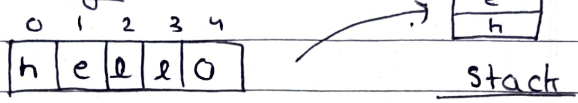
Date _____ Page _____

Day - 134

Stack - 2

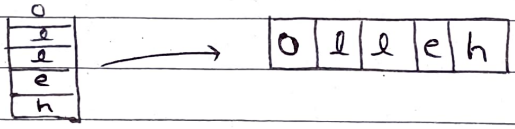
* **Reverse array:**



⇉ We have to solve this problem by using stack.

⇒ First, we create a stack and insert or push the elements one by one into the stack.

⇒ Now, we will pop the element one by one and insert into the array.

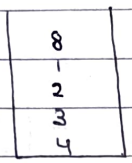


Code

```
stack <char> st;
for( i=0; i< s.size(); i++){
    st.push(s[i]);
}
~~for( i=0; i<s.size();~~
int i = 0;
while ( !st.empty()){
    ~~ans[i]~~ s[i] = st.top();
        i++;                    T.C → O(n)
    st.pop();                   S.C → O(n)
}
```

Date _____ Page _____

* Insert at Bottom:



int x = 2

⇒ We have insert the given element at the bottom of the stack.

⇒ For this, we have two options, we can use array or another temp stack to store the elements of original stack.

⇒ First, we will store the elements.

⇒ Then push the given element.

⇒ After that again push the elements into the original stack from temp stack.

Code

```
stack<int> temp;
while(!st.empty()){
    temp.push(st.top());
    st.pop();
}
st.push(x);
while(!temp.empty()){
    st.push(temp.top());
    temp.pop();
}
```

T.C. → O(n)

S.C. → O(n)

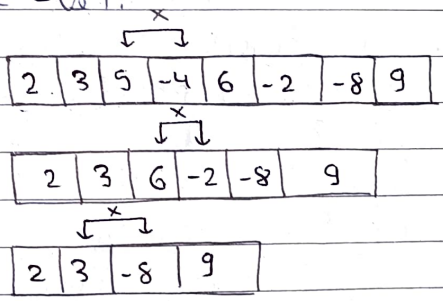
Date _____ Page _____

* Make the array beautiful:

2	3	5	-4	6	-2	-8	9

⇒ According to the question, only +ve & +ve adjacent or -ve & -ve adjacent numbers can come in array.

⇒ Otherwise, we have to destroy other type of adjacent pairs (-ve & +ve) or (+ve & -ve).



2 | 9 → answer

⇒ So, we will maintain neighbours and we will check the pointer value neighbour with neighbours top value.

⇒ If it follows the rule then we do not do any thing otherwise we will also delete the neigh. top element.

⇒ So, for neighbour, we can use stack data structure.

Date _____ Page _____

⇒ Also, if the element follows rule then we push it into the neighbour.

Code

```
stack <int> s;
for(i=0; i<arr.size(); i++){
    if(s.empty())
        s.push(arr[i]);
    else if(arr[i] >= 0){
        if(s.top() >= 0){
            s.push(arr[i]);
        }else{
            s.pop();
        }
    }
    else{
        if(s.top() < 0)
            s.push(arr[i]);
        else
            s.pop();
    }
}
vector <int> ans (s.size());
int i = s.size()-1;
while(!s.empty()){
    ans[i] = s.top();
    i--;
    s.pop();
}
return ans;
```

Date _____ Page _____

* **String Manipulation:**

ab	ac	da	da	ac	db	ea

⇒ Same as previous question.

⇒ But here, we have remove same adjacent pairs.

⇒ We will use the same approach as of the previous question.

Code

```
stack<string> s;
for( i=0; i<arr.size(); i++){
    if(s.empty())
        s.push(arr[i]);
    else if( s.top()== arr[i]){
        s.pop();
    else
        s.push(arr[i]);
}
return s.size();
```

* **Parenthesis is valid or not:**

str = (((())()))

⇒ We have to check the valid parenthesis i.e. (), (()), etc.

⇒ If a bracket is open then it should have their closing bracket for valid

Date _____ Page _____

parenthesis.

=> Also, for a closing parenthesis, there should be a starting or opening parenthesis.

Ex:  → Not valid

 → Not valid

=> str =  → Valid

=> So, we will maintain a stack.

=> If we get an opening bracket then we will push into the stack.

=> If we get an closing bracket then we will check with the top of stack.

=> If we will get an opening bracket then we will pop the top of the stack.

=> In the end, if we will get an empty stack then the string is valid otherwise not.

Code

```
bool check (string str){
    stack <char> s;
```

Date _____ Page _____

```
for(i=0; i<str.size(); i++){
    if(str[i] == '(')
        s.push(str[i]);
    else{
        if(s.empty())
            return 0;
        else
            s.pop();
    }
}
return s.empty();
}
```

T.C. → O(n)
S.C. → O(n)

⇒ We can solve this question in O(1) S.C.
⇒ We will count the opening bracket & when any closing bracket will come, we will decrease the count.

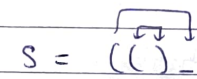
Code

```
bool check(string str){
    int left = 0;
    for(i=0; i<str.size(); i++){
        if(str[i] == '(')
            left++;
        else{
            if(left == 0) return 0;
            else left--;
        }
    }return left == 0;
}
```

Date _____ Page _____

* Minimum add to make parenthesis valid:

⇒ we have to return how many paranthe. we required to make the current str. valid.



S = (()_ → 1

S = __ (()))) → 2

⇒ For this, we will use stack for opening bracket and use a count variable for closing bracket that don't have opening bracket.

⇒ So, if a closing bracket come and we don't have any opening bracket then we increase the count by one.

⇒ In the end, we return count + size of stack.

⇒ Size of stack for those opening bracket that don't have closing bracket.

Code

```
stack<char> st;
int count =0;
for( i=0; i< s.size(); i++){
```

Date _____ Page _____

```
        if (s[i] == '(')
            st.push(s[i]);
        else {
            if( st.empty())
                count++;
            else
                st.pop();
        }
    }
    return count + st.size();
```

* **Valid Parentheses:**

⇉ s = (()) {} [()]

⇉ Here, we have three different types of brackets.

⇉ If the previous will not match with current bracket that means if the previous of the closing bracket is not opening bracket then we will return 0.

Code

```
stack <char> st;
for(i=0; i<s.size(); i++){
    if(s[i]=='(' || s[i]=='{' || s[i]=='[')
        st.push(s[i]);
    else {
        if(st.empty())
            return 0;
```

Date _____ Page _____

```
else if(s[i] == ')'){
    if(st.top() != '(')
        return 0;
    else
        st.pop();
}
else if(s[i] == '}'){
    if(st.top() != '{'){
        return 0;}
    else
        st.pop();
}
else{
    if(st.top() != '['){
        return 0;}
    else
        st.pop();
}
}
return st.empty();
```

Back space

* ~~Background~~ String Compare:

S = ab#c , t = ad#c

So, here, we have to check to string are equal or not.

In this question, if you got ~~an~~ a '#' then remove the back character.

Date _____ Page _____

⇒ After that, check the string, if they are equal return 1 otherwise 0.

* Print Bracket Number:

S = (aa(bdc))p(de)

⇒ 1 2 2 1 3 3 → 1 2 2 1 3 3

⇒ Here, in this question, we have to print the bracket no.

⇒ We will take a stack and a count variable.

⇒ Now, if we get an opening bracket then we will increase the value of count & push this value in stack of count.

⇒ After that, if we will get ~~at~~ a closing bracket then we will check the stack and print that no. & pop that no..

* Get min at pop:

⇒

2	1	3	5	0	6

⇒ We have to return the min element of stack at pop.

⇒ 0 0 1 1 1 2 → ans

Date _____ Page _____

and then

⇒ We can do that we will take a stack and by using it find the min element then print it & pop the element.

⇒ But this process ~~is~~ will increase the T.C upto $O(n^2)$.

⇒ So, for getting a faster approach, we will do that when we are pushing the element into the stack, we will only push the min element.

⇒ For ex: if the min element of the stack is 1 & 5 comes then we will push 1 into the stack.

⇒ So, in short, we can say, we only push the min. element.