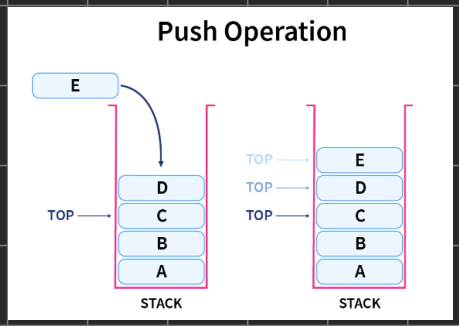
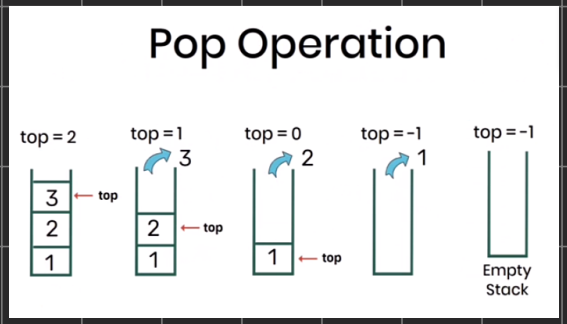
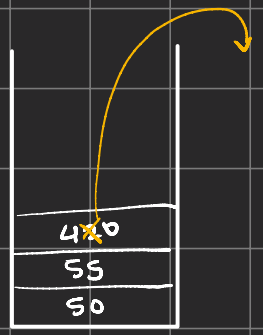
Stack - data structure → item / data storage in LIFO (last in first out) order







s.push(50)
s.push(55)
s.push(420)
s.pop()

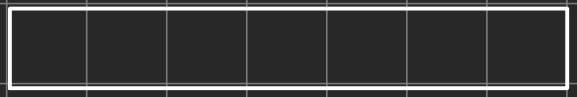
s.top() ~→ return topmost Ele

s.empty() ~→ T/F

s.size() ~→ no of Ele

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main(){

    // creation
    stack<int> s;

    // push
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);

    // pop
    s.pop();

    // ele on top
    cout << s.top() << endl;

    // size
    cout << s.size() << endl;

    // is empty
    cout << s.empty() << endl;

    // print
    while (!s.empty()){
        cout << s.top() << " ";
        s.pop();
    }
```

size = 7 → show total size



top → store index of last ele

= size - top
= 7 - 6
= 1

Push

space available → insert
top++
arr[top] = data

space not available → overflow
size - top > 1

space available

pop

stack empty → top == -1

stack has elements → top --

getTop

stack empty

stack has ele

getSize

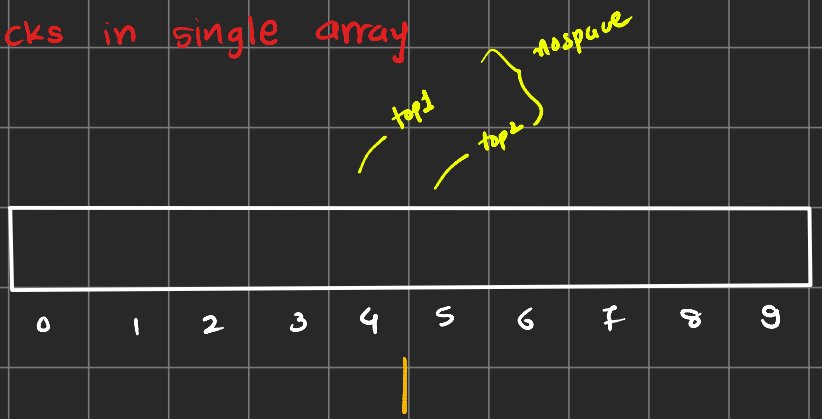
stack empty

return no of valid ele present in top

```cpp
class Stack {
public:
    int* arr;
    int top;
    int size;

    Stack(int size) {
        this->arr = new int[size];
        this->size = size;
        top = -1;
    }

    void push(int data) {
        if (size - top > 1) {
            top++;
            arr[top] = data;
        } else {
            cout << "stack overflow" << endl;
        }
    }

    void pop() {
        if (top == -1) {
            cout << "stack underflow/empty" << endl;
        } else {
            top--;
        }
    }

    int getTop() {
        if (top == -1) {
            cout << "stack underflow/empty" << endl;
            return -1;
        } else {
            return arr[top];
        }
    }

    int getSize() {
        if (top == -1) {
            cout << "stack underflow/empty" << endl;
            return 0;
        } else {
            return top + 1;
        }
    }

    bool empty() {
        return top == -1;
    }
```

can make them private

☆ 2 stacks in single array

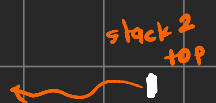


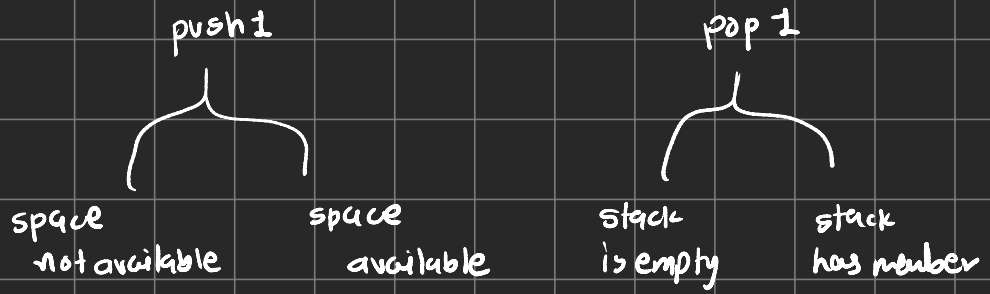
Approch① Half1 means stack1
Half2 means stack2
} but memory wastage hoga idar

Haan agar 3 stack create karne hai then we have to do n/3

Approach② stack 1 top

stack 2 top





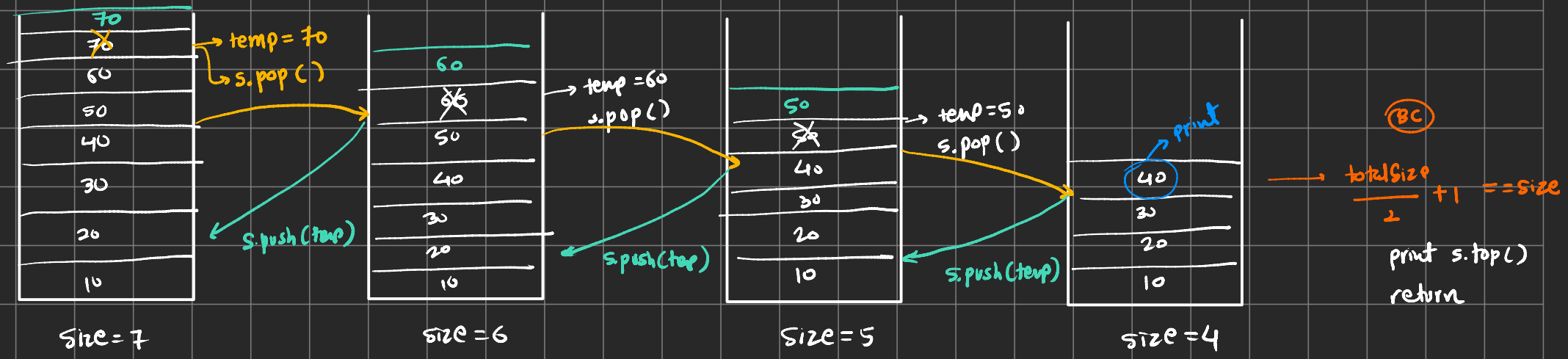
top2 - top1 == 1

```
class Stack {
private:
    int* arr;
    int size;
    int top1;
    int top2;

public:
    Stack(int size) {
        arr = new int[size];
        this->size = size;
        top1 = -1;
        top2 = size;
    }

    void push1(int data) {
        if (top2 - top1 == 1) {
            cout << "overflow" << endl;
        } else {
            top1++;
            arr[top1] = data;
        }
    }

    void push2(int data) {
        if (top2 - top1 == 1) {
            cout << "overflow" << endl;
        } else {
            top2--;
            arr[top2] = data;
        }
    }

    void pop1() {
        if (top1 == -1) {
            cout << "underflow" << endl;
        } else {
            top1--;
        }
    }

    void pop2() {
        if (top2 == size) {
            cout << "underflow" << endl;
        } else {
            top2++;
        }
    }
};
```

string Reverse

```
string str = "parth";

stack<char> st;

for(int i=0; i<str.length(); i++){
    st.push(str[i]);
}

while(!st.empty()){
    cout << st.top() << " ";
    st.pop();
}
```

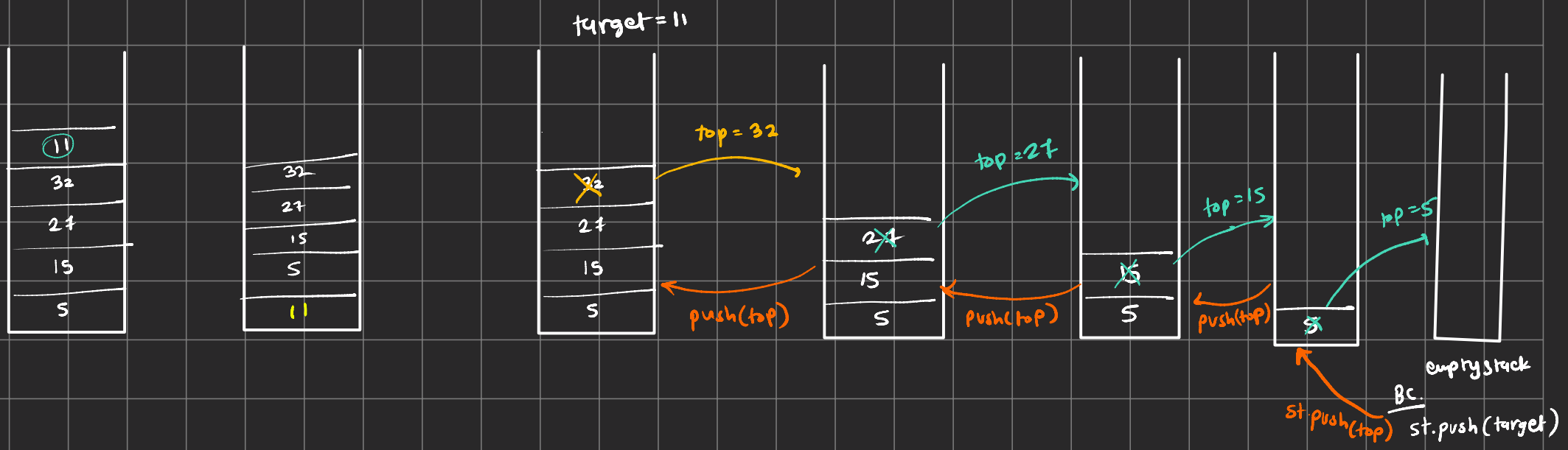
find middle ele from stack



```cpp
void findMiddleEle(stack<int> &s, int totalSize){
    if(totalSize/2 + 1 == s.size()){
        cout << "middle ele is => "<< s.top() <<endl;
    }
    int temp = s.top();
    s.pop();

    findMiddleEle(s,totalSize);

    s.push(temp);
}
```

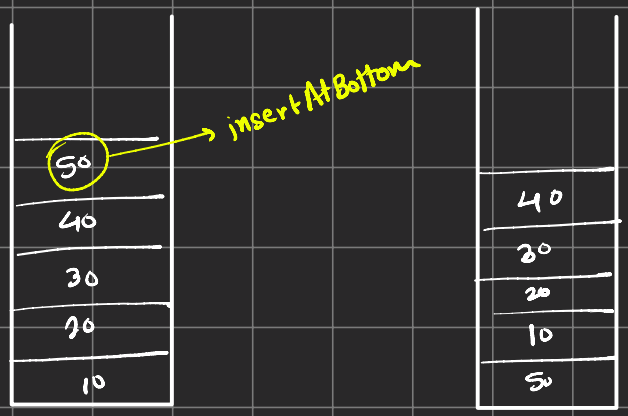
```cpp
stack<int> st;
int totalSize = st.size();
```

insert at bottom

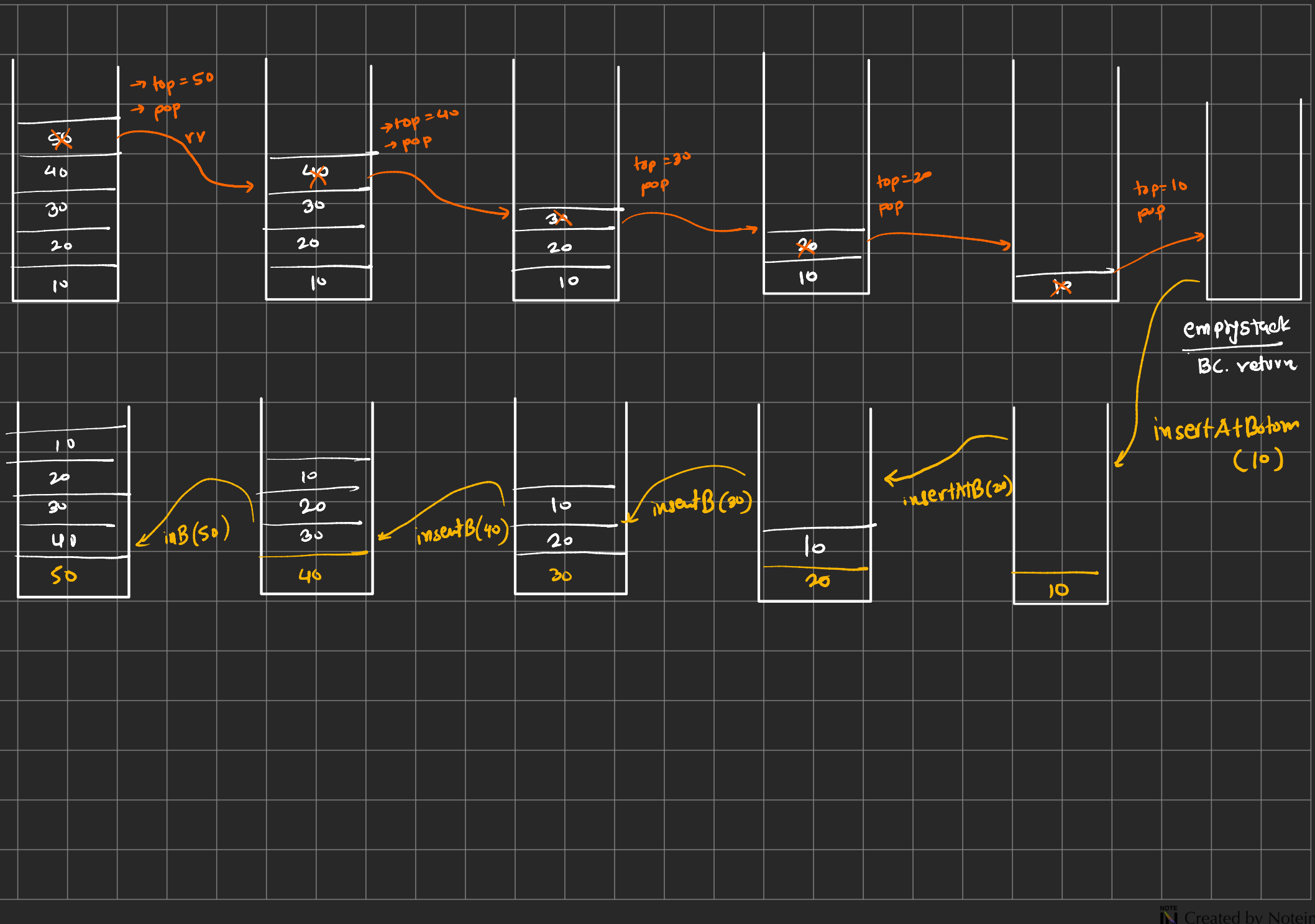


```cpp
void solve(stack<int> &st, int target){
    //bc
    if(st.empty()){
        st.push(target);
        return;
    }

    // rr
    int topEle = st.top();
    st.pop();
    solve(st, target);

    // bkt
    st.push(topEle);
}

void insertAtBottom(stack<int> &st){
    if(st.empty()){
        cout << "stack is empty cannot insert at bottom" << endl;
        return;
    }

    int target = st.top();
    st.pop();
    solve(st, target);
}
```

☆ Reverse stack




```cpp
void reverseStack(stack<int> & st){
    // bc
    if(st.empty()){
        return;
    }

    int target = st.top();
    st.pop();

    // stack reverese
    reverseStack(st);

    // insert target at bottom
    insertAtBottom(st,target);

}
```



⁂ Valid Parenthesis

([ ]) → T

([{ }]) → T

([} {]) → F

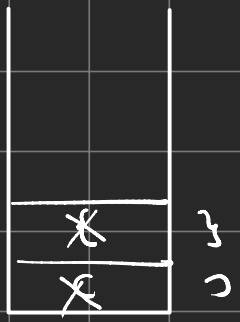
} { → F

valid → open Bracket, same close Brackets

({ })

( ) [ ] { }

- openBracket mile to stack me daal do
- jab bhi closing Bracket mile to stack me check karo
  opening Bracket pada hai ya nahi → if present hai to pop
  ↳ else
- last mei stack empty hai to expression valid tha



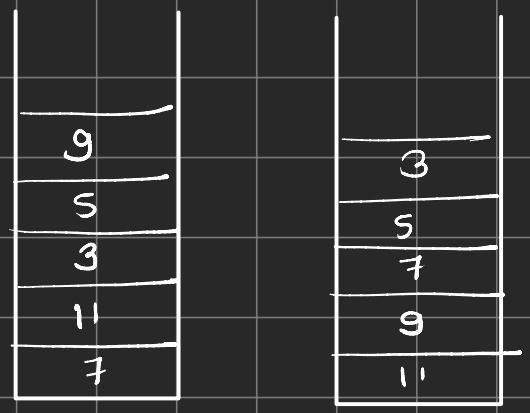
```cpp
bool isValid(string s) {

    stack<char> st;

    for (int i = 0; i < s.length(); i++) {
        char ch = s[i];

        // opening bracket
        if (ch == '(' || ch == '{' || ch == '[') {
            st.push(ch);
        } else {
            // closing bracket
            if (!st.empty()) {
                char topch = st.top();
                if (ch == ')' && topch == '(') {
                    st.pop();
                } else if (ch == '}' && topch == '{') {
                    st.pop();
                } else if (ch == ']' && topch == '[') {
                    st.pop();
                } else {
                    // bracket not matching
                    return false;
                }
            }
            else{
                return false;
            }
        }
    }

    return st.empty() ? true : false;
}
```

case = '}'

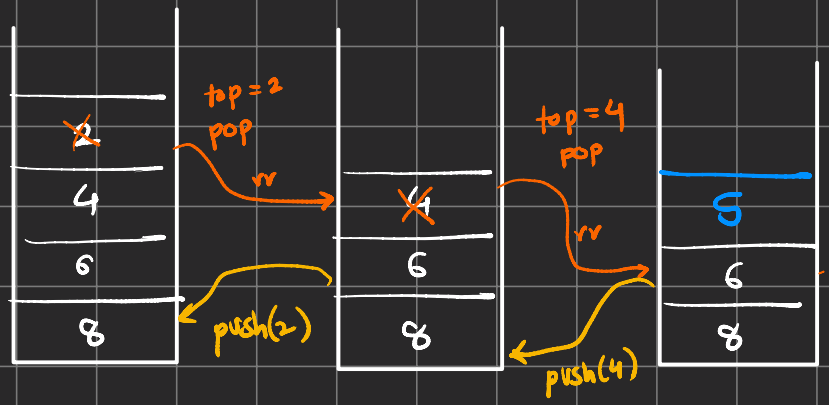
st empty hai but fir bhi hum

topCh = st.top() find kar rahe hai wo error degu

* sort stack



target = 5 insert into sorted stack



5 > 2

5 > 4

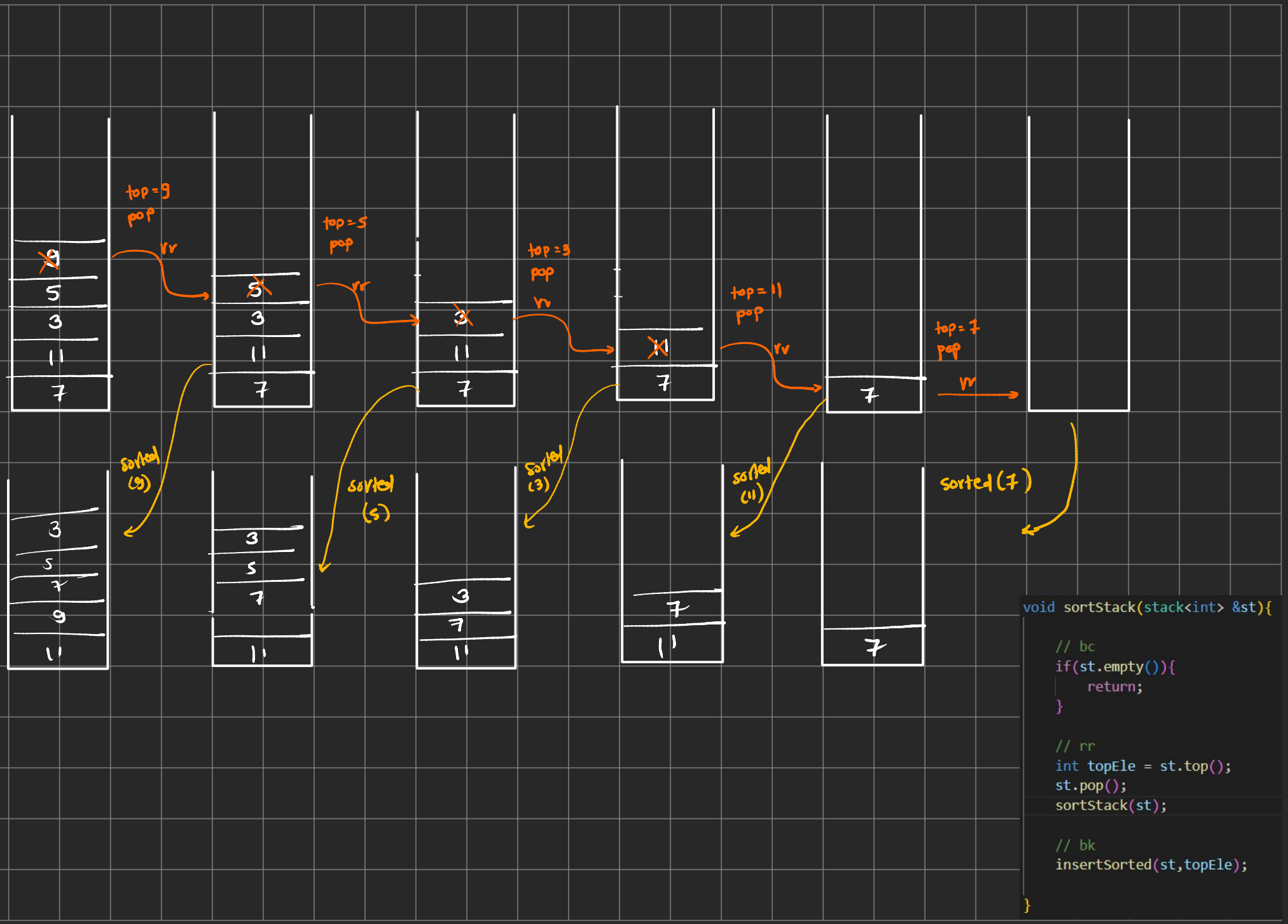
5 > 6 → False

target > top → False

} BC → insert target

push(target)

```
void insertSorted(stack<int> &st, int target){
    // bc
    if( st.empty() || (st.top() >= target) ){
        // stack empty hoga
        // agar empty hai to st.top() find nhi hoga and wo protection rahega
        st.push(target);
        return;
    }

    // rr
    int top = st.top();
    st.pop();
    insertSorted(st, target);

    //bk
    st.push(top);
}
```




```
void sortStack(stack<int> &st){

    // bc
    if(st.empty()){
        return;
    }

    // rr
    int topEle = st.top();
    st.pop();
    sortStack(st);

    // bk
    insertSorted(st,topEle);

}
```

* Remove redundant brackets

useless
needless

- (()) (a+b)
- (a+b)

→ (a+b) — No

→ ((a+b*c)+(d)) → yes

→ (a+b+(c/d)) → No

→ ((a/b)) → yes

(operator)

Brackets ke bich me operator present hai to useful hai

agar operator nahi hai to useless hai

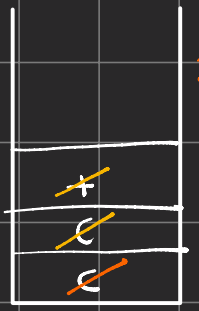
(a + b)

opening bracket, operator → push

a, b, c, d → ignore

closing bracket → go until first opening bracket (while checking operator inbetween or not)

((a+b))



→ for this I poped till opening bracket but I didnot get operator

needless

yes

✱ min stack - design a stack that supports push, pop, retriving the minimum element in constant time (O(1))

this is main because we know push, pop, top are always in O(1)

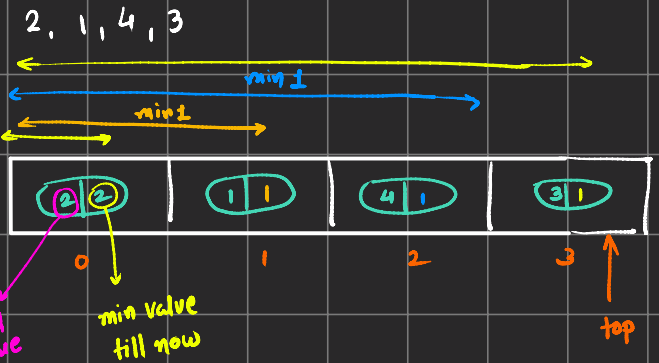
why? - because to find min. ele we need to travel whole array

pair<int, int> p;

how to access

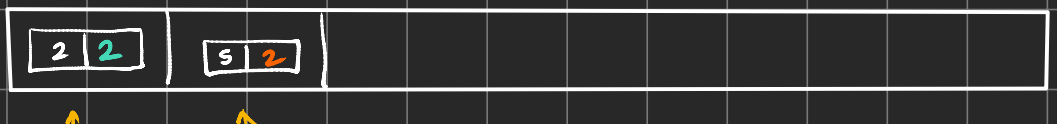
p.first

p.second



top Element is → vector.back().first

min value in stack → vector.back().second




```
pair<int,int> p
p.first = val
p.second = val
st.push_back(p)
```

→ min value wo element khud hi hoga

```
pair<int,int> p
p.first = val
p.second = min(val, st.back().second)
st.push_back(p)
```

↳ min value (value, lastEle ka min val)

st.back() is similar to st[st.size()-1]

both shows last Element

= min(2,5)
= 2

```cpp
class MinStack {
public:

    vector< pair<int,int> > st;

    MinStack() {


    }

    void push(int val) {
        if(st.empty()){
            pair<int,int> p = make_pair(val,val);
            st.push_back(p);
        }else{
            pair<int,int> p;
            p.first = val;
            p.second = min(val, st.back().second);  // min of currVal and abtak ka minimum means stackTop par ka min
            st.push_back(p);
        }
    }

    void pop() {
        st.pop_back();
    }

    int top() {
        // stack k top par
        return st.back().first;
    }

    int getMin() {
        // stack k top par
        return st.back().second;
    }
};
```

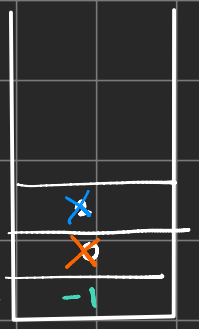
make own pair

```
class pair {

  public:
     int a;
     int b;
}
```

Longest valid Parenthesis

`)()())()(` → 4

`))))()` → 2



`(()())`
`-1 0 1 2 3 4 5`

initial (-1) add karte hai to find length

openBracket → pushIndex
closeBracket → pop
→ mean is valid expression mila tha
→ length = currentIndex − s.top()
= 1 − (−1)
= 2

} for index 1

→ len = 3 − (−1)
= 4

} for index 3

`))`
`-1 0 1`

0th index closeBracket → pop
→ len = curr − s.top()

ERROR
we do not have top
Stack is empty

if (ch == '(') → s.push(i)
else
↳ if (!s.empty) → len = i − s.top()
non empty

Else
↳ ignore karo ye ')' bracket ko but (-1) add karna hai to maintain our logic for len

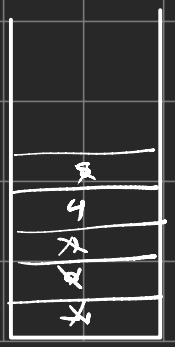
s.push(i)

`(()))(()))(`
`0 1 2 3 4 5 6 7 8`

len = 2
len = 4
len = 2

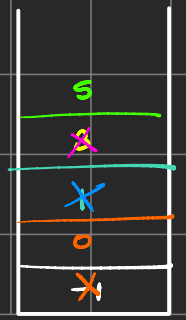
longest Parenthesis length is max(2,4,2) = 4

max of all lengths



= 2 − 0 = 2
= 3 − (−1) = 4
= 6 − 4 = 2

`)()())(`
`0 1 2 3 4 5`



0th index → pop
→ push i

1th index → push

2 index → pop
→ len = i − top = 2 − 0 = 2

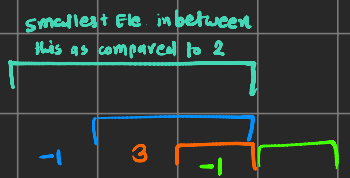
3 index → push

4 index → pop
→ len = i − top = 4 − 0 = 4

5 index → push

```cpp
int longestValidParentheses(string s) {

    stack<int> st;
    st.push(-1);
    int maxLen = 0;

    for(int i=0; i<s.length(); i++){

        char ch = s[i];
        if( ch =='(' ){
            st.push(i);
        }
        else{
            st.pop();
            if(st.empty()){
                st.push(i);
            }else{
                int len = i - st.top();
                maxLen = max(len, maxLen);
            }
        }

    }
    return maxLen;
```

★ Next smaller Element



-1 means koi smallest nahi hai

I/P	2	1	4	3
	0	1	2	3

L → R approach

O/P {1, -1, 3, -1}

$O(n^2)$

```
vector<int> v;
v.push_back(2);
v.push_back(1);
v.push_back(4);
v.push_back(3);

stack<int> st;
st.push(-1);

vector<int> ans(v.size());

for(int i=v.size()-1; i>=0; i--){
    int currEle = v[i];

    while(st.top() >= currEle){
        st.pop();
    }
    ans[i] = st.top();
    st.push(currEle);
}
```

$O(n)$

R → L approach

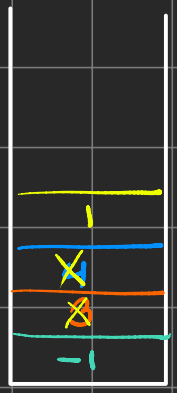
① Stack push (-1)

② array → k (R → L)

↳ stack k andar chota ele find karo jab tak nahi milta pop karte raho

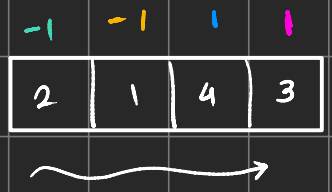
↳ milte he store karo

③ array → k push in stack



initial put it bcz for last index (-1) is going to be answer everytime

* prev smaller




```cpp
vector<int> findPreviousSmallerElements(const vector<int>& v) {
    stack<int> st;
    st.push(-1);
    vector<int> ans(v.size());

    for (int i = 0; i <v.size(); i++) {
        int currEle = v[i];
        while (st.top() >= currEle) {
            st.pop();
        }
        ans[i] = st.top();
        st.push(currEle);
    }

    return ans;
}
```

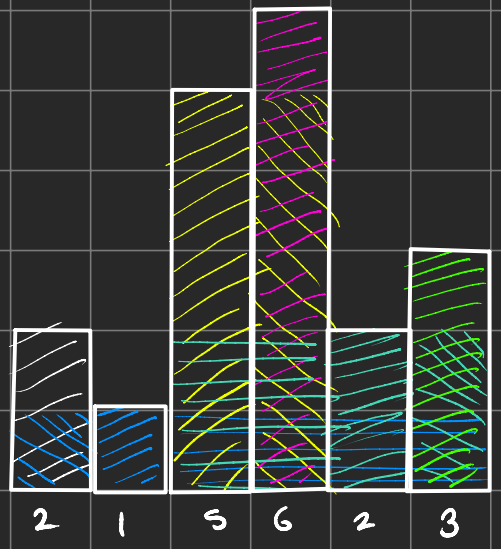
→ $O(2n)$
$\approx O(n)$

↳ total n element hai to har ek element par ek hi baar gaya ho → push n + pop n = $2n$

→ sorted kar lo then next daal do
but $O(n \log n) > O(n)$

two loop lagne se (n^2) o nahi ho jata

✱ Largest Element in Histogram



A1 = 2
A2 = 6
A3 = 10 → max
A4 = 6
A5 = 8
A6 = 3

extend tabhi hoga
jab adjacent bar ki
height >= tumhari
height

minus one → prev
size → next
to find difference

	2	1	5	6	2	3
	0	1	2	3	4	5
prev smaller ele. index	-1	-1	1	2	1	4
next smaller ele index	1	~~6~~	4	4	~~6~~	~~6~~

→ in next use length (size of array) instead of (-1)

width = next - prev - 1
= 1 - (-1) - 1
= 1

w
= 6 - (-1) + 1
= 6

= 4 - 1 - 1
= 2

= 4 - 2 - 1
= 1

= 4

= 6 - 4 - 1
= 1

Area = w x h
= 1 x 2

A = 6 x 1 = 6
A = 2 x 5 = 10
A = 1 x 6 = 6
A = 4 x 2 = 8
A = 1 * 3 = 3

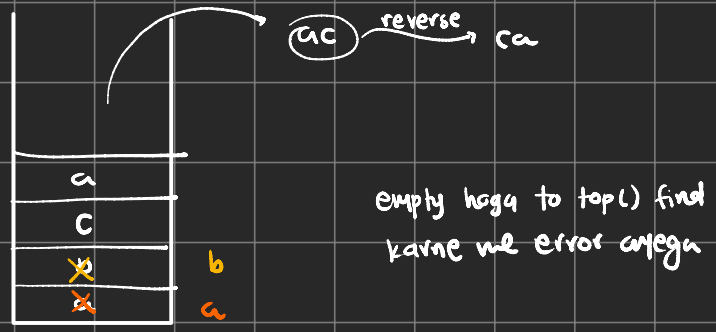
```cpp
int getRectangularAreaHistogram(vector<int> &height){

    vector<int> prev = findPreviousSmallerElements(height);
    vector<int> next = findNextSmallerElements(height);

    int maxArea = INT_MIN;

    for(int i=0; i<height.size(); i++){
        if(next[i]==-1){
            next[i]=height.size();  // -1 to size
        }

        int len = height[i];
        int width = next[i]-prev[i]-1;
        int area = len*width;
        maxArea = max(maxArea, area);
    }

    return maxArea;
}
```

```cpp
vector<int> findPreviousSmallerElements(const vector<int>& v) {
    stack<int> st;
    st.push(-1);
    vector<int> ans(v.size());

    for (int i = 0; i <v.size(); i++) {
        int currEle = v[i];
        while (st.top()!=-1 && v[st.top()]>= currEle) {
            st.pop();
        }
        ans[i] = st.top();
        st.push(i); // inserting index
    }

    return ans;
}
```

```cpp
vector<int> findNextSmallerElements(const vector<int>& v) {
    stack<int> st;
    st.push(-1);
    vector<int> ans(v.size());

    for (int i = v.size() - 1; i >= 0; i--) {
        int currEle = v[i];
        while (st.top()!=-1 && v[st.top()]>= currEle) {     // acess ele from top index
            st.pop();
        }
        ans[i] = st.top();
        st.push(i); // index insert
    }

    return ans;
}
```

Remove all adjacent duplicates in string

a b b a c a → a a c a → c a



empty hoga to top() find karne me error aayega

```
string ans = "";
stack<char> st;

for(int i=0; i<s.length(); i++){
    if(!st.empty()){
        if( st.top() == s[i] ){
            st.pop();
        }else{
            st.push(s[i]);
        }
    }else{
        st.push(s[i]);
    }
}

while(!st.empty()){
    cout << st.top() << endl;
    ans.push_back(st.top());
    st.pop();
}
reverse(ans.begin(), ans.end());
return ans;
```

Minimum bracket Reversal

s =) (( )) (((

0 1 2 3 4 5 6 7

→ ((( )) ( ))

0 1 2 3 4 5 6 7

① ((→ same character

↳ ( ) → (reverse) switch one

②)(→ diff character

↳ ( ) → reverse both

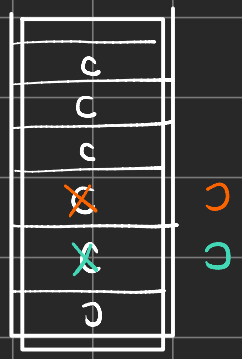
① valid paranthesis logic will be used

↳ I will remove valid pairs

② if stack is non-empty → we will try to find reversal count

① CCC → string size odd → so we cannot make pairs return (-1)

} C C } } C C C
0 1 2 3 4 5 6 7



valid Paranthesis Logic

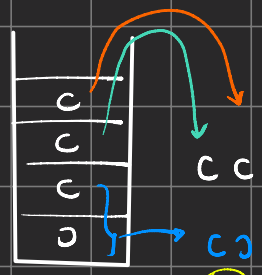
if (ch == '{') → push

else ↳ if (!s.empty && s.top() == '{') → pop

else → push

↳ after this all remaining one are

```
void validParanthesis(stack<char> &st, string s){
    for(int i=0; i<s.length(); i++){
        if(s[i]=='{'){
            st.push('{');
        }
        else{
            // if '}' closing bracket he to
            if( !st.empty() && st.top() == '{' ){
                st.pop();
            }
            else{
                st.push('}');
            }
        }
    }
}

int countRev (string s)
{
    // your code here

    if( s.size()&1 ) return -1; // size odd then

    int count = 0;
    stack<char>st;

    validParanthesis(st, s);

    while(!st.empty()){
        char one = st.top();
        st.pop();
        char two = st.top();
        st.pop();

        // if( one=='{' && two=='{' ){
        //     count = count+1;
        // }else if( one == '}' && two == '}' ){
        //     count = count+1;
        // }else{
        //     count = count+2;
        // }

        if(one==two){
            count = count+1;
        }else{
            count = count+2;
        }
    }

    return count;
```



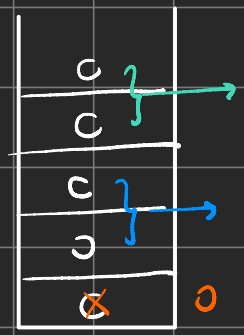
C C → same char case → count+=1

C } → opposite actually → count+=2

} C → actually this case

because after valid paranthesis we are not going to get C } type of paranthesis bez they are already validate in this valid paranthesis Logic

② C } } C C C
0 1 2 3 4 5

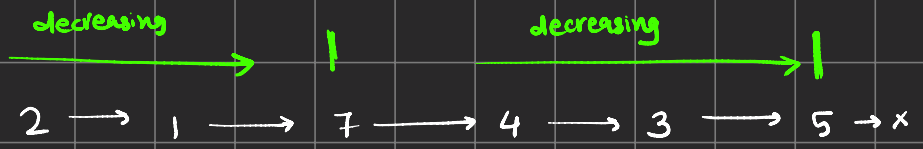


CC → same so count+1

}C → diff so count+2

next greater element in linked list

5	5	0



LL →

2	1	7	4	3	5
0	1	2	3	4	5

ans →

7	7	0	5	5	0


```
if ( ll[i] > ll[st.top()] )
    ↳ while ( ll[st.top()] < ll[i] )
          → ll[st.top()]'s greater element will be ll[i]
          st.pop()
Else
    ↳ st.push(i)
```

stack me jitne elements hai un sabka element ll[i]

and for remaining indexes put 0 as value

saving Index = bcz we need to access eles afterwards

```
vector<int> nextLargerNodes(ListNode* head) {
    vector<int> ll;
    ListNode* temp = head;

    while (temp != NULL) {
        ll.push_back(temp->val);
        temp = temp->next;
    }

    stack<int> st;
    vector<int> ans(ll.size(), 0);

    for (int i = 0; i < ll.size(); i++) {
        while (!st.empty() && ll[i] > ll[st.top()]) {
            int index = st.top();
            st.pop();
            ans[index] = ll[i];
        }
        st.push(i);
    }

    return ans;
}
```

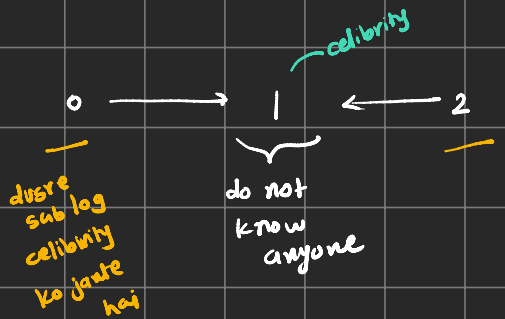
★ celebrity problem

3 person total

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

a[0][1] = 1 → 0th person knows 1st person

a[2][1] = 1 → 2nd person knows 1st person



BruteForce

(a) celibrity → rows sab 0's hogi (wo kisi ko nahi janta)

&& (b) celebrity → col's all has 1's (so all knows celebrity)

(M2) ① put all person in stack

② while (st.size != 1)

A → s.top()

B → s.top()

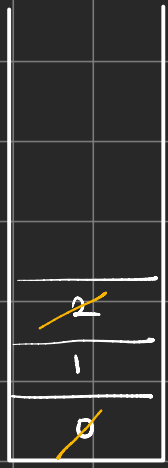
if (A knows B) → means A is not celebrity so, discard A, pushed B again

else → B knows A means B is not celebrity, discard B, pushed A again

③ That single peson in stack might be celebrity so verify it

```cpp
int celebrity(vector<vector<int> >& mat) {
    // code here

    int n = mat[0].size();
    stack<int> st;

    for(int i=0; i<n; i++){
        st.push(i);
    }

    while( st.size() != 1 ){
        int personA = st.top(); st.pop();
        int personB = st.top(); st.pop();

        if( mat[personA][personB] == 1 ){
            st.push(personB);
        }else{
            st.push(personA);
        }
    }

    int mightBeCelebrity = st.top(); st.pop();

    for(int i=0; i<n; i++){
        if( mat[mightBeCelebrity][i] != 0 ) return -1;
    }

    for(int i=0; i<n; i++){
        if( mat[i][mightBeCelebrity] == 0 && i!=mightBeCelebrity ) return -1;
    }

    return mightBeCelebrity;

}
```

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0



↳ persons index

→ A=2, B=1

if (M[A][B] == 1)

↳ means A know B

2 is not celebrity so discard

push (1)

→ A=1, B=0

if (M[i][0] == 1) ↳ false

else ↳ B knows A so discard B, push (1)

→ 1 is in stack so might be celebrity

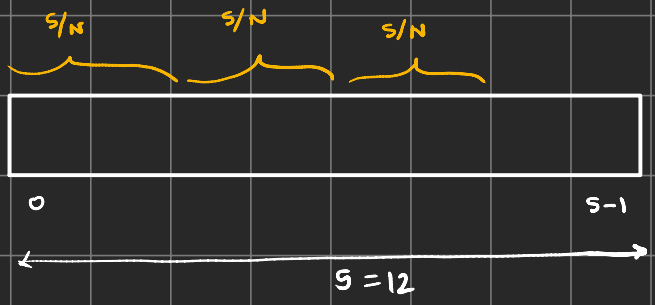
ⓐ 1 ki rows sab 0 honi chiye → T

ⓑ 1 ki col sab 1 honi chiye (celebrity wali row ko chod kar) → T

Ans 1

N stacks in array

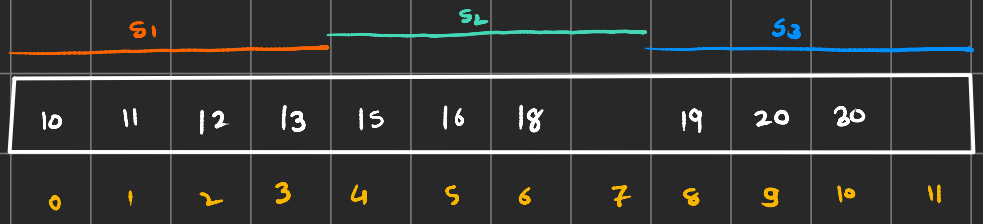
N → no of stacks
S → size of array



one part size = $\frac{S}{N}$ → you got N stacks and some parts

so implement stack for those all parts

→ $\frac{S}{N} = \frac{12}{3} = 4$ parts



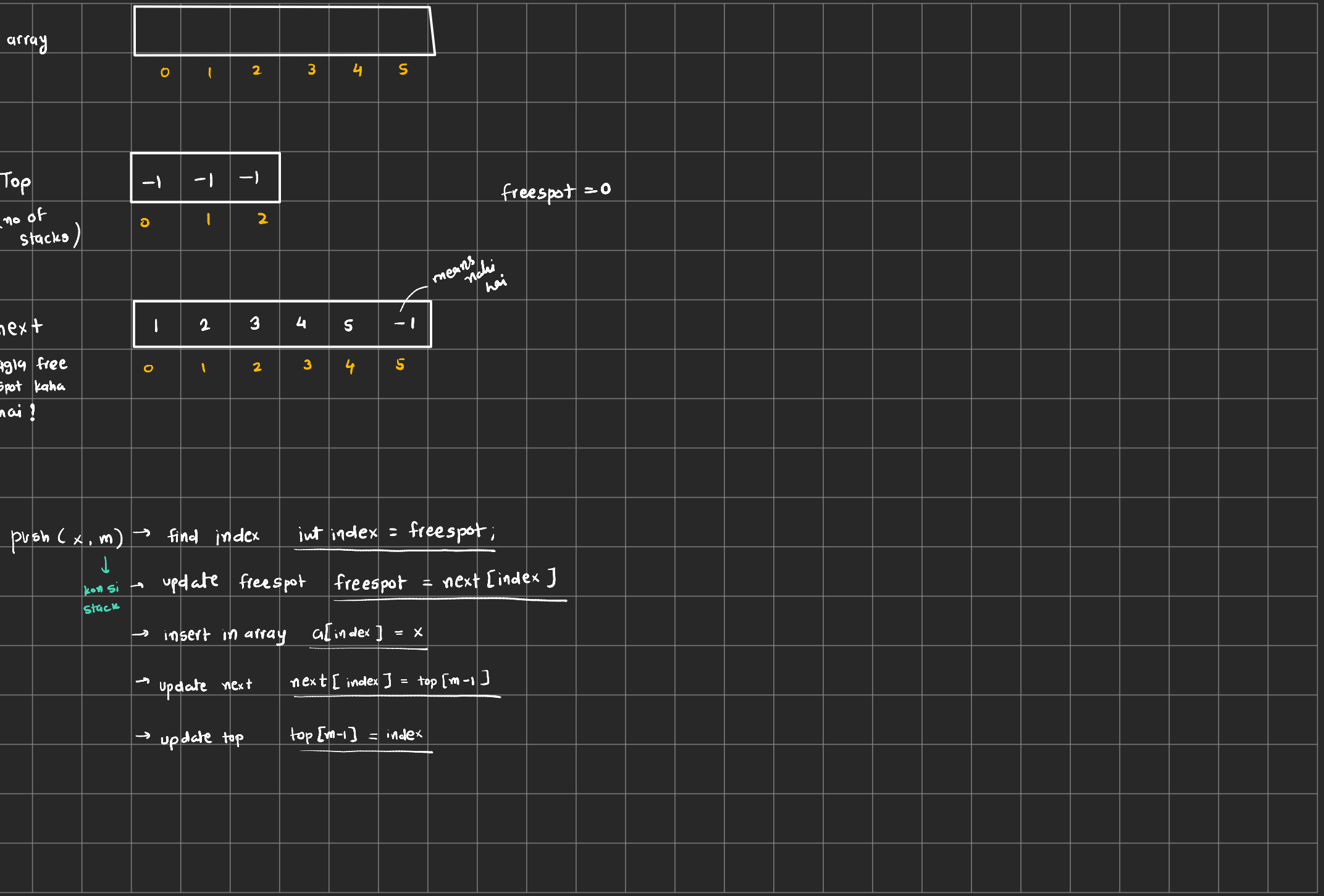
→ some array boxes are empty so it is not space optimised for stack means there is fragmentation issue

M-2 two additional arrays

① top[] → size → N → it stores index of top element of ith stack

② next[] → (a) it can point to next element after top element
(b) it can point to next free space

N → no of stacks → 3 → S_1 S_2 S_3
S → size of array → 6



Online stock span

→ minimum consecutive days for which the stock price was lesser than or equal to current day

stockSpanner

1 ← Next → 7

1 ← Next → 2

1 ← Next → 1

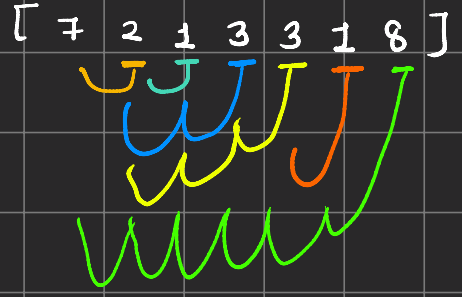
3 ← Next → 3

4 ← Next → 3

1 ← Next → 1

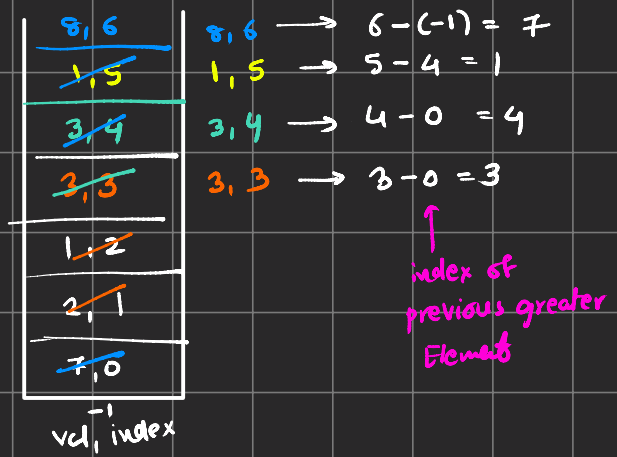
7 ← Next → 8

BF.




```
arr.add(value)

count = 1

for (i = currentArrSize-2 → 0)
{
    if (arr[i] <= value) count++
    else break
}

return count
```

[7 2 1 3 3 18]
0 1 2 3 4 5 6




```cpp
class StockSpanner {
public:

    stack< pair<int,int> > st;
    int index = -1;

    StockSpanner() {
        index = -1;
    }

    int next(int price) {

        index = index+1;
        while( !st.empty() ){
            if( st.top().first <= price ){
                st.pop();
            }else{
                break;
            }
        }

        int prevGreaterIndex = st.empty() ? -1: st.top().second;
        pair<int,int> p;
        p.first = price;
        p.second = index;
        st.push(p);

        int ans = index - prevGreaterIndex;
        return ans;
    }
};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner* obj = new StockSpanner();
 * int param_1 = obj->next(price);
```

st.push({price, index})

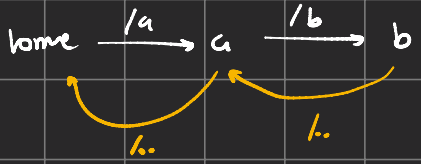
★ Simplify path

/a/./b/../../c/ — ignore last /

→ directory pushing and poping.

① /a/./b

② /../..

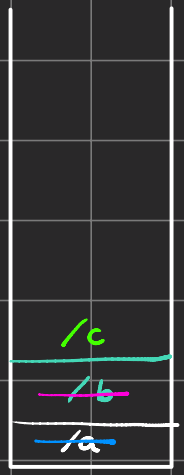


③ /c

simplify "/c"

/a/./b/../../c/

do nothing — last / ignore



/.. means pop

/.. means pop

```
string simplifyPath(string path) {
    stack<string> st;
    int i=0;
    while(i<path.length()){
        int start = i;
        int end = i+1;
        while(path[end] != '/' && end<path.size() ){
            end++;
        }

        string minPath = path.substr(start, end-start);
        cout << minPath << endl;

        i = end;

        if( minPath == "/" || minPath == "/." ){
            continue;
        }

        if( minPath != "/.." ){
            st.push(minPath);
        }else if( !st.empty() ){
            st.pop();
        }
    }
    string ans = st.empty() ? "/" : "";
    buildReverseAns(st,ans);
    return ans;
```

```
void buildReverseAns( stack<string> &st, string &ans ){
    if( st.empty() ){
        return;
    }
    string minPath = st.top(); st.pop();
    buildAns(st,ans);
    ans+=minPath;
}
```

check if word is valid after substitutions

given string ⑤

valid = t = ""

can transform t → s

after doing below operations : ① t → "abc" insert

s = aabcbc

① t = "" → tleft = "" tright = ""

t = abc

② t → t1 = a tr = bc

t = t1 + abc + tr

= aabcbc → valid

s = abcabc ababcc

① t = "" , t1 = "" tr = ""

t = abc

② t = abc → t1 = abc tr = ""

t = abcabc"" = abcabc

③ t → t1 = abcabc tr = abc

t = abcabcabc

④ t → t1 = abcabcab tr = c

t = abcabcab abcc → valid

s = abccbca

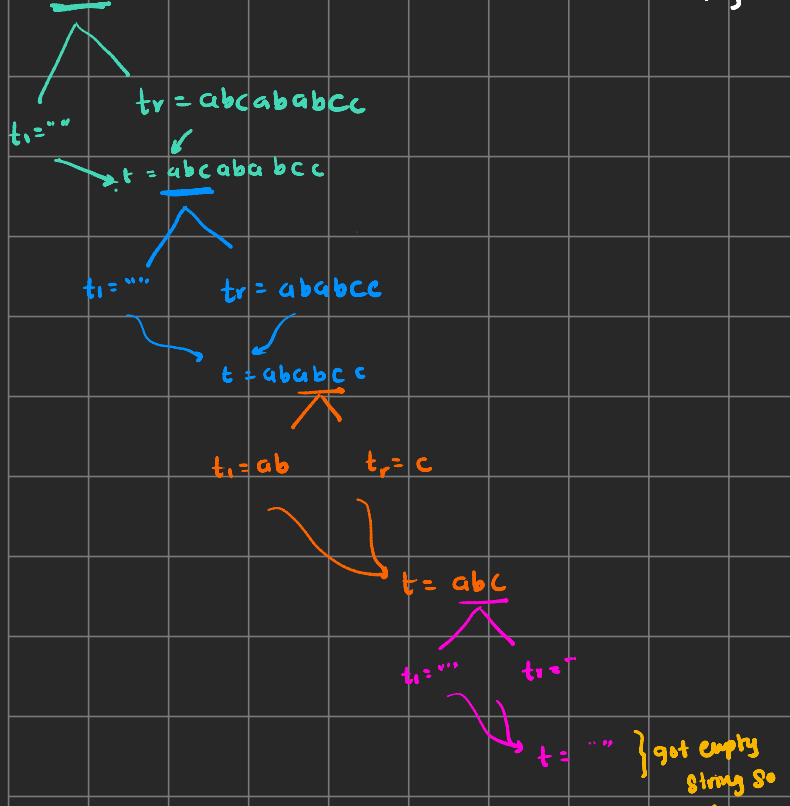
t = "" → t1 = abc t2 = ""

t = abc

t → t1 =

I cannot make it false not valid

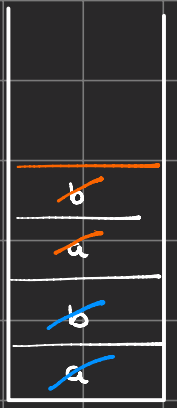
abcababcc → lets make empty string from S




```
bool isValid(string s) {

    // bc
    if( s.size() ==0 ){
        return true;
    }

    // rr
    int find = s.find("abc");
    if( find != string::npos ){
        // found
        string tl = s.substr(0,find);
        string tr = s.substr(find+3, s.length());

        bool ans = isValid(tl+tr);
        return ans;
    }else{
        return false;
    }

}
```

O(n²) Recursion

a b a b c c



Stack top par 'a' pada hoga tabhi 'b' push hoga

end - stack empty so ans True.

```
bool isValid(string s) {

    if (s[0] != 'a') {
        // means string ki starting he 'b' or 'c' se ho rhi
        return false;
    }
    stack<char> st;

    for (int i = 0; i < s.length(); i++) {
        char ch = s[i];
        if (ch == 'a') {
            st.push(ch);
        } else if (ch == 'b') {
            if (!st.empty() && st.top() == 'a') {
                st.push(ch);
            } else {
                return false;
            }
        } else {
            // ch = 'c' hai to
            if (!st.empty() && st.top() == 'b') {
                st.pop();

                if (!st.empty() && st.top() == 'a') {
                    st.pop();
                }
            }
            else{
                return false;
            }
        }
    }

    return st.empty() ? true : false;
}
```

Decode string

3[a]2[bc]

3 times a 2 times bc

aaabcbc

3[a2[c]]

inside case.

3[acc]

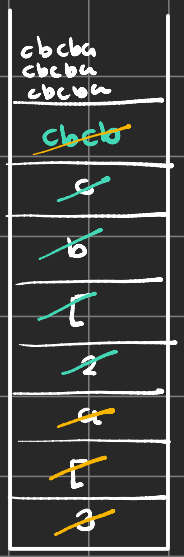
accaccacc

by seeing this example we can see that firstly we need to do inside one operations so in this types of case we can assume that we need to use stack

2[abc]3[cd]ef

abcabccdcdcdef

3[a2[bc]]



cbcb + a ③ → cbcbacbcbacbcba

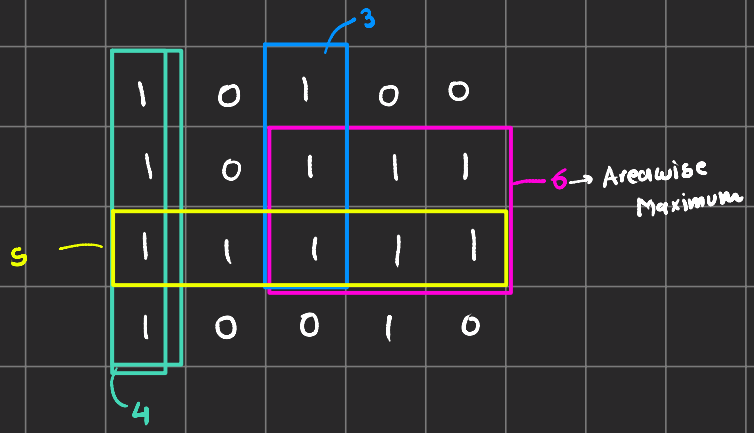
reverse → abcbcabcbcabcbc

cb 2

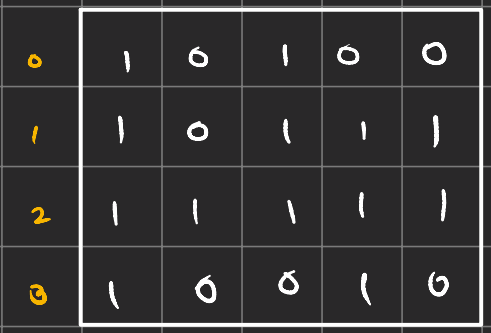
2 times repeat cb

```
string decodeString(string s) {

    stack<string> st;
    for(auto ch: s){
        if( ch == ']' ){
            string stringToRepeat = "";
            while( !st.empty() &&  !isdigit(st.top()[0]) ){
                string top = st.top();
                stringToRepeat = stringToRepeat + ( top=="[" ? "" : top);
                st.pop();
            }

            string numericTimes = "";
            while( !st.empty() &&  isdigit(st.top()[0]) ){
                string top = st.top();
                numericTimes = numericTimes + top;
                st.pop();
            }// we got "321" types of digits that we need to convert to "123"
            reverse(numericTimes.begin(), numericTimes.end());

            int n = stoi(numericTimes); // string to int

            // final decoding
            string currentDecode ="";
            while(n--){
                currentDecode = currentDecode + stringToRepeat ;
            }
            st.push(currentDecode);


        }else{
            string temp(1,ch);  // converting ch to string
            st.push(temp);
        }
    }

    string ans = "";
    while(!st.empty()){
        ans = ans + st.top();
        st.pop();
    }
    reverse(ans.begin(), ans.end());

    return ans;
}
```

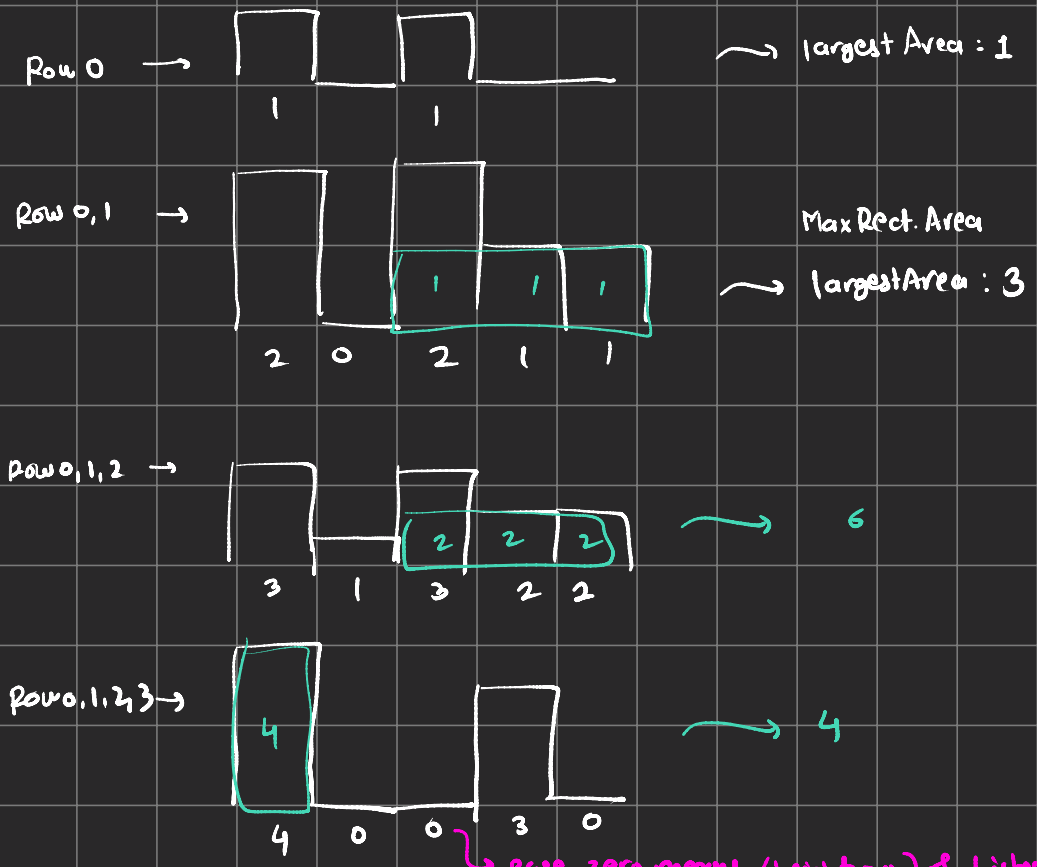
☆ Max Rectangle in Binary Matrix



Row wise Approach



Histograms

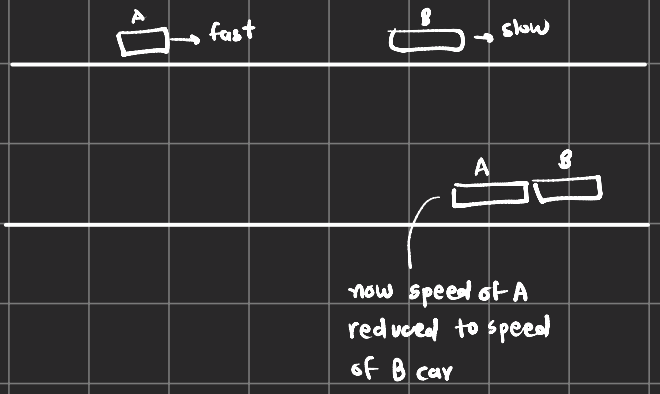



```
int maximalRectangle(vector<vector<char>>& matrix) {
    vector< vector<int> > v;    // current row
    int n = matrix.size();
    int m = matrix[0].size();

    for(int i=0; i<n; i++){
        vector<int> t;
        for(int j=0; j<m; j++){
            t.push_back(matrix[i][j] - '0');    // converting to int
        }
        v.push_back(t);
    }

    int area = getRectangularAreaHistogram(v[0]);
    for(int i=1; i<n; i++){
        for(int j=0; j<m; j++){
            // lets update current row with previous values
            if( v[i][j] ){
                v[i][j] = v[i][j] + v[i-1][j];
            }else{
                // base 0 wala case
                v[i][j] = 0;
            }
        }
        area = max(area, getRectangularAreaHistogram(v[i]) ) ;
    }

    return area;
}
```

Car Fleet

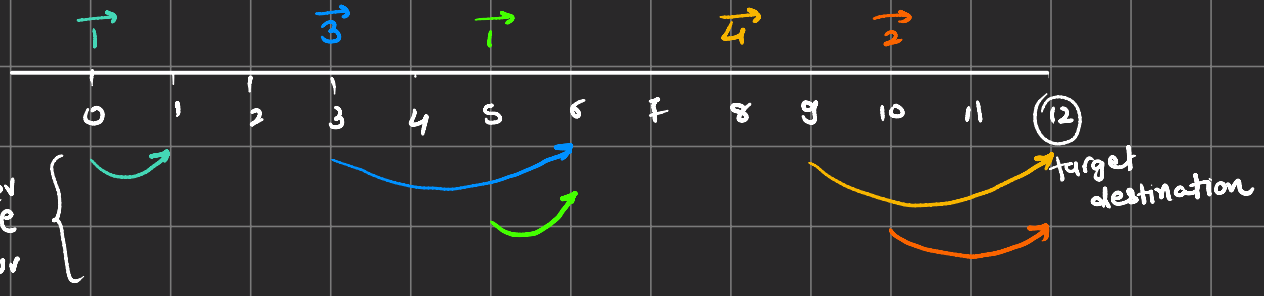
↳ no overtaking

Bumper to bumper drive at same speed



pos = 10 | 8 | 0 | 5 | 3

speed = 2 | 4 | 1 | 1 | 3

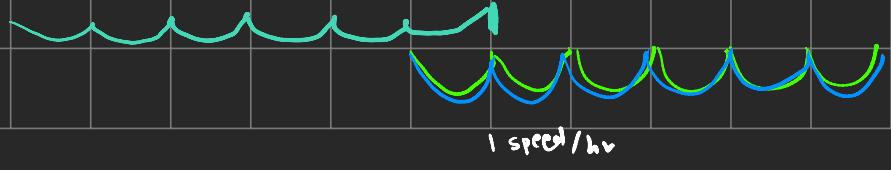


{4, 2} →

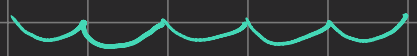
3 piche se aayi hai

1 already aage hai

so bumper to bumper drive hogi means already aage (right) wali car ki speed = dono car ki speed



{3, 1} → 2nd set



{1} → 3rd set

means 3 set destination par pahuchega

{ {4,2} , {3,1} , {1} }
① ② ③

→ sort positions

| 0 | 3 | 5 | 8 | 10 |

← so we can know which car is in left and right

make speed according to it

| 1 | 3 | 1 | 4 | 2 |

Target = 12

$$time = \frac{dist}{speed} = \frac{target - position[i]}{speed[i]}$$

absolute Time

↳ only for that particular single car if car is alone

| 12 | 3 | 7 | 1 | 1 |

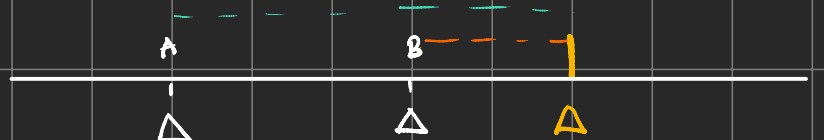
12 Hr se jyada time lene wali koi car nahi hai

Single fleet

{12} sabse last me pahuchegi

{3,7} ⑦

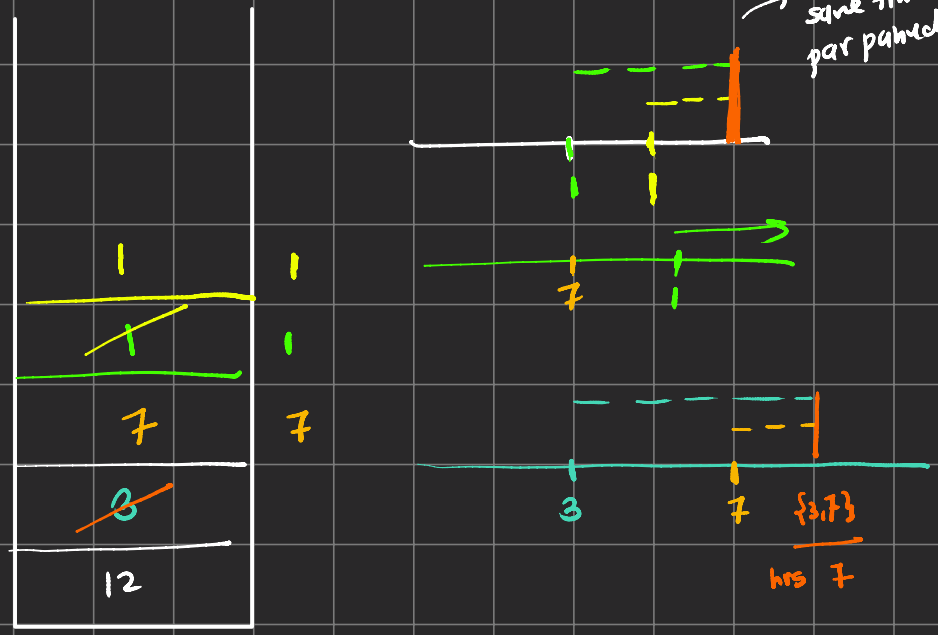
{1,1} ①



12 Hr time to reach to destination if single car fast

15 Hr means B is taking long time to reach destination slow

A,B Both meets So speed of B = speed of A and speed of B



→ dono car same time par pahuchegi

```
while (time >= st.top())
    st.pop()
st.push(time)
```

{1, 7, 12}

1 hr me pahuch ne wali cars

{1,1} {3,7} {12}

Pos {8,10} {3,5} {0}

```
class Solution {

    class Car{
        public:
            int pos,speed;

            Car(int p, int s): pos(p), speed(s) {}
    };

    static bool myComp(Car&a, Car& b){
        return a.pos < b.pos;
    }

public:
    int carFleet(int target, vector<int>& position, vector<int>& speed) {

        vector<Car> cars;

        for(int i=0; i<position.size(); i++ ){
            Car car(position[i], speed[i]);
            cars.push_back(car);
        }

        sort(cars.begin(), cars.end(), myComp);

        stack<float> st;

        for(auto car: cars){
            float time = (target - car.pos) / ((float) car.speed);

            while(!st.empty() && time>=st.top()){
                st.pop();
            }
            st.push(time);
        }

        return st.size();

    }
};
```

⭐ car fleet II

Actually difficult

	0	1	2	3
pos →	3	5	6	9
speed →	4	4	3	1

is there any car ahead of me with which I can collide?

Collision Time: 3/2 | 1 | 1.5 | -1

last wali 3rd car to nahi takrayegi

0th car

0th → 1st → X
same speed

1st car = (6-5)/(4-3) = 1

2nd car = (9-6)/(3-1) = 3/2 = 1.5

0th → 2nd → V

= (6-3)/(4-3)

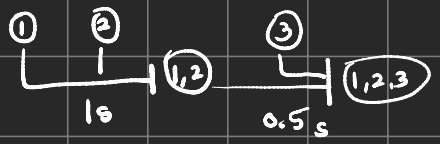
= 3s me takrayegi

But 2nd already 3s se pehle he 3rd se takraguyi hai

↑

1.5 s mei second car third se takra jati hai

But 1s me first car second se takra jayegi



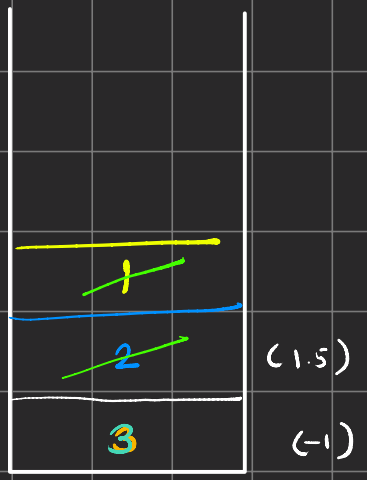
0th → 3rd se

= (9-3)/(4-1)

= 2

pos =	3	5	6	9
speed =	4	4	3	1
Time =	2	1	1.5	-1
	0	1	2	3

3 → sabse ahead wali car kisi se nahi takrayegi



stack me indexes save ho rahe ahead chalne wali car ke

if (stack top → speed >= currentCar)

→ means aage chalne wali car fast hai meri carse collision nahi hoga

s.pop()

0th → 1st : 4 >= 4 → T

0th → 2nd : 4 >= 3 → False

collide hogi

$= \frac{6-3}{4-3} = 3$ sec

current <= st.top

if (3 <= 1.5)

→ False

st pop { cannot update current car collision Time

0th → 3rd : $\frac{9-3}{4-1} = 2$ sec → if (2 <= -1) → T

invalid hai so

1st car

$= \frac{6-5}{4-3}$

$= 1$ sec

if (current collision time <= stack ke top ka collision time)

→ then current collision time = 1 sec

if there is any car which is ahead of is faster than me?

→ one car is ahead which is 3rd and speed is 1

2nd car speed is 3

→ 3 speed → 1 speed

car ahead of me is slower

→ collision can be happen

$= \frac{9-6}{3-1} = 1.5$

```cpp
vector<double> getCollisionTimes(vector<vector<int>>& cars) {

    vector<double> ans(cars.size(), -1);

    stack<int> st;

    for(int i=cars.size()-1; i>=0; i--){

        // check if ahead car is faster ?
        while(!st.empty() && cars[st.top()][1] >= cars[i][1] ){
            st.pop();
        }

        while(!st.empty() ){
            double collisonTime = (double)(cars[st.top()][0] - cars[i][0]) / (cars[i][1] - cars[st.top()][1]) ;
            if( ans[st.top()] == -1 ){
                ans[i] = collisonTime;
                break;
            }

            if( collisonTime <= ans[st.top()] ){
                ans[i] = collisonTime;
                break;
            }

            st.pop();
        }

        st.push(i);

    }

    return ans;

}
```

→ one question remaining N stacks in Array (lec 157)