Find Running Time

$\rightarrow T(n)$

```
function( int n ) {
        if( n == 1 ) return;
        for(int i = 1 ; i <= n ; i + + )
                for(int j = 1 ; j <= n ; j + + )
                        printf("*");
        function( n-3 );
}
```

$\rightarrow$ constant $\rightarrow c$

$\rightarrow n$

$\rightarrow n$

$+$

$\rightarrow n^2$

$+$

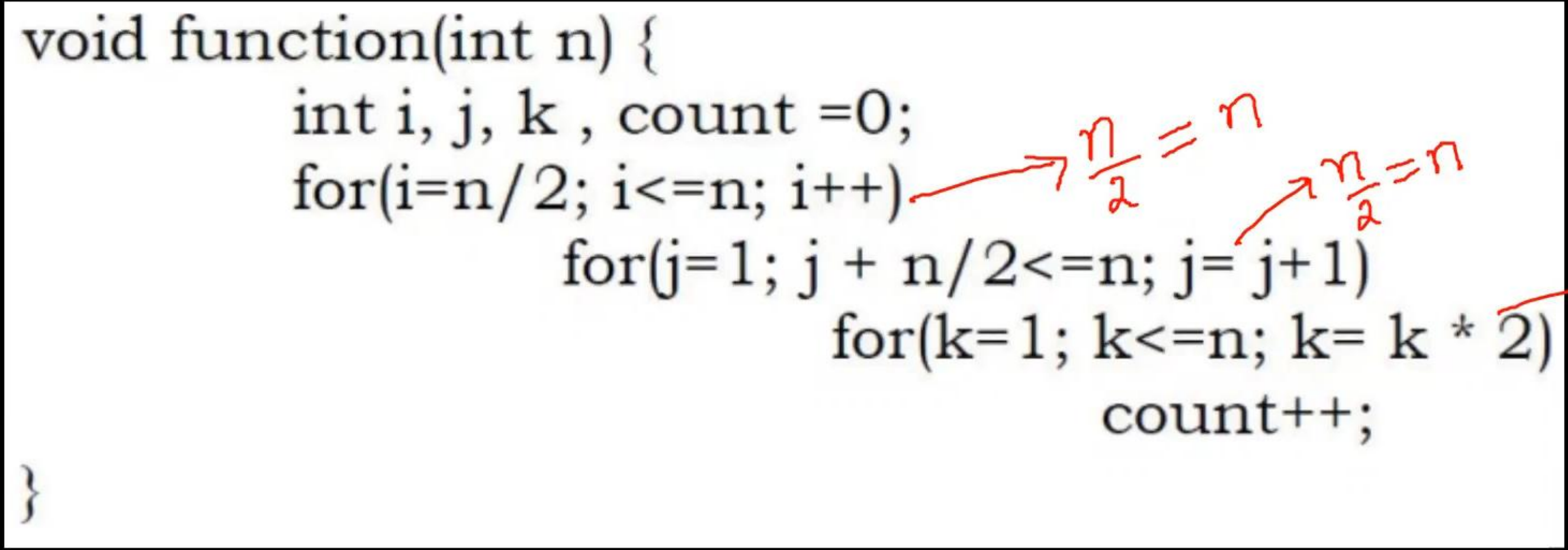
$\rightarrow T(n-3)$

$n \times (n^2) = O(n^3)$

$$T(n) = T(n-3) + n^2 = O(n^3)$$

$$\boxed{a=1,\ b=3,\ k^2}$$

Find Running Time



```
void function(int n) {
        int i, j, k , count =0;
        for(i=n/2; i<=n; i++)
                for(j=1; j + n/2<=n; j= j+1)
                        for(k=1; k<=n; k= k * 2)
                                count++;
}
```

$\to \frac{n}{2} = n$

$\to \frac{n}{2} = n$

$\to \log n$

$O(n^2 \log n)$

Find Running Time

```
void function(int n) {
          int i, j, k , count =0;
          for(i=n/2; i<=n; i++)
                    for(j=1; j<=n; j= 2 * j)
                              for(k=1; k<=n; k= k * 2)
                                        count++;
}
```

$\to \frac{n}{2} \approx n$

$\to \log n$

$\to \log n$

$O(n \log^2 n)$

$$\boxed{\log\log n \neq (\log^2 n) = (\log n)^2 \neq \log n^2}$$

Find Running Time

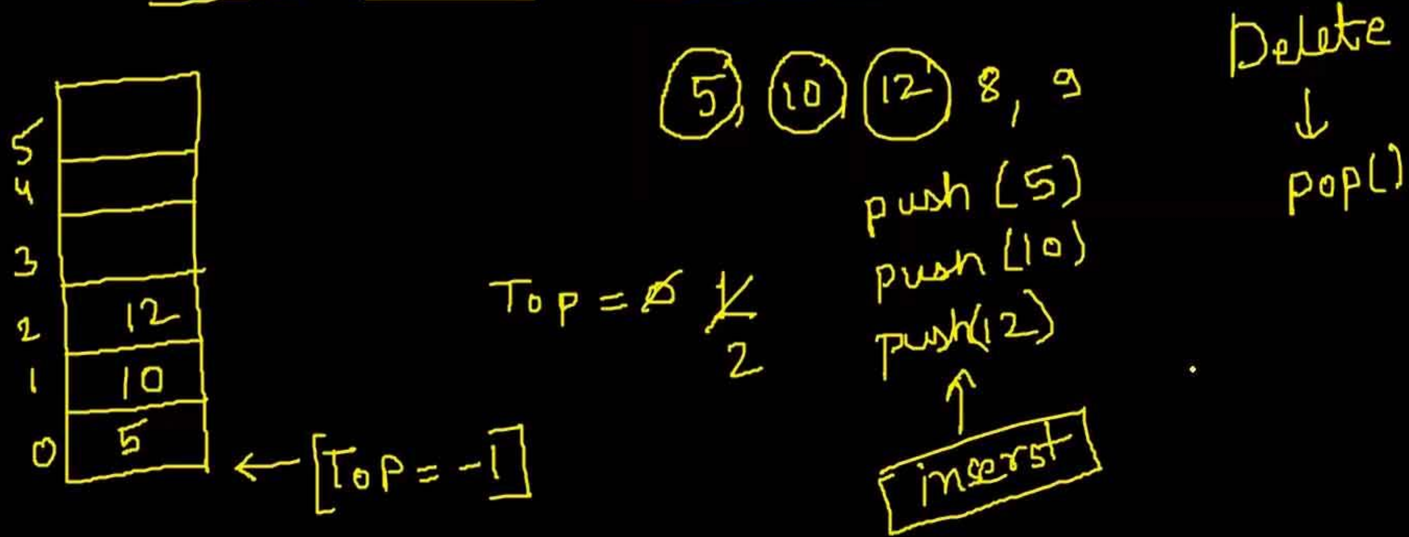
```
function( int n ) {
        if(n == 1) return;  → constant
        for(int i = 1 ; i <= n ; i + + ) {  → n
                for(int j= 1 ; j <= n ; j + + )
                        printf("*" );  → 1 (constant)
                        break;
                }
        }
}
```

O(n)

Stack

A stack is a simple data structure used for storing data. In a stack, the order in w
arrives is important.

Definition: A stack is an ordered list in which insertion and deletion are done at one end, called top. The last element inserted is the first one to be deleted. Hence, it is called the Last in First out (LIFO) or First in Last out (FILO) list.



Stack ADT

The following operations make a stack an ADT

- ✓ void Push(int data)
- ✓ int Pop()
- ✓ int Top()
- ✓ int Size()
- ✓ int isEmptyStack()
- ✓ int isFullStack()

Exceptions –

1) Overflow → stack full
2) Underflow → stack empty

```
class Stack
{
     int a[100];
     int Top;
  public:              → int size();
     void push(int data);
     int pop();  → delete(Top)
     int Top();  → return(Top)
     int isEmptystack() → 0 / 1
     int isFullstack → 0/1
};
```

isFullstack: 0 → Not full, 1 → full

isEmptystack: 0 → Not Empty, 1 → Empty

Applications

Following are some of the applications in which stacks play an important role.

- ✓ Balancing of symbols
- ✓ Infix-to-postfix conversion
- ✓ Evaluation of postfix expression
- ✓ Implementing function calls (including recursion)
- ✓ Finding of spans (finding spans in stock markets, refer to Problems section)
- ✓ Page-visited history in a Web browser [Back Buttons]
- ✓ Undo sequence in a text editor
- ✓ Matching Tags in HTML and XML

((a*b)+c)

ab*c+

- ✓ Auxiliary data structure for other algorithms (Example: Tree traversal algorithms)
- ✓ Component of other data structures (Example: Simulating queues, refer Queues chapter)

Implementation

There are many ways of implementing stack ADT; below are the commonly used

- Simple array-based implementation
- Dynamic array-based implementation
- Linked lists implementation

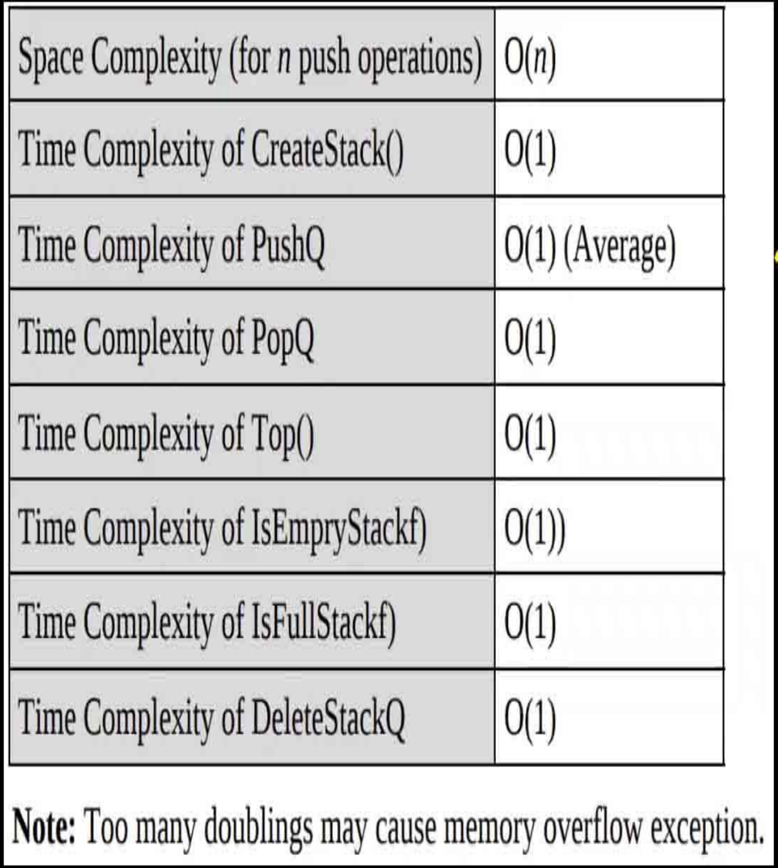
Array Implementation

Space Complexity (for n push operations)	O(*n*)
Time Complexity of Push()	O(1)
Time Complexity of Pop()	O(1)
Time Complexity of Size()	O(1)
Time Complexity of IsEmptyStack()	O(1)
Time Complexity of IsFullStackf)	O(1)
Time Complexity of DeleteStackQ	O(1)

Limitations-

The maximum size of the stack must first be defined, and it cannot be changed. Trying to push a new element into a full stack causes an implementation-specific exception.

Dynamic Array Implementation

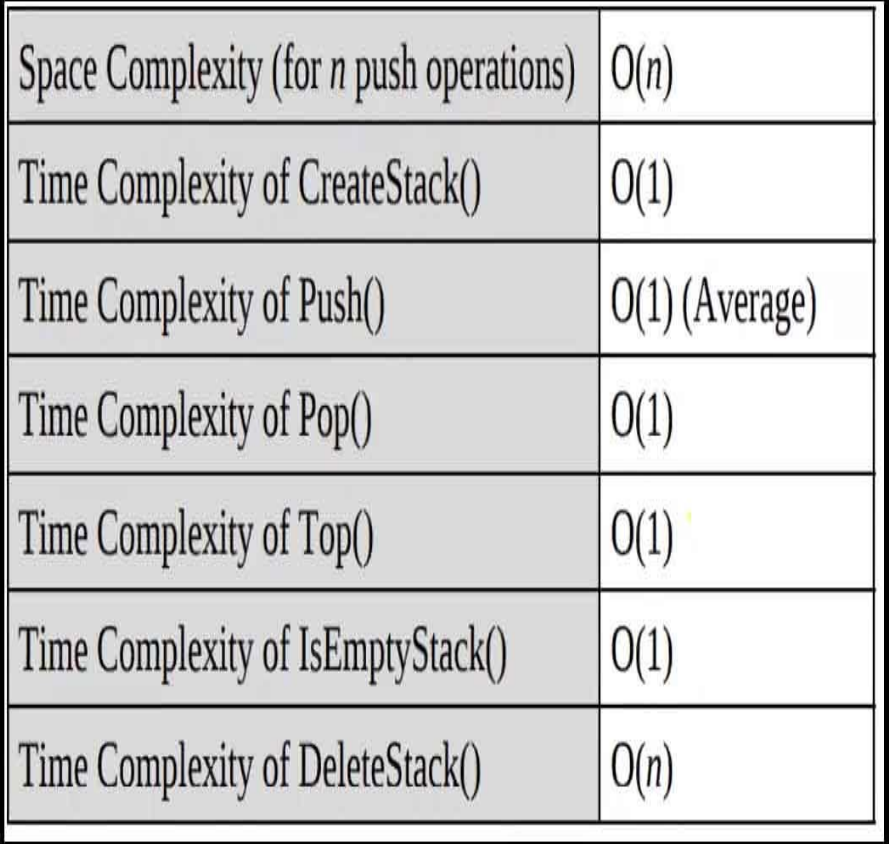


Space Complexity (for *n* push operations)	O(*n*)
Time Complexity of CreateStack()	O(1)
Time Complexity of PushQ	O(1) (Average)
Time Complexity of PopQ	O(1)
Time Complexity of Top()	O(1)
Time Complexity of IsEmpryStackf)	O(1))
Time Complexity of IsFullStackf)	O(1)
Time Complexity of DeleteStackQ	O(1)

Note: Too many doublings may cause memory overflow exception.

$\leftarrow O(n)$

Linked List Implementation



Space Complexity (for *n* push operations)	O(*n*)
Time Complexity of CreateStack()	O(1)
Time Complexity of Push()	O(1) (Average)
Time Complexity of Pop()	O(1)
Time Complexity of Top()	O(1)
Time Complexity of IsEmptyStack()	O(1)
Time Complexity of DeleteStack()	O(*n*)

Stacks – Problems & Solutions

Problem - postfix evaluation using stacks?

① 2 3 * 5 +

operands

operator

a + b * C

Infix Notation

+ a * b C

Prefix Notation

Postfix Notation

a b c * +

11

~~5~~

~~6~~

~~3~~

~~2~~

$3 * 2 = 6$

$5 + 6 = 11$

$\boxed{11}$ ✓

② $(2 + (3 * 5)) * 6$

3 5 * 2 + 6 *

$3 * 5 = 15$

$2 + 15 = 17$

$6 * 17$

$\boxed{102}$

~~6~~

~~17~~

~~2~~

~~15~~

~~5~~

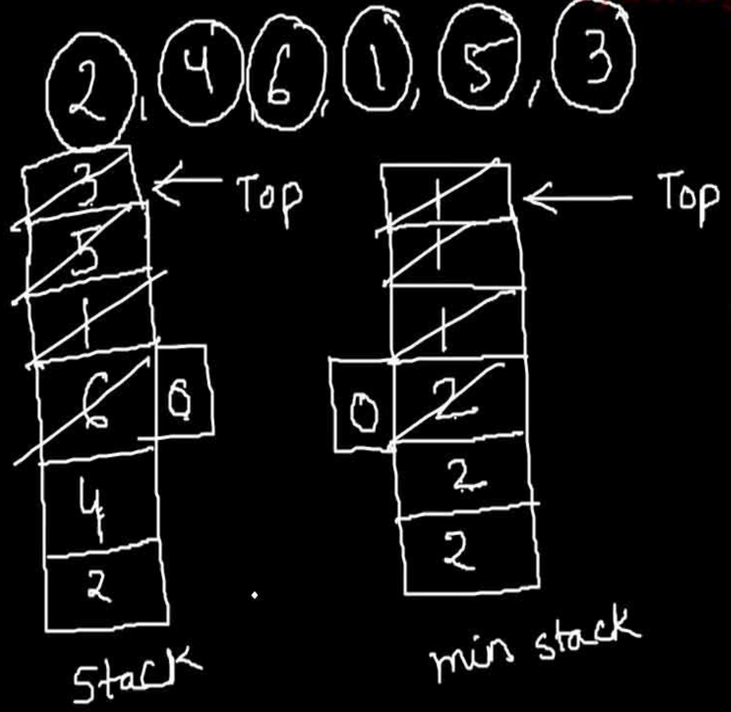
~~3~~

Stacks – Problems & Solutions

Problem - Design a stack such that GetMinimum( ) should be O(1)?

Solution - When we push the main stack, push either the new element or the current minimum, whichever is lower.

Main stack	Min stack
5 → top	1 → top
1	1
4	2
6	2
2	2



getmin( )

{ return minstack[Top];

5