Alternative Approach of Stack – Time Complexity

1. ***ISFULL function to check the stack is full or not as discussed before*:**

```
int isFull(Stack st)
{
    if (st.top == st.size - 1)
    {
        return 1;
    }
    return 0;
}
```

2. ***Double Stack function to double the stack if the stack is full* :**

```
void doubleStack(Stack *st)
{
    st->size = st->size * 2;
    st->s = (int *)realloc(st->s, st->size *
sizeof(int));
}
```

ReAlloc will reallocate the memory again with a new size of the array created.

3. *In Push , when the element is pushed in Stack then its checked that if its full then double the stack , else just increment the top and push the element.*

```
void push(Stack *st, int x)
{

    if (isFull(*st))
    {
        doubleStack(st);
        st->top++;
        st->s[st->top] = x;
        return;
    }

    else{

        st->top++;
        st->s[st->top] = x;
    }


}
```

Now, Coming back to execution:

At first we create object of Stack and declare a variable to decide what should be capacity of the stack in main function.

```
int main()
{

    Stack st;
    int cap;
    cout << "Enter the capacity of the stack" << endl;

    cin >> cap;

}
```

4. We know during push it will take 2n elements as as the size gets doubled:

```
int main()
{

int elem;
  for (int i = 0; i < 2* cap; i++)
   {
   cout << "Enter the element to be pushed :" << endl;
   cin >> elem;
   push(&st, elem);
    }

}
```

5. ***Now pop function must pop the first set of elements that are (inserted before the array got doubled) :***

```
int pop(Stack *st)
{

    if (isEmpty(*st))
    {
        cout << "Stack Underflow" << endl;
    }

    return st->s[st->top--];
}

int main()
{

if (!isEmpty(st))
    {
       for (int i = 0; i < cap; i++)
          {
   cout << "Popped element is: " << pop(&st) << endl;
           }

      }
       else
       {
           cout << "Stack Underflow" << endl;

        }
}
```

Else , every other functions are all same earlier discussed .

Now coming to Time Complexity

1. ***Time Complexity of Push Operation*:**

Ans*: *As discussed earlier, Push Operation here takes $T(n) = O(n)$, ***where amortized time complexity*** $= O(1)$.

2. ***Time Complexity of Pop Operation*:**

Ans : As we are poping from each stack at n times, we will take the amortized time Complexity $= O(1)$.

And Other Time Complexity of all types of Operations remains same that is $O(1)$.

Now coming to Space Complexity

As the stack takes `n` times push takes n units of space in memory , hence space complexity $= T(n) = O(n)$.

`Hence, we sum up:`

Space complexity	$O(n)$
Creation of Stack	$O(1)$
Push	$O(1)$ ***Average***
Pop()	$O(1)$ ***Average***
Peek()	$O(1)$
IsEmpty()	$O(1)$
IsFull()	$O(1)$
DeleteStack()	$O(1)$