Time Complexity Explanation of Post fix and Prefix

This is a seperate section to explain what actually the time complexity of Postfix and Prefix expression: –

More or less the code is same during prefix and postfix:

1. ***Stack st; i.e. creating a variable of Stack , hence it is done at constant amount of time.***

2. ***int len*** = ***strlen(infix);***

Now think of calculating the length of array of characters(also known as String) think of a function like:

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

int strlen (char *str) {
    int c = 0;
    for (int i = 0; str[i] != '\0'; i++)
    {
        c=c+1;
    }
    return c;
}
```

```
int main(){

    char str[100];
    cout<<"Enter the string: ";
    cin>>str;
    cout<<"Length of the string is:
"<<strlen(str)<<endl;

    return 0;

}
```

$\boldsymbol{Hence\ obviously\ when\ we\ are\ importing\ some\ inbuilt\ function}$ $\boldsymbol{i.e. strlen() \rightarrow which\ calculates\ length\ of\ a\ string\ takes\ `n`}$ $\boldsymbol{amount\ of\ time\ i.e. O(n).}$

$\boldsymbol{And\ as\ we\ know\ creating\ `n`\ length\ of\ array\ takes\ O(n) time}$ $\boldsymbol{complexity.}$

```
void create(Stack *st, int cap)
{
    st->size = cap;
    st->top = -1;
    st->s = (char *)malloc(st->size * sizeof(char));
}

create(&st, len);
```

$Hence\ create()function\ is\ creating\ an\ stack\ of\ array\ dynamically$ $of\ length\ `n`\ takes\ O(n)$.

$Next\ is\ declaration\ int\ i = 0, j = 0, which\ takes\ O(1) constant\ time.$ $For\ infix\ to\ prefix\ int\ i = len - 1, and\ int\ r, op1, op2\ takes$ $O(1)\ constant\ time.$

$But\ the\ real\ thing\ starts\ when\ we\ are\ traversing\ the\ array\ of$ $characters\ from\ first\ to\ last\ during\ infix\ to\ postfix\ conversion:$

```
while (infix[i] != '\0')
    {
        if (isOperand(infix[i]) == 1)
        {
            postfix[j] = infix[i];
            j++;
            i++;
        }
      …….
     }
```

$We\ know\ that\ the\ no.of\ inner\ statement\ of\ loop\ runs\ is\ the\ time$ $taken\ by\ the\ loop.$

$Now\ postfix[j] = infix[i]$, $takes\ O(1)\ constant\ time\ per\ run\ of$ $`if`\ and\ length\ of\ infix\ statement\ is\ `n`$, $postfix[j] = infix[i]\ run$ $n\ time.$

$Now\ if\ we\ take$:

```
 if (pre(infix[i]) > pre(st.s[st.top]) || isEmpty(st)
|| infix[i] == '(')
            {
                push(&st, infix[i]);
                i++;
            }
```

Push operation too take $O(1)$constant time ,now this statement is linked with next statement.

```
else if (pre(infix[i]) <= pre(st.s[st.top]) &&
infix[i] == ')')
            {
                while (st.s[st.top] != '(')
                {
                    postfix[j] = pop(&st);
                    j++;
                }
                pop(&st);
                i++;
            }
```

i.e. as soon as we find `(` ,postfix[j] = pop() ,i.e. the top element of the stack gets added to array by popping out till we find `)`.

Lets assume there are `k` amounts of `(` opening braces and `q` amount of `)` closing braces and `n` amount data inside those braces, Hence, postfix[j] = n where `p` and `k` will get excluded.

$Now\ if\ there\ is\ `s`\ no.of\ operand\ and\ `q`\ no.of\ operator\ outside\ consider\ free\ then:-$

$Postfix\ array\ will\ have:\ N-(p+k)-(s+q)=n\ ,N\ is\ the\ total\ length\ of\ the\ characters.$

$Hence\ the\ both\ loop\ constitutes\ the\ above\ equation\ which\ we\ get.$
$Now,$

```
else
            {
                while (pre(infix[i]) <=
pre(st.s[st.top]) && !isEmpty(st) && infix[i] != '('
&& infix[i] != ')')
                {

                    postfix[j] = pop(&st);
                    j++;
                }
                push(&st, infix[i]);
                i++;
            }
….
  while (!isEmpty(st))
    {
        postfix[j] = pop(&st);

        j++;
    }
```

$\boldsymbol{This\ will\ pop\ out\ and\ add\ \ `s`\ no.of\ operand\ and\ `q`\ no.of\ operator}$
$\boldsymbol{outside\ `()`\ hence\ it\ will\ be}$:

$$\boldsymbol{N-(n+p+k)=(s+q)}$$

$\boldsymbol{Now, if\ we\ see\ that\ postfix\ equation\ will\ have}$:

$$\boldsymbol{P=n+s+q.}$$

$\boldsymbol{Where\ if\ we\ include\ the\ push\ and\ pop\ the\ total\ iteration\ will\ occur}$:

$$\boldsymbol{N=(n+s+q)+(p+k)}$$

$\boldsymbol{or, N=P+(p+k), where\ P\ is\ postfix\ expression.}$

$\boldsymbol{Hence\ full\ iteration\ needed\ is: O(N), where\ N\ is\ the\ length\ of}$
$\boldsymbol{Postfix\ expression.}$

$\boldsymbol{At\ each\ condition\ push\ and\ pop\ occurred\ \ at\ constant\ time\ O(1)}$
$\boldsymbol{till\ the\ iteration\ it\ takes\ 1+1+1+..+N\ ,=O(N), where\ N\ is\ the}$
$\boldsymbol{Length\ of\ the\ array.}$

$\boldsymbol{This\ same\ thing\ repeats\ in\ Prefix\ expression.}$

$In\ Prefix\ expression\ we\ have\ an\ extra\ section\ , that\ is\ swapping$

$and\ interchanging\ of\ characters\ .$

```
  // Reverse the prefix expression to get the correct order
    int start = 0;
    int end = j - 1;
    while (start < end)
    {
        char temp = prefixExpr[start];
        prefixExpr[start] = prefixExpr[end];
        prefixExpr[end] = temp;
        start++;
        end--;
    }

    free(st.s);

    return prefixExpr;
}
```

$The\ while\ loop\ iterates\ over\ the\ entire\ prefix\ expression,$

$but\ not\ N\ i.e.\ length\ of\ the\ prefix\ expression\ as\ the\ reason:$

$Each\ statement\ inside\ loop\ runs\ at\ a\ constant\ time\ O(1).$

$But\ swapping\ occurs\ between\ two(2)element\ whose\ index\ is$

$`start`\ and\ `end`.$

$Hence\ whole\ complexity\ takes: \frac{n}{2}\ times. Hence\ time\ complexity:$

$$O\left(\frac{n}{2}\right) = \frac{1}{2} \times O(n) = O(n).$$