# Valid Parenthesis

Input: String

characters: '(', ')', '{', '}', '[' and ']'

Conditions (Processing)

Valid String iff:

1.) Open bracket has a close bracket

2.) brackets must be closed in correct order

(i.e adjacent brackets must be closed internally first)

3) Every close bracket have a corresponding open bracket of same type.

Constraints

1.) $1 <= n <= 10^4$ ∴ Solution cannot be greater than $O(n^2)$

2.) 's' can contain only characters given.

Two corner cases can be handled through

⇒ `if (n == 1 && n%2 != 0)`
`return false;`

`if (n == 0)`
`return true;`

empty string is always valid.

lets think about the problem first

1.) For the problem we need to close adjacent brackets to each first, the internals must be closed first so that external might be closed

2.) One approach which is brute force can be to take two loops one of which checks if we have found a paired brackets or not (Why?) Because we want to remove them and then go to external bracket again.
· until a condition is true we can make the pseudocode as follows first

have to be replaced with some condition (described later

```
While (True)
{
    ...
    ...
    ...
    // turn condition off
    for i to n-1
    {   if (str[i] == '(' && str[i+1] == ')' ||
            str[i] == '[' && str[i+1] == ']' ||
            str[i] == '{' && str[i+1] == '}')
        {   str.erase(i, 2)
            // turn condition on
            break;
        }
    }
```

3) Now what should replace "True" and conditions I have placed., we can take a bool variable or a flag to check until when there are pairs found.

Modified pseudocode

```
bool isPaired = true;   int n = str.size();
while (isPaired)
{
    isPaired = false; // because right we have n't found any closing pair.

    for i to n-1
        if ( str[i] == '(' && str[i+1] == ')' ||
             str[i] == '{' && str[i+1] == '}' ||
             str[i] == '[' && str[i+1] == ']' )
        {
            str.erase(i, 2)
            isPaired = true; // as pair is found we need to restart the outer loop and break from current.
            break;
        }
}
return str.empty();
```

T: $O(n^2 \times m)$ ← no. of opening brackets

S: $O(1)$

Dry Run

[ [ ) ]
0 1 2 3

0 → 1 ✗
1 → 2 ✗
is replaced = false.

( )
0 1

0 → 1 ✓
is replaced = true
break out

∴ skip interval i ≠ n-1 (as str.size reduces) & both

# Dry Run For Brute force

{ [ ) }
0 1 2 3

is pairfound = ~~true~~ false ;

| i | n | i+1 |
|---|---|---|
| 0 | 4 | 1 |
| 1 | | 2 |
| 2 | | 3 |
| 3 | | |

0 → 1 ⊗ 4

1 → 2 ⊗ 4

is ~~pairfound~~ pairfound = false

breaks out of both loops.

( )
0 1

is pairfound = false

i = 0 | n = 2 | i+1 = 1

0 → 1 ✓ → remove (2) adjacent characters i & i+1

2 ⊗

length of string = '0'

is pairfound = true

i = 0 | n = 0 | i+1 = 1

(⊗ not possible as loop runs till n-1)

∴ returns str. empty ( )
which is true.

Optimized approach.

1.) In the brute force we were taking the external loop to revert back to the starting and removing the elements internally fast but it had two flaws.

1.) ~~so~~ modifying original string
(can be fixed by taking copy string)

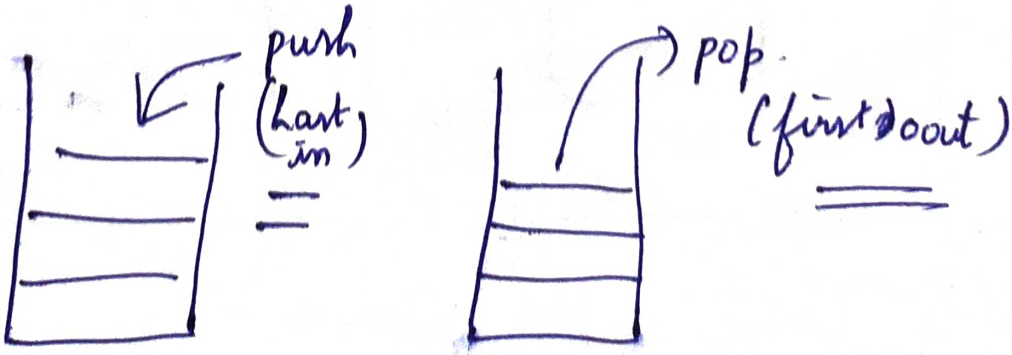
2.) Time complexity

2.) If we observe closely we can see that the last internal open element needs to be closed first in order to close other internal open elements coming before it.

" last elements closed before, ~~first~~ earlier elements to it ".

↳ hints ??

Let. me explain how we can use a data structure called stack




Thus stack might help in reverting to the internal brackets and even before it because it works that

Optimized approach

1.) Push if element is open bracket

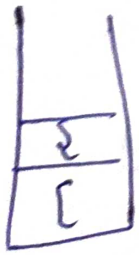
2.) check if stack is empty before pop operation.
as. element can't be popped from empty stack - and check the valid conditions.

if valid conditions are not true or stack is empty before popping
⇒ string is not valid

3) return. if stack is empty(), if so it means all elements got popped and stack became. empty. and false if elements are still left in stack

Three test cases

[{ )]



) ≠ {
⇒ false.

[{ 3 ]

3 ≡ { ✓
] ≡ [ ✓

stk.empty = true
returns.

{ 3 ]

3 ≡ { ✓
]

no element in stack to pop(·) or at top of

# Pseudocode

```
stack<char> st; // to store open braces.
int n = str.size;
~~while~~ {
~~for~~ if (n==0)
        return true;
    if (n==1 && n%2!=0)
        return false;
for it : str
    if (it=='(' || it=='{' || it=='[')
        st.push(it)
    else                          important.
    {   if (!st.empty() &&
            (it==')' && st.top()=='(')
            ||
            (it==']' && st.top()=='[')
            ||
            (it=='}' && st.top()=='{')
        )
        st.pop()
    else
        return false;
}
return st.empty();
```