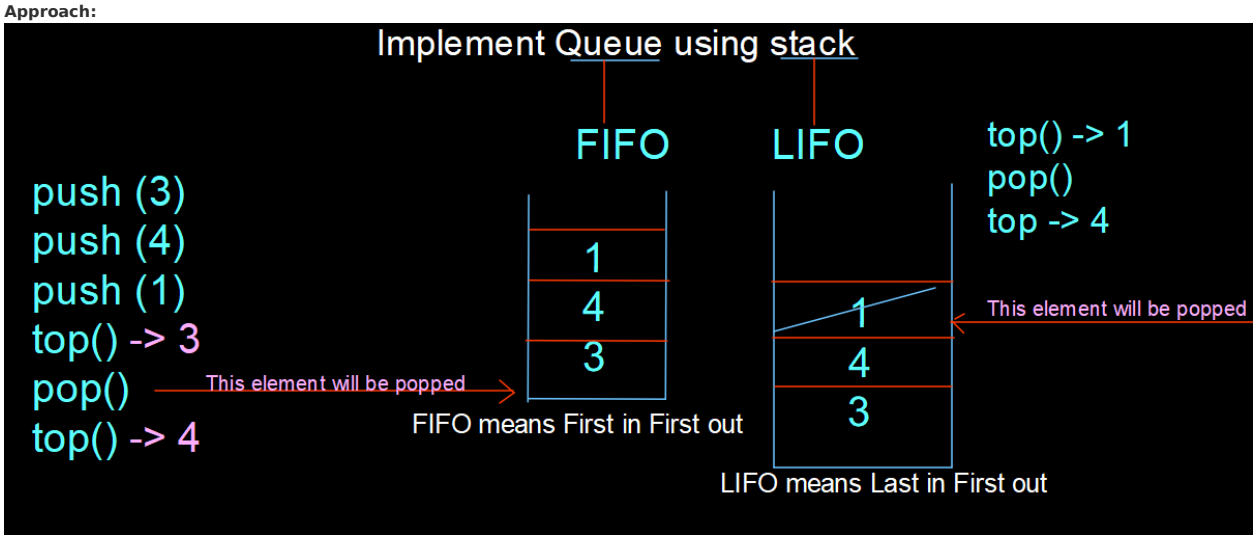
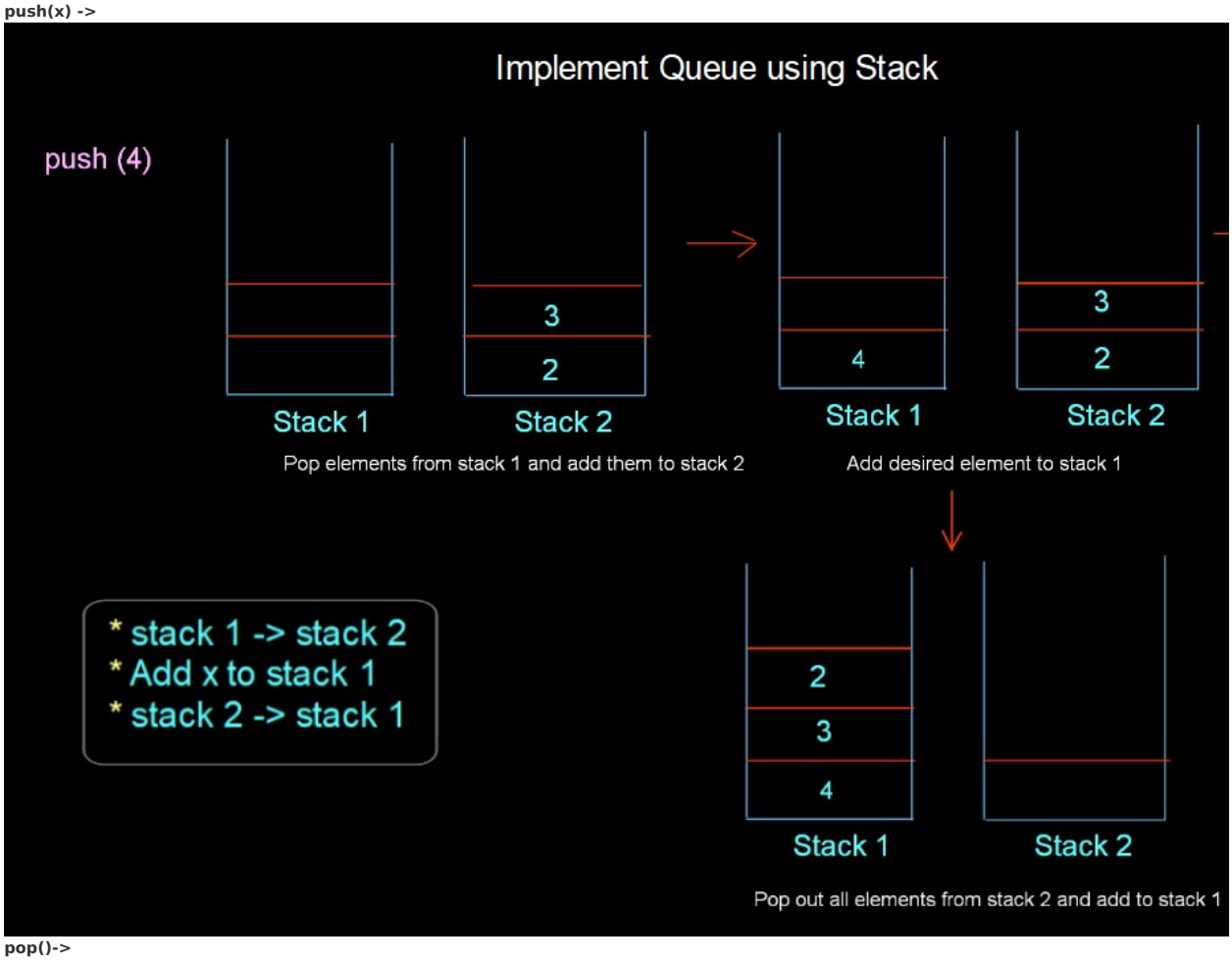
Implement Queue using Stack
Problem Statement: Given a Stack having some elements stored in it. Can you implement a Queue using the given Stack?
Queue: A Queue is a linear data structure that works on the basis of **FIFO(First in First out).** This means the element added at first will be removed first from the Queue.
Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Using two Stacks where push operation is O(N)
Approach:

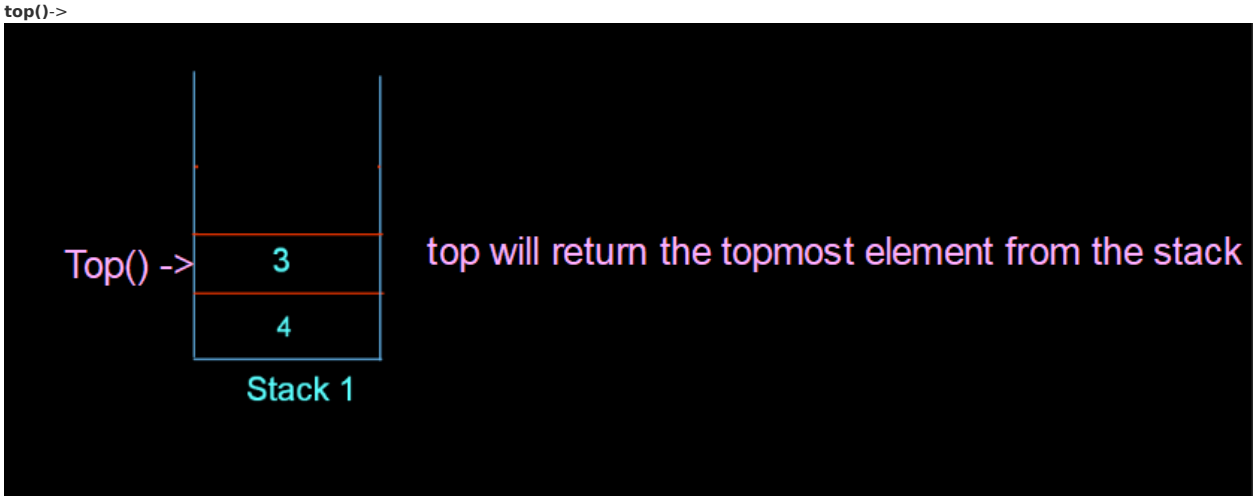


push(x) ->



pop()->

top()->



size()->

size() operation is for returning the size of a queue which can be done by using the function Stack1. size(). It will actually return the total number of elements in the queue.

Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Queue {
  stack < int > input, output;

  // Push elements in queue
  void Push(int data) {
    // Pop out all elements from the stack input
    while (!input.empty()) {
      output.push(input.top());
      input.pop();
    }
    // Insert the desired element in the stack input
    cout << "The element pushed is " << data << endl;
    input.push(data);
    // Pop out elements from the stack output and push them into the stack input
    while (!output.empty()) {
      input.push(output.top());
      output.pop();
    }
  }
  // Pop the element from the Queue
  int Pop() {
    if (input.empty()) {
      cout << "Stack is empty";
      exit(0);
    }
```

```
        int val = input.top();
        input.pop();
        return val;
    }
    // Return the Topmost element from the Queue
    int Top() {
        if (input.empty()) {
            cout << "Stack is empty";
            exit(0);
        }
        return input.top();
    }
    // Return the size of the Queue
    int size() {
        return input.size();
    }
};
int main() {
    Queue q;
    q.Push(3);
    q.Push(4);
    cout << "The element poped is " << q.Pop() << endl;
    q.Push(5);
    cout << "The top of the queue is " << q.Top() << endl;
    cout << "The size of the queue is " << q.size() << endl;
}
```

Output:

The element pushed is 3

The element pushed is 4

The element poped is 3

The element pushed is 5

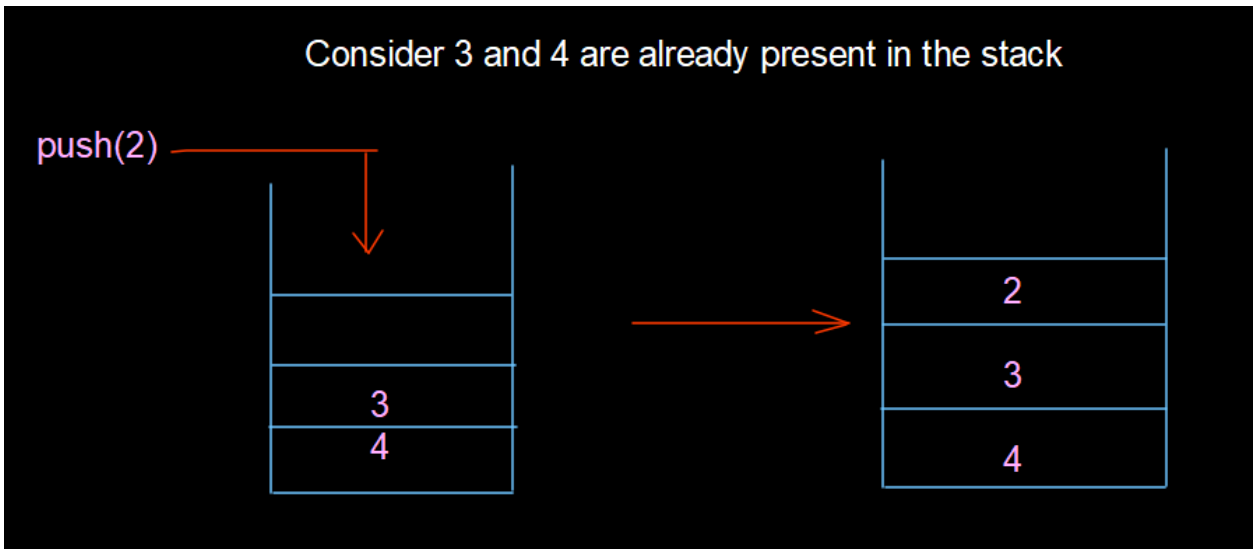
The top of the queue is 4

The size of the queue is 2

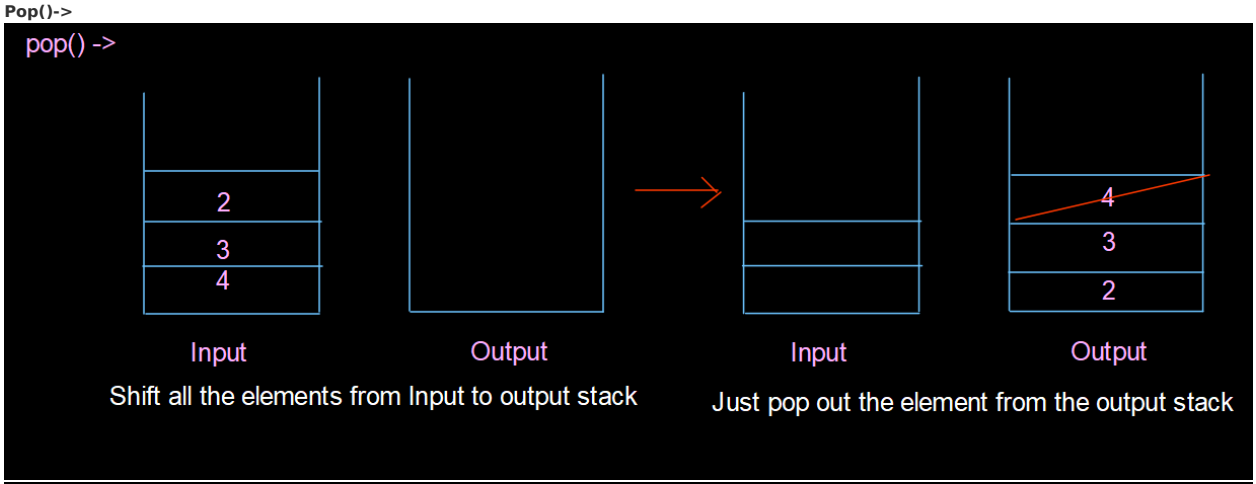
Time Complexity: O(N)

Space Complexity: O(2N)

Solution 2: Using two Stacks where push operation is O(1)
Approach :
Push()->



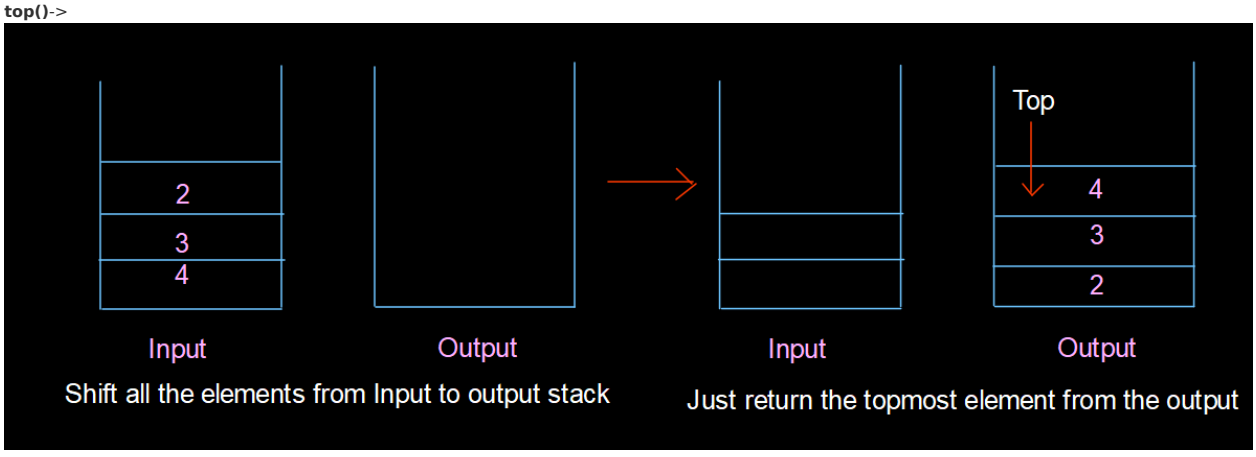
Pop()->

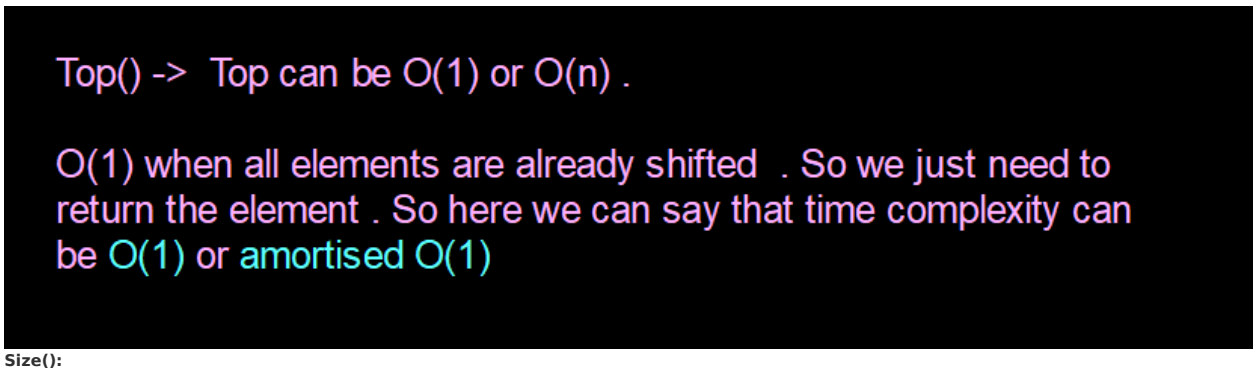


pop() -> pop can be O(1) or O(n) .

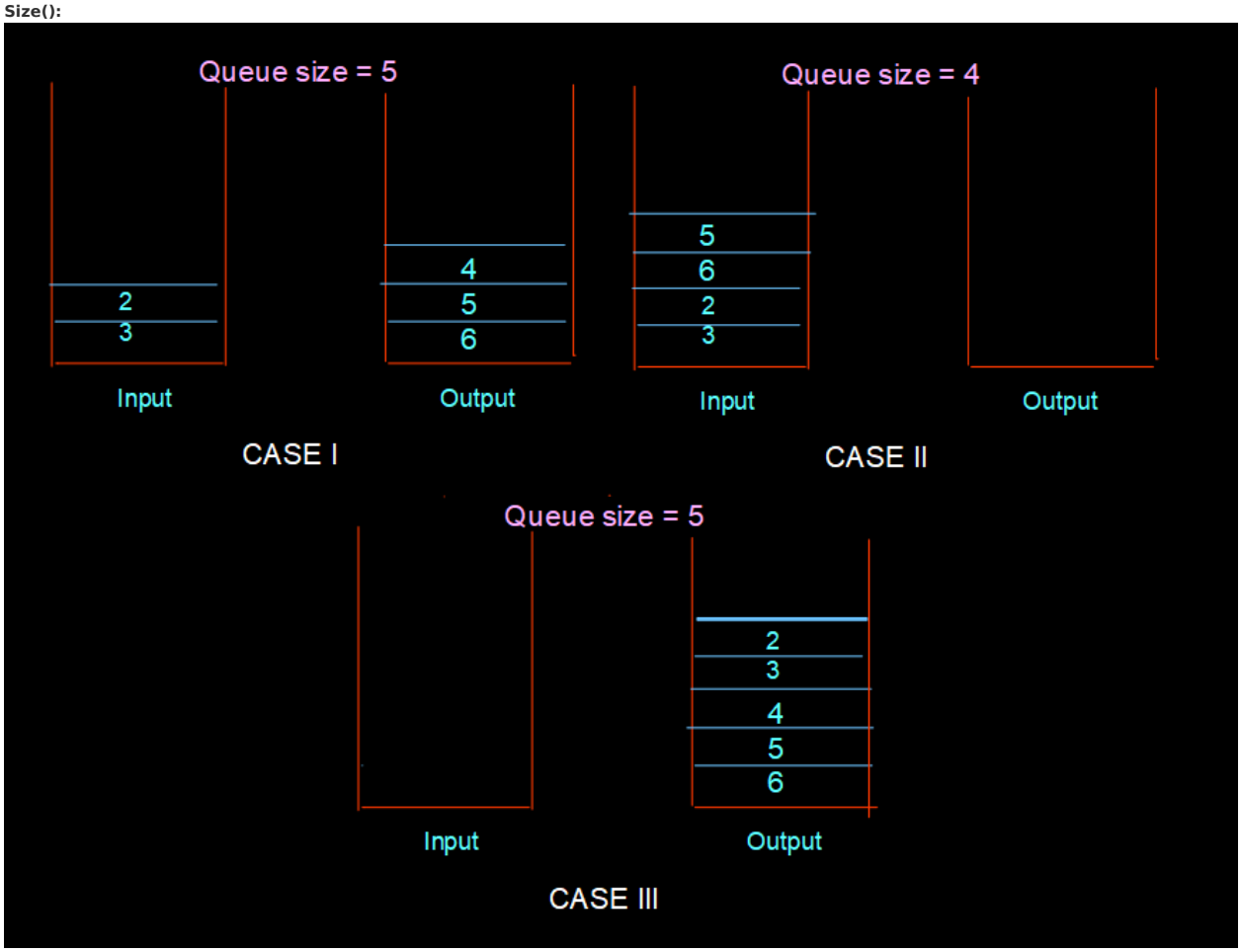
O(1) when all elements are already shifted . So we just need to delete the element . So here we can say that time complexity can be O(1) or amortised O(1)

top()->





Size():



Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include <bits/stdc++.h>

using namespace std;

class MyQueue {
  public:
    stack < int > input, output;
  /** Initialize your data structure here. */
  MyQueue() {

  }

  /** Push element x to the back of queue. */
  void push(int x) {
    cout << "The element pushed is " << x << endl;
```

```
        input.push(x);
    }

    /** Removes the element from in front of queue and returns that element. */
    int pop() {
        // shift input to output
        if (output.empty())
            while (input.size())
                output.push(input.top()), input.pop();

        int x = output.top();
        output.pop();
        return x;
    }

    /** Get the front element. */
    int top() {
        // shift input to output
        if (output.empty())
            while (input.size())
                output.push(input.top()), input.pop();
        return output.top();
    }

    int size() {
        return (output.size() + input.size());
    }

};
int main() {
    MyQueue q;
    q.push(3);
    q.push(4);
    cout << "The element poped is " << q.pop() << endl;
    q.push(5);
    cout << "The top of the queue is " << q.top() << endl;
    cout << "The size of the queue is " << q.size() << endl;

}
```

Output:

The element pushed is 3

The element pushed is 4

The element poped is 3

The element pushed is 5

The top of the queue is 4

The size of the queue is 2

Time Complexity: O(1)

Space Complexity: O(2N)