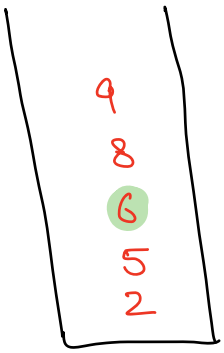
Stack :-

↳ LIFO



- Push(x) ⇒ O(1)
- pop() ⇒ O(1)
- top() | peek()
- size()
- isEmpty()

ADT (Abstract Data Type)

Stack implementation → Arrays

→ L.L :

Push & POP at Head.

* Applications

1) Fun^ call stack
2) Undo/Redo
3) Browser back button
4) Calculations / Expression evaluation.

Q.1 Amazon

Given a string, Remove every consecutive duplicate pairs until there are No consecutive duplicate pairs.

S: a c b b c k

a c c k

a k

S: a a a b

a b

S: a b c k k c b a m

a b c c b a m

a b b a m

a a m

m

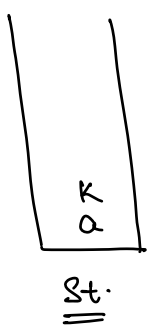
S: a b a b a b

a b a b a b

S: a b c c c d b

a b c d b

S: a c b b c k ⇒ a k

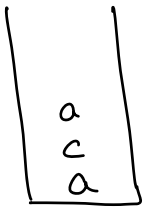


b
c

TC: O(N)

SC: O(N)

S: a c c c a ⇒ a c a



Follow-up question (Stack)

1) Remove all duplicates.

baaabbc ⇒ c , baabac ⇒ c

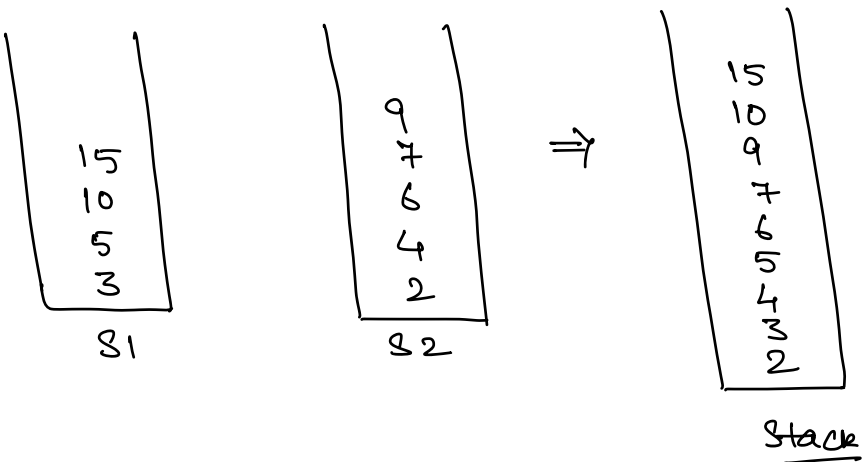
2) Remove K duplicate

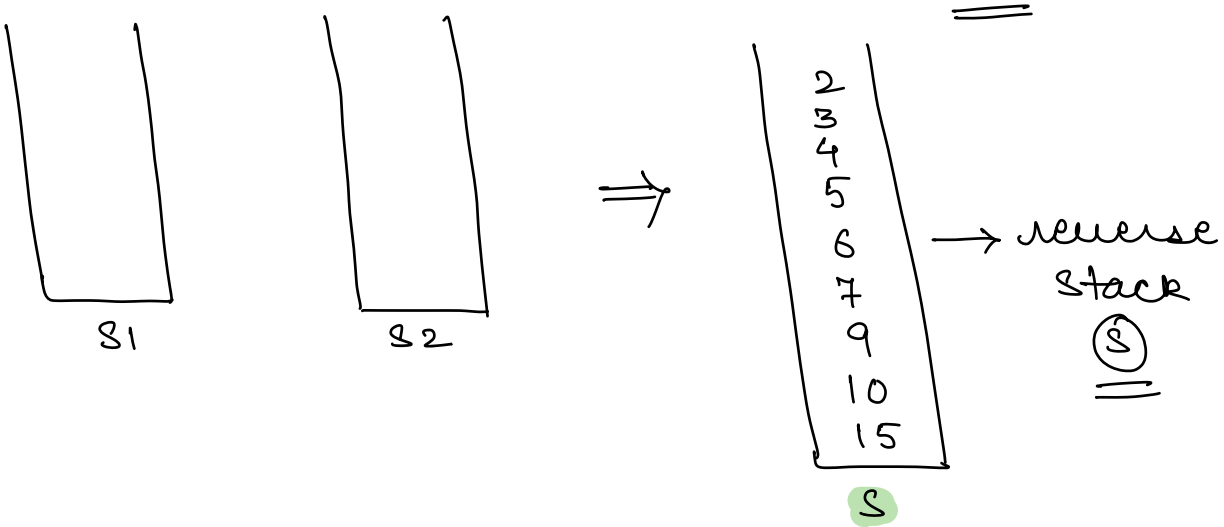
baaaa, K = 3.

↳ ba.

Q.2 Given 2 sorted stacks, Merge them in sorted order.

Amazon





* pick the larger value from top of stacks.

```
Stack<int> merge(Stack<int> s1, s2) {

    Stack<int> s;
    while (s1.size()>0 && s2.size()>0) {

            if (s1.top() > s2.top()) {
                    s.push(s1.pop());
            }
            else {
                    s.push(s2.pop());
            }
    }
    while (s1.size()>0) {
        s.push(s1.pop());
    }
    while (s2.size()>0) {
        s.push(s2.pop();
    }
    return reverse(s);
}

Stack<int> reverse(Stack<int> s) {
        stack<int> revStack;
        while (s.size()>0) {
            revStack.push(s.pop());
        }
        return revStack;
}
```

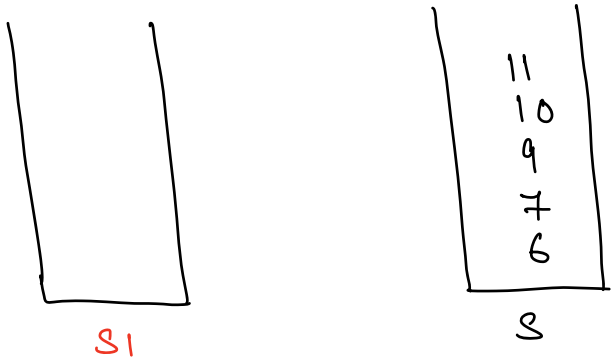
TC: O(N+M) SC: O(N+M)

* Reverse the stack using Recursion.

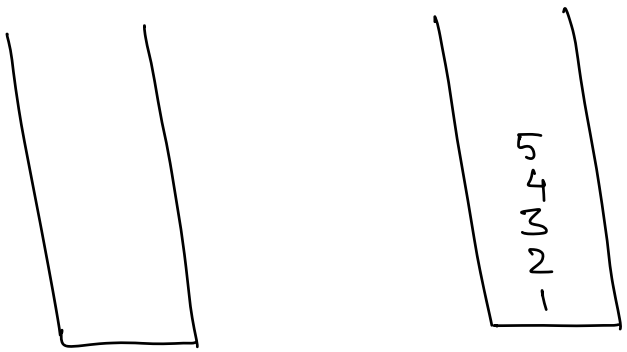
Q. Given a stack, sort in descending order. (Only using stack DS)

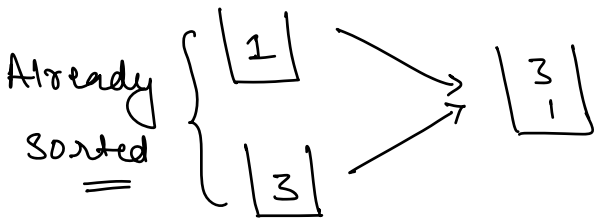
Google.

Approach #1 :-



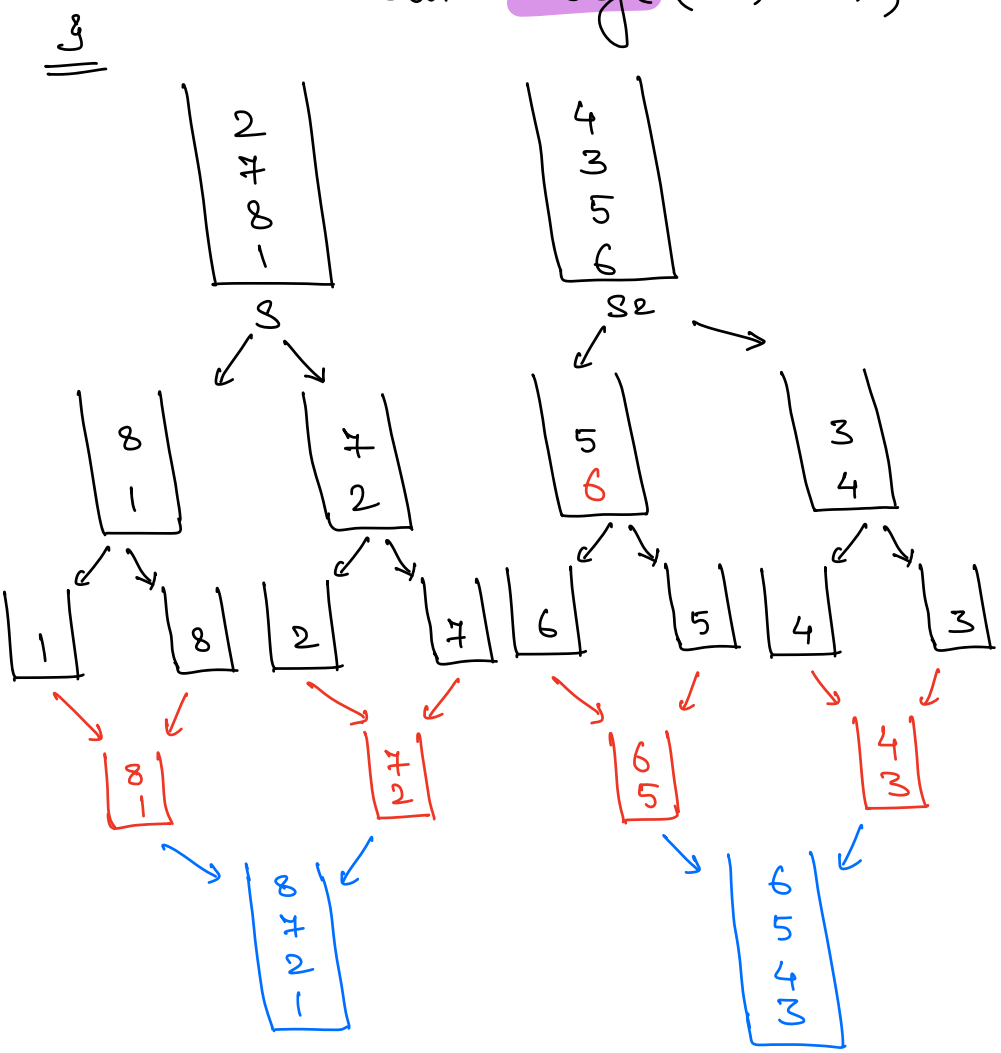
TC: $O(N^2)$

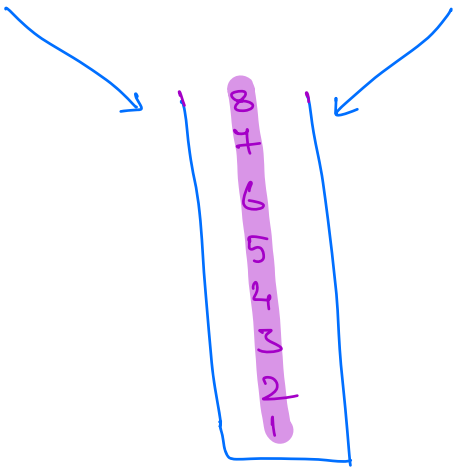





```
mergeSort (Data) {
    D1 ← firstHalf(Data)
    D2 ← secondHalf(Data)
    mergeSort (D1)
    mergeSort (D2)
    return merge(D1, D2);
}
```

```
Stack<int> mergeSort(Stack<int> s) {
    if (s.size() <= 1)
        return s;
    Stack<int> s2;
    int size = s.size();
    for(i = 0; i < size/2 ; i++) {
        s2.push(s.pop());
    }
    s = mergeSort(s);
    s2 = mergeSort(s2);
    return merge(s, s2);
```





TC : O(N log N)

SC : O(N) [O(N+logN)]

* Expression Evaluation :-

$7 \times 1 + 2 - 8 \times 3 + 10/5 =$ ~~?~~

$7 + 2 - 24 + 2$

$9 - 24 + 2 = -15 + 2 = -13$

/, *

+, -

Infix Notation	Postfix Notation
A x B	A B x
A / B	A B /
A + B	A B +
A - B	A B -

* Calculation of Infix notation is NOT optimal.

* $A + B \times C \Rightarrow A + BC\times$

$\Downarrow$

$A\ BC\times +$

Expression Evaluation :-

1) Convert from infix to postfix.
2) Evaluate postfix expression.

Quiz $4 + 8 \times 7 \Rightarrow 4 + 87\times$

$\Downarrow$

$4\ 8\ 7\ \times\ +$

Quiz $10 + 3 \times 4 - 7$

$\Rightarrow 10 + 34\times - 7$

$\Rightarrow 10\ 34\times + - 7$

$\Rightarrow 10\ 3\ 4\ \times + 7\ -$

Quiz $10 / (4-2) * 6 + 9$

$10/42- \times 6 + 9$

$1042-/ \times 6 + 9$

$1042-/6\times + 9 \Rightarrow 1042-/6\times 9+$

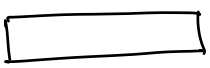
Quiz $(10+3) \times 2 - (7-6) \times (4+8)$

$10\ 3+ \times 2 - 7\ 6- \times 4\ 8+$

$10\ 3+2\times \ominus 7\ 6-4\ 8+\times$

$10\ 3+2\times 7\ 6-4\ 8+\times -$

Ex :- $10 + 3^{\downarrow} \Rightarrow 10\ 3+$

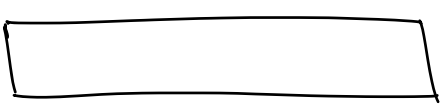


* Operands follow the same relative ordering in postfix & infix notations.

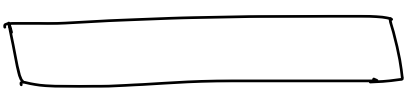
Ex :- $10 + 3 \times 4^{\downarrow} \Rightarrow 10\ 3\ 4 \times +$

Ex $10 \times 3 + 4^{\downarrow} \Rightarrow 10\ 3\ 4 + \times$

X

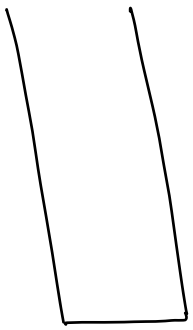


Ex $10 \times 3 + 4^{\downarrow} \Rightarrow 10\ 3 \times 4 +$



Ex

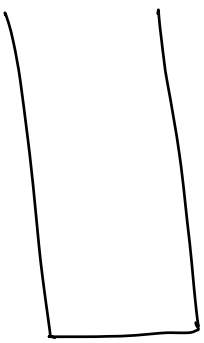
$10 \times (3+4)$



10 3 4 + x

Ex

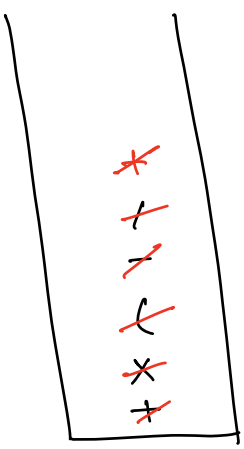
$(10+8) \times 5$



10 8 + 5 x

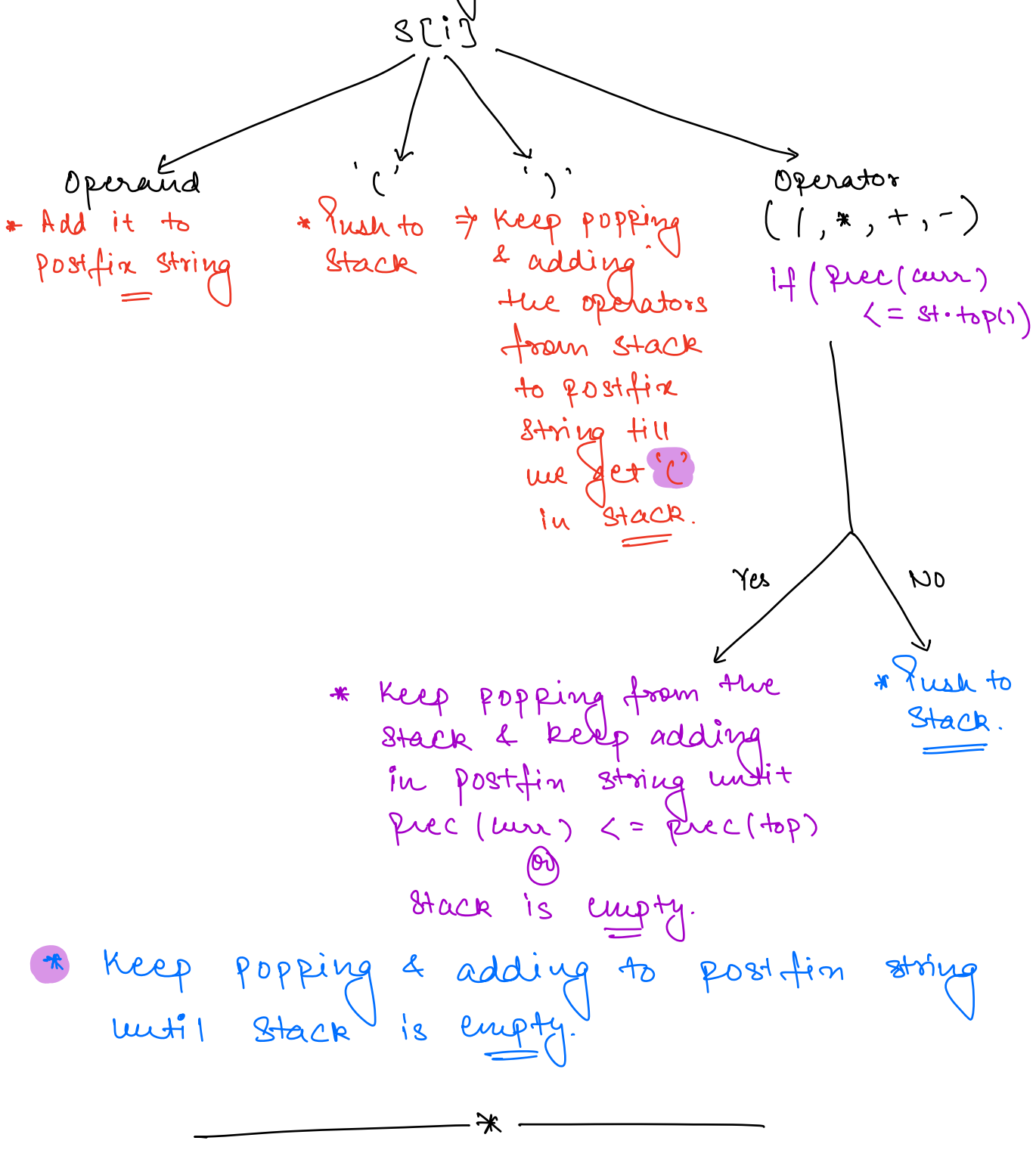
Ex

$3 + 10 \times (3 - 4/2) + 3$



3 10 3 4 2 / − x + 3 +

Traverse the infix notation :-



* Keep popping & adding to postfix string until Stack is empty.
