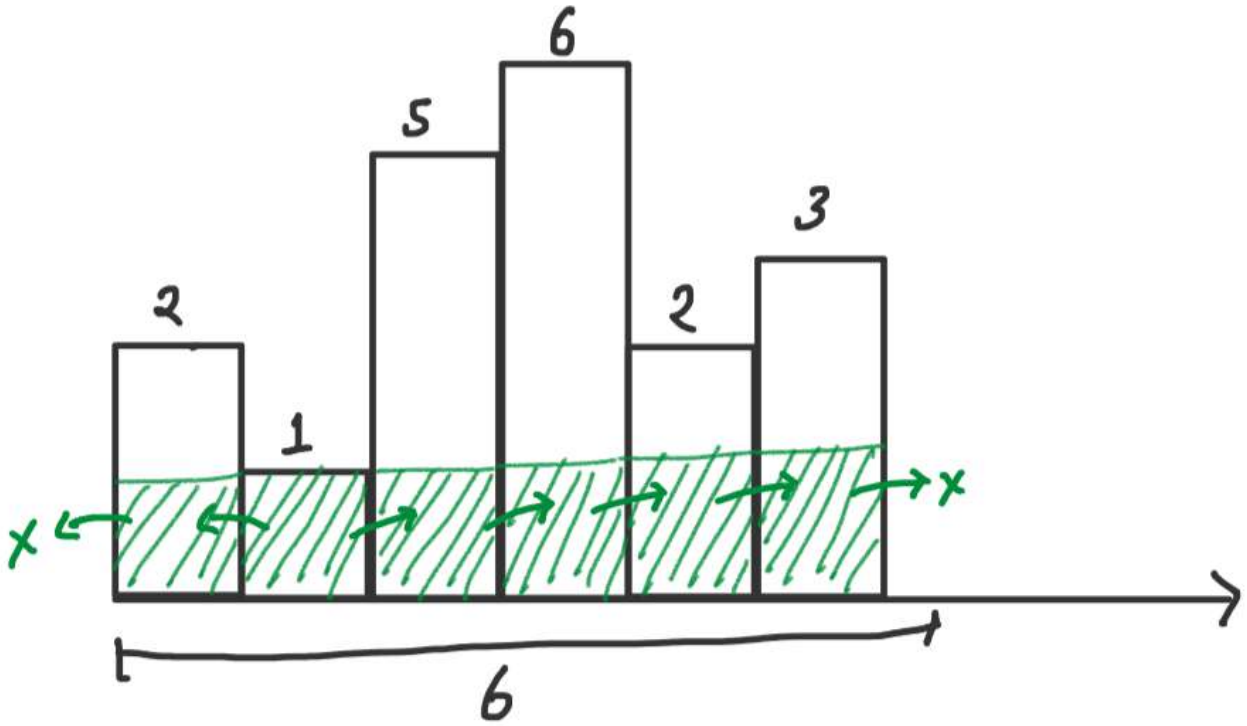
$= 2$



$A_2 = w_2 \times H_2$

$= 6 \times 1$

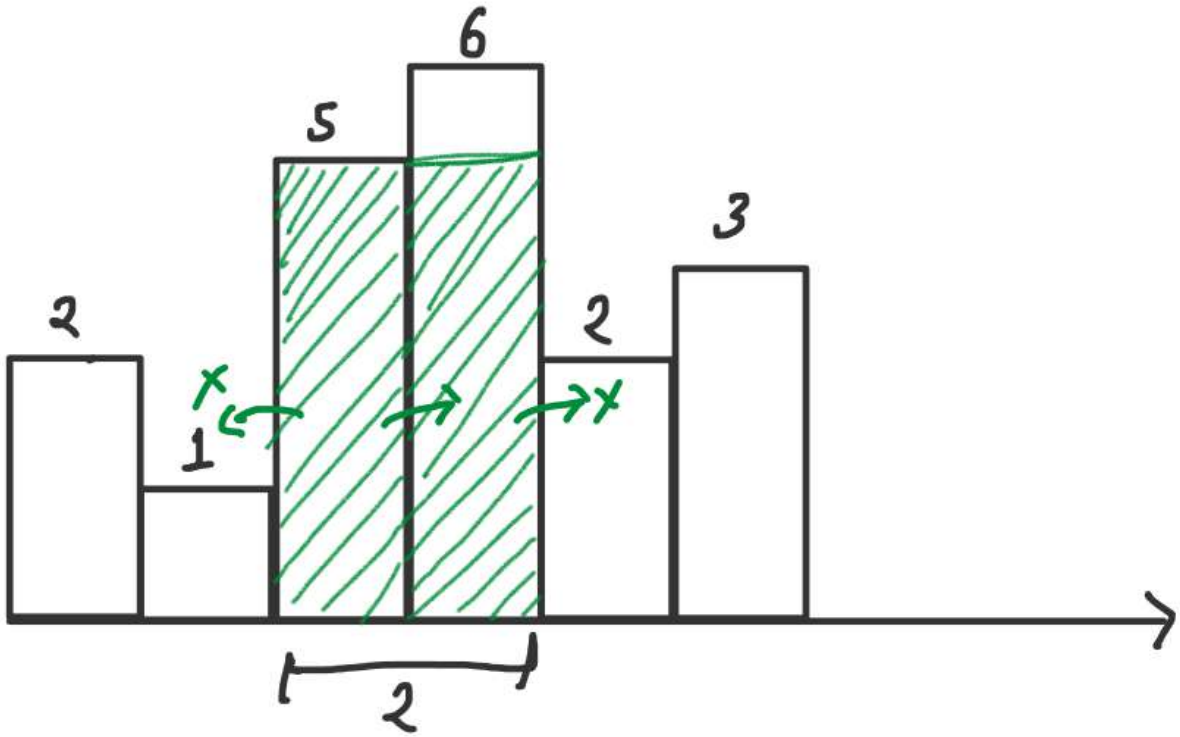
$= 6$



$A_3 = w_3 \times H_3$

$= 2 \times 5$

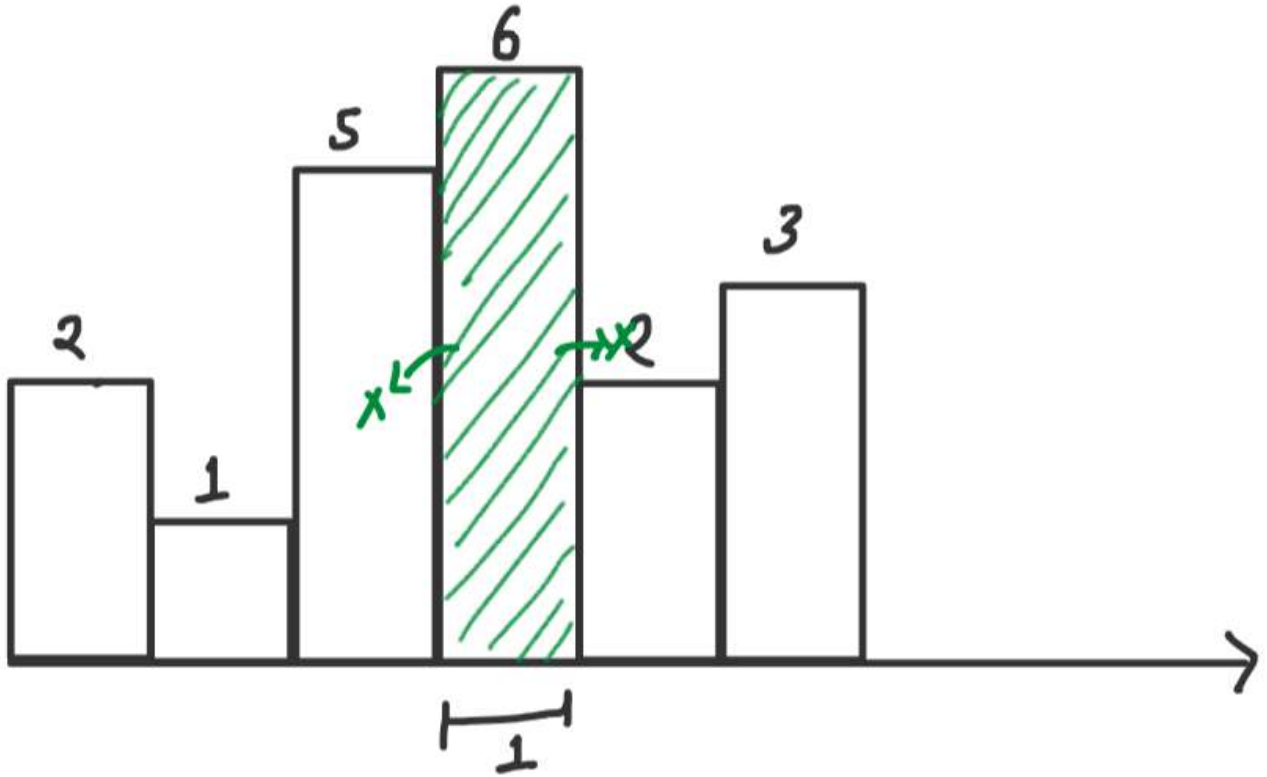
$= 10$



$A_4 = w_4 \times H_4$

$= 1 \times 6$

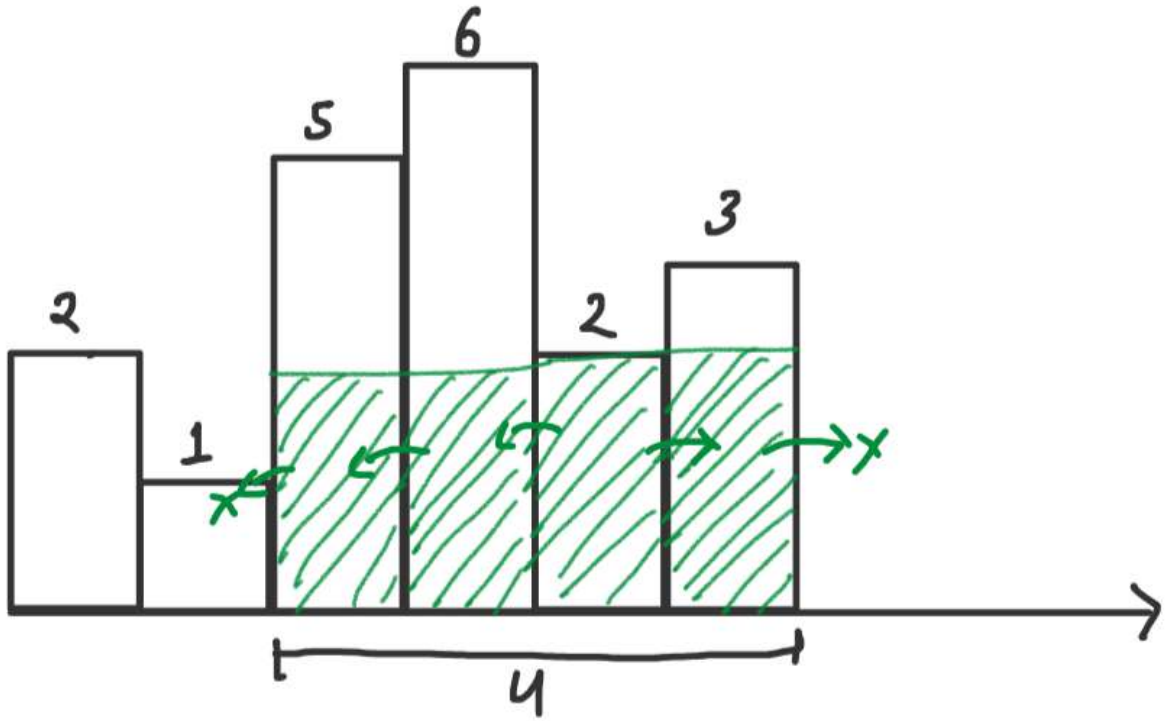
$= 6$



$A_S = w_S \times H_S$

$= 4 \times 2$

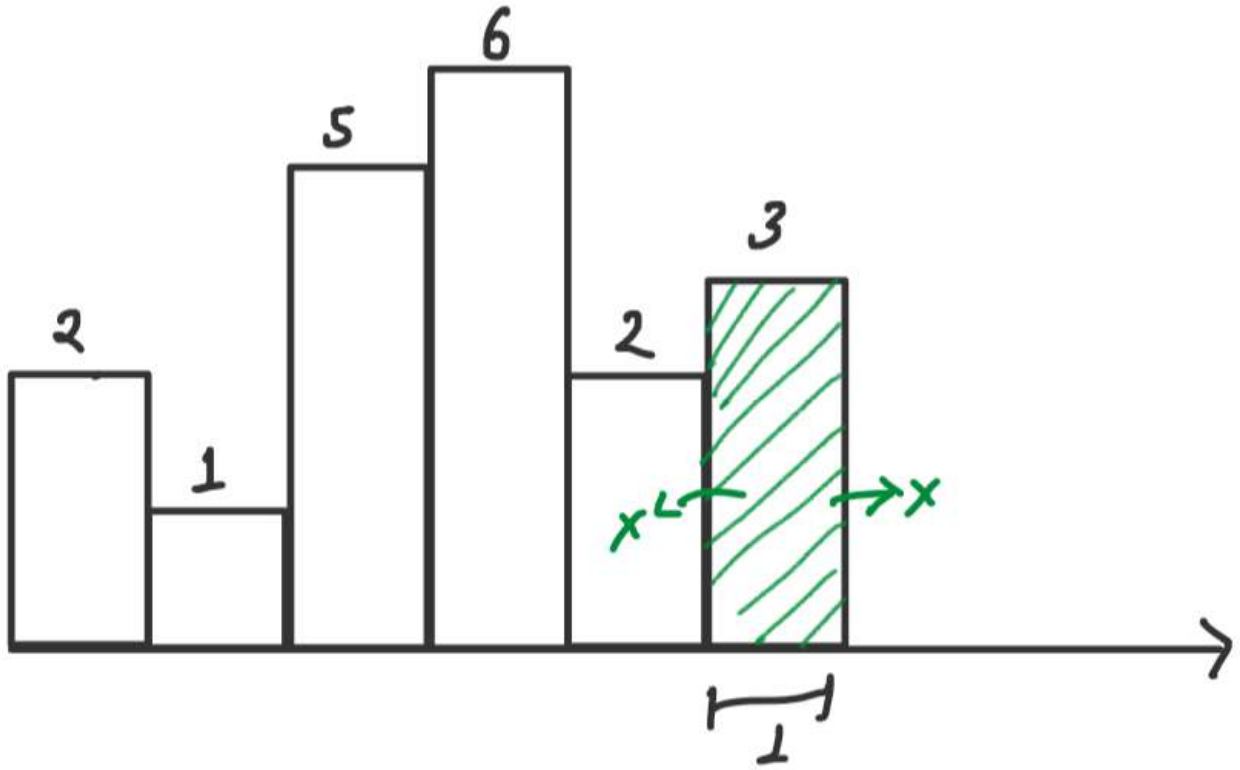
$= 8$



$A_6 = w_6 \times H_6$

$= 1 \times 3$

$= 3$



$A_1 = 2$

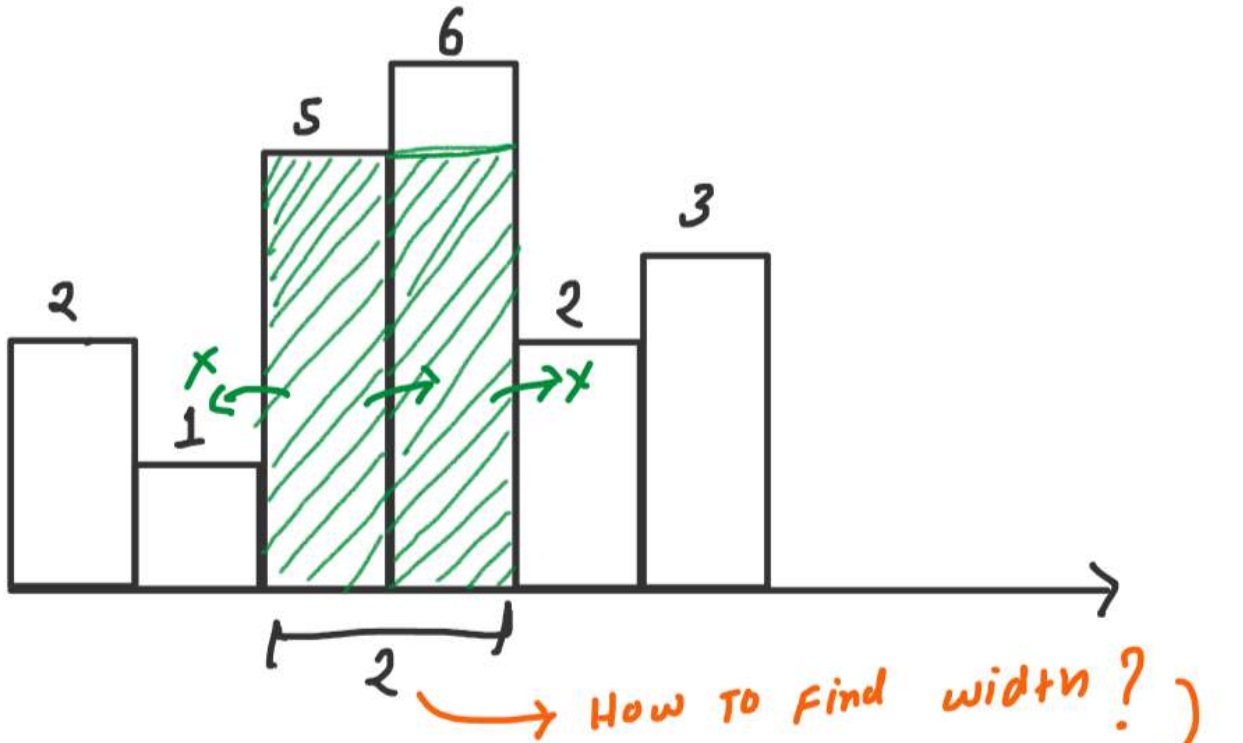
$A_2 = 6$

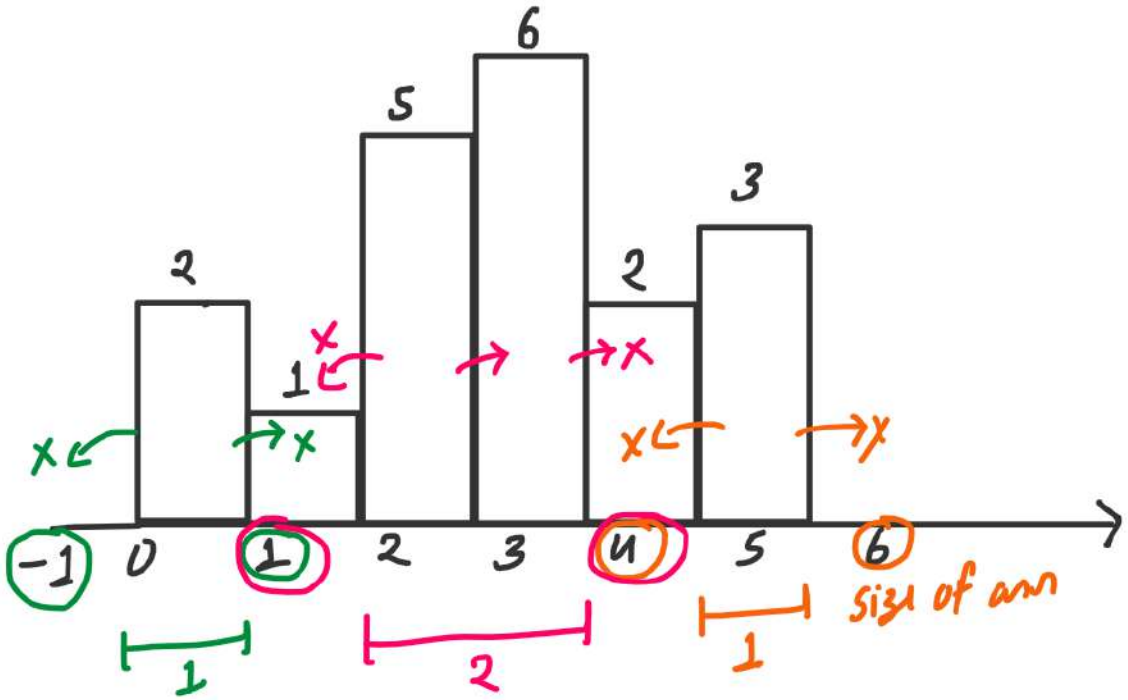
$A_3 = 10$ → MAX AREA

$A_4 = 6$

$A_5 = 8$

$A_6 = 3$





(A) left[] → Prev smaller Element Index

(B) right[] → Next smaller Element Index

(C) width = Next − Prev − 1

FOR 2

$w = 1 - (-1) - 1$

$= 1 + 1 - 1$

$= 1$

FOR 3

$w = 6 - 4 - 1$

$= 1$

FOR 5

$w = 4 - 1 - 1$

$= 2$

Complete code

```cpp
class Solution {
public:
    vector<int> nextSmallerElement(vector<int>&arr){...}
    vector<int> prevSmallerElement(vector<int>&arr){...}

    int largestRectangleArea(vector<int>& heights) {
        // Step A: Next smaller element index
        vector<int> next = nextSmallerElement(heights);

        // ❌Make sure karna ki next me koi bhi index -1 na ho
        // yadi hai to use next ke size se update kardo
        for(int i=0; i<next.size(); i++){
            if(next[i] == -1){
                next[i] = next.size();
            }
        }

        // Step B: Prev smaller element index
        vector<int> prev = prevSmallerElement(heights);

        // Now find the max area
        int maxArea = INT_MIN;

        for(int i=0; i<heights.size(); i++){
            // Step C: Find the width
            int height = heights[i];
            int width = next[i] - prev[i] - 1;
            int currArea = width * height;
            maxArea = max(maxArea, currArea);
        }
        return maxArea;
    }
};
```

T.C. = O(N)

S.C. = O(N)

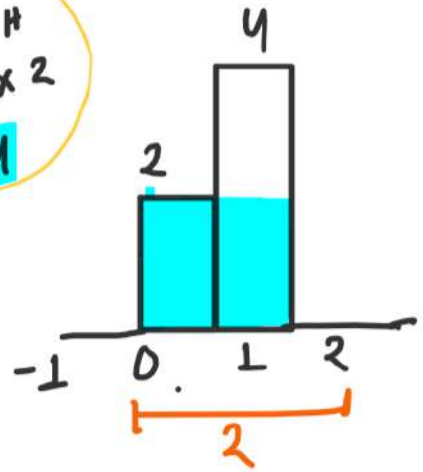
```cpp
vector<int> nextSmallerElement(vector<int>&arr){
    int size = arr.size();
    vector<int> ans(size);
    stack<int> st;
    st.push(-1);

    for(int i=size-1; i>=0; i--){
        int currElement = arr[i];

        // Find ans to currElement from stack
        while (st.top() != -1 && arr[st.top()] >= currElement)
        {
            st.pop();
        }

        // Ynha tak me tabhi pahuncha hoga jab st.top() < currElement
        ans[i] = st.top();
        st.push(i);
    }
    return ans;
}
```

```cpp
vector<int> prevSmallerElement(vector<int>&arr){
    int size = arr.size();
    vector<int> ans(size);
    stack<int> st;
    st.push(-1);

    for(int i=0; i<size; i++){
        int currElement = arr[i];

        // Find ans to currElement from stack
        while (st.top() != -1 && arr[st.top()] >= currElement)
        {
            st.pop();
        }

        // Ynha tak me tabhi pahuncha hoga jab st.top() < currElement
        ans[i] = st.top();
        st.push(i);
    }
    return ans;
}
```

width = 2 − (−1) − 1
= 2

AREA
⇒ w × h
⇒ 2 × 2
⇒ 4

if (nxt[i] == -1)
nxt[i] = nxt.size();



FOR 2

nxt ⇒ [2 | -1] [-1 | 2]
0

prev ⇒ [-1 | 1]
0 1

width = −1 − (−1) − 1
= −1 + 1 − 1
= −1

width −1 nahi ho sakti hai

RESOLVE THIS ISSUE